

---

# SOFTWARE DESIGN

for

## Encost Smart Graph Project

Version 1.0

Prepared by: Student 1  
SoftFlux Engineer

SoftFlux

April 5, 2024

# 1 Introduction/Purpose

## 1.1 Purpose

The purpose of this document is to formulate the Software Design Specification (SDS) for the Encost Smart Graph Project (ESGP). This specification will present a high-level overview of the proposed system, detailing the software architecture and its primary components. It will aim to translate the features outlined by the Software Requirements Specification (SRS) into a coherent design plan.

## 1.2 Document Conventions

The following conventions are used in this document:

ESGP: Encost Smart Graph System

SDS: Software Design Specification

CLI: Command Line Interface

## 1.3 Intended Audience and Reading Suggestions

Developer: Primarily, this specification is intended for the developer, serving as an essential guide through the development process. It offers an in-depth look at the architectural framework and outlines the strategy required for successful implementation.

Tester: The tester is also a primary target of this document. They will need the SDS to develop tests that verify the system's ability to fulfill the software requirements without error.

## 1.4 Project Scope

Encost is a technology company that specializes in the development of home-based smart devices/appliances. The ESGP is a console application that allows users to visualize the use of the company's devices across New Zealand. This SDS will present the architectural structure of the ESGP system and the high-level interaction of its components. Additionally, this document will provide a comprehensive outline of each component, including the corresponding user interface.

## 1.5 Specialized Requirements Specification

This document operates under the assumption that Encost smart devices connect to hubs/controllers via Wi-Fi. The geographical location of each device is determined based on the region associated with the respective household ID.

## 2 Software Architecture

### 2.1 Component Diagram

The diagram below provides a high-level overview of the system's architecture. It consists of three main components: the CLI (Command Line Interface), the File, and GraphStream. The CLI component serves as the user interface, allowing users to interact with the application. The system relies on the File component, which represents the input data required for the application's operations. Finally, the GraphStream component represents an external Java library utilized for rendering graphical visualizations. The data processed by the application is used in conjunction with the GraphStream library to produce these visualizations.



Figure 2.1: Component Diagram

## 2.2 Activity Diagram

The activity diagram represents the system's control flow. The application begins by requesting the user type; the subsequent stage of execution is determined by the user type. If the user type is determined to be 'community,' the user is simply presented with the option of graph visualization. Otherwise, if the user type is 'encost,' the user will be prompted to enter their login details. The information is then verified by the system for security; if either the username or password is invalid, the user will be prompted to re-enter their information. Once the user's account has been verified, the feature options will appear. The features available to an encost member include uploading a custom dataset, viewing summary statistics, and accessing graph visualization. If the user opts to upload a custom dataset, the file format will be checked. If the format is incompatible, the user can either re-enter the file path or choose to use the default encost dataset provided.

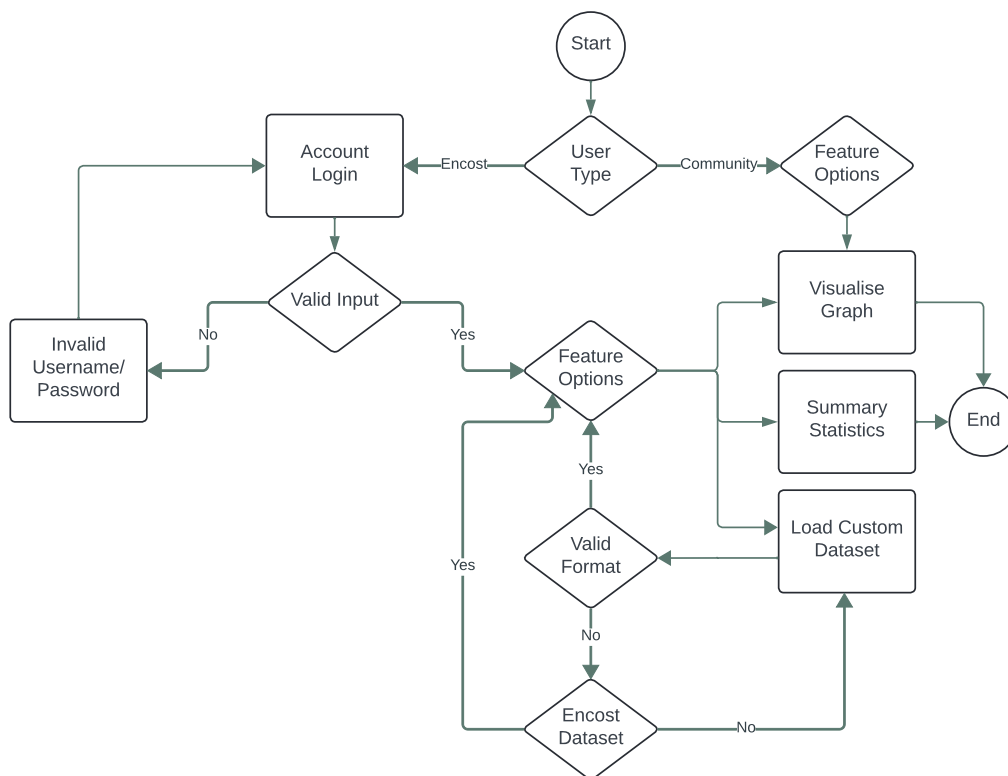


Figure 2.2: Activity Diagram

## 2.3 Class Diagram

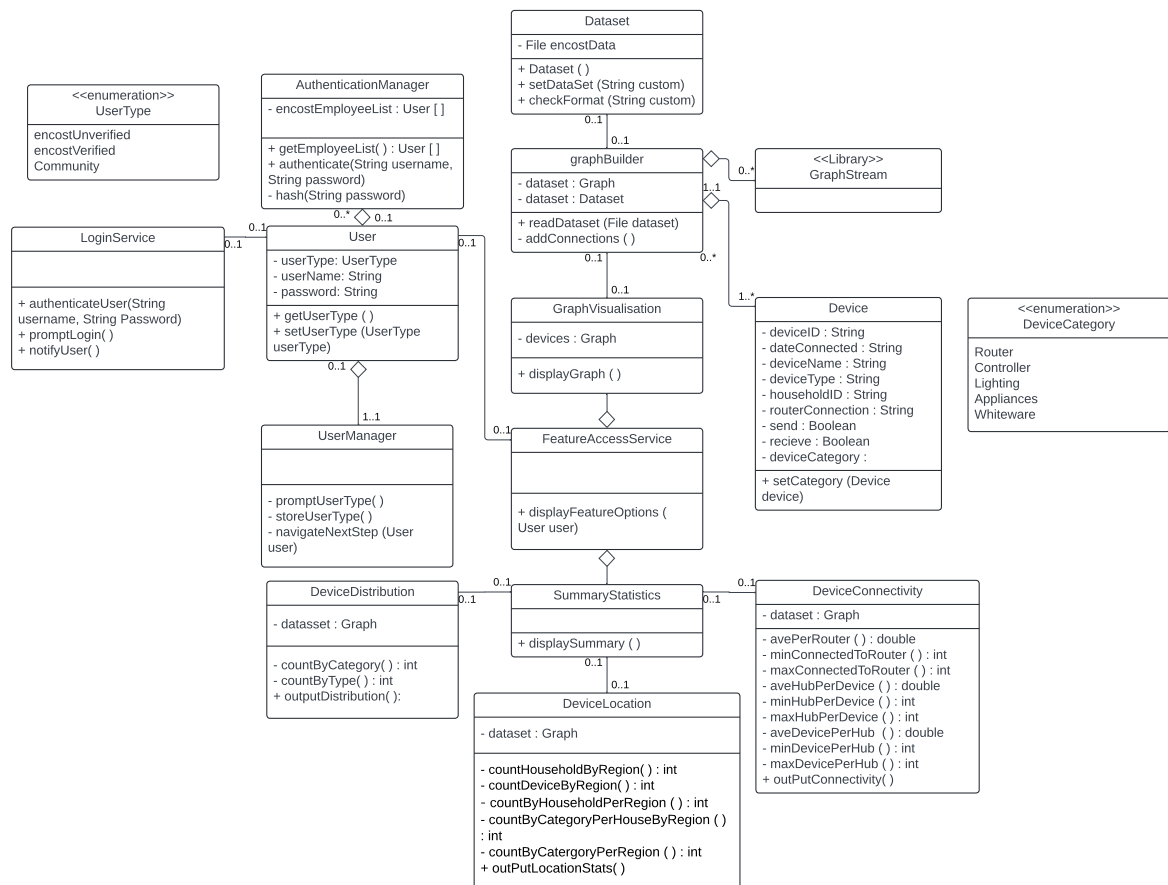


Figure 2.3: Class Diagram

### 2.3.1 User Class

#### Attributes

**userType** Stores the 'user-type' using the UserType enumeration.

**userName** Stores the username of an Encost employee as a string.

**password** Stores the hashed password of an Encost employee as a string.

#### Functions

**getUserType()** Retrieves the user type for the current user.

**setUserType(UserType userType)** Sets the user type for the current user.

### 2.3.2 UserManager Class

#### Functions

**promptUserType()** Displays a message to prompt the user to enter their 'user type'.

**storeUserType(UserType userType)** Stores the current user's 'user type' in a User object.

**navigateNextStep(User user)** Presents the subsequent prompt to the user, contingent on their 'user-type'.

### 2.3.3 LoginService Class

#### Functions

**authenticateUser(User user, String userName, String Password):** Checks the encost employee list for the username and then proceeds to authenticate the users password.

**promptLogin()** Displays a message to prompt the user to enter their username & password.

**notifyUser()** Displays a message to prompt the user to re-enter their information when the authentication for entered username or password fails.

### 2.3.4 AuthenticationManager Class

#### Attributes

**encostEmployeeList** Stores an array of User objects containing the information for ten Encost employees.

**password** Holds the input password of the current user as a string.

#### Functions

**authenticate()** Retrieves an array of User objects for ten Encost employees.

**hash(String password)** Applies a secure hash function to the current user's password input.

**authenticate()** Verifies the current user by comparing the provided password's hash against the stored hashed passwords. If the hash matches, the user's status is updated to 'encostVerified'.

### 2.3.5 GraphVisualization Class

#### Attributes

**graphDevices** Holds the dataset in a graph format.

#### Functions

**displayGraph()** Launches a window to visualize the graph.

### 2.3.6 FeatureAccessService Class

#### Functions

**displayFeatureOptions(User user)**

Presents the feature options menu corresponding to the 'user-type' of the current user.

### 2.3.7 SummaryStatistics Class

#### Functions

**displaySummary()** Displays summary statistics to user.

### 2.3.8 DeviceConnectivity Class

#### Attributes

**Graph graph** Stores each device from the dataset as a node within the graph data structure.

#### Functions

**avePerRouter()** Calculates the average number of devices connected to an Encost Wifi Router and returns the results as a double.



**minConnectedToRouter()** Calculates the minimum number of devices that an Encost Wifi Router is connected to, returning the result as an integer.

**maxConnectedToRouter()** Calculates the maximum number of devices that an Encost Wifi Router is connected to, returning the result as an integer.

**aveHubPerDevice()** Calculates the average number of Hubs/Controllers that a Smart Device is receiving commands from, returning the result as a double.

**minHubPerDevice()** Calculates the minimum number of Hubs/Controllers that a Smart Device is receiving commands from, returning the result as a integer.

**maxHubPerDevice()** Calculates the maximum number of Hubs/Controllers that a Smart Device is receiving commands from, returning the result as a integer.

**aveDevicePerHub()** Calculates the average number of devices that a Hub/Controller is sending commands to, returning the result as a double.

**minDevicePerHub()** Calculates the minimum number of devices that a Hub/Controller is sending commands to, returning the result as an integer.

**maxDevicePerHub()** Calculates the maximum number of devices that a Hub/Controller is sending commands to, returning the result as an integer.

**outputConnectivity()**

Displays the connectivity statistics in a clear and well-organized format.

### 2.3.9 DeviceLocation Class

#### Attributes

#### Functions

##### **countHouseholdByRegion()**

Calculates the number of households in each region of New Zealand, returning the result as an integer.

**countDeviceByRegion()** Calculates the number of devices in each region of New Zealand, returning the result as an integer.

##### **countByHouseholdPerRegion()**

Calculates the number of devices per household in each region of New Zealand, returning the result as an integer.

##### **countByCategoryPerHouseByRegion()**

Calculates the devices by category for each household across all regions of New Zealand, returning the result as an integer.

##### **countByCatergoryPerRegion()**

Calculates the number of devices by category across all regions of New Zealand, returning the result as an integer.

##### **outputLocationStats()**

Displays the device location statistics in a clear and well-organized format.

### 2.3.10 DeviceDistribution Class

#### Attributes

#### Functions

##### **countByCategory()**

Calculates the number of devices within each category.

##### **countByType()**

Calculates the number of devices of a specific type within each category.

##### **outputDistribution()**

Displays the distribution statistics in a clear and well-organized format.

### 2.3.11 GraphBuilder Class

#### Attributes

**Graph dataset** Represents a collection of devices within a graph structure.

**Dataset dataset** Stores the dataset to be used to build a graph.

#### Functions

**readDataset(File dataset)** Processes the dataset file line by line, generating new device objects to be integrated into a graph structure.

**addConnections()** Establishes connections among devices within the graph structure.

### 2.3.12 Dataset Class

#### Attributes

**File encostData** Stores the encost device dataset file.

#### Functions

**Dataset()** Initializes the dataset with the Encost data file.

**checkFormat(String filePath)** This method opens the file at the given file path and verifies that its format is correct.

**setDataset(String custom)**

**setDataset(String customPath)** Assigns the verified custom dataset file as the default dataset.

### 2.3.13 Device Class

#### Attributes

**deviceID** A unique identifier for the device, such as a serial number, MAC address, or any other unique code, used to distinguish this device from others in the network.

**dateConnected** The date and time when the device was first connected to the network, useful for tracking the device's tenure or for maintenance schedules.

**deviceName** A human-readable name for the device, which could be a default name based on the device type or a custom name assigned by the user or administrator.

**deviceType** Specifies the category of device (e.g., smartphone, laptop, smart thermostat, IoT device), assisting in network management and policy application.

**householdID** An identifier that links the device to a specific household or location, enabling location-based management and policy enforcement.

**routerConnection** Records the unique identifier (device ID) of the router to which it is currently connected, facilitating network management and connectivity tracking.

**send** The amount of data sent from the device to the network over a specified period, useful for monitoring network usage and identifying heavy senders.

**recieve** The amount of data received by the device from the network over a specified period, instrumental in monitoring network usage and bandwidth needs.

**deviceCategory** A broader classification of the device, such as 'personal device', 'home appliance', 'entertainment', or 'security device', which can be used for setting policies and permissions.

#### 2.3.14 Functions

**setCategory()** A function to assign or update the 'deviceCategory' attribute based on predefined criteria or user input, facilitating dynamic management and categorization of devices.

## **2.4 Performance Considerations**

### **2.4.1 Response Time**

If a dataset load, graph visualization display, or summary statistics calculation exceeds a 1-second delay, the application will provide immediate feedback to the user. A notification will appear in the command line, informing the user that the process is underway. To guarantee that these processes do not surpass a 10-second waiting period, tests will be conducted during the implementation phase to verify adherence to this requirement.

## **2.5 Dependencies**

This application is dependent on the GraphStream library, a Java framework designed for graph modeling, analysis, and visualization. GraphStream is essential to the application's operation, allowing for the efficient storage of data within a graph data structure. This enables not only data analysis but also graphical visualizations of the stored information.

## **2.6 Security Considerations**

### **2.6.1 Enhanced Authentication Measures**

To securely grant Encost employees additional access privileges, a login mechanism will be implemented. Ten distinct username and password pairs will be allocated for authenticated access to the application.

### **2.6.2 Password Security**

To safeguard user credentials, passwords are encrypted using a hashing function before integration into the application. This process involves generating a cryptographic hash of each password, a method that converts the original password into a fixed-size string of

characters, which is virtually impossible to reverse-engineer. These hashes, along with their corresponding usernames, are securely stored within the system's internal database.

### **2.6.3 Hash Verification Process**

Upon login attempts, the password provided by the user is subjected to the same hashing process. The resulting hash is then compared to the stored hash associated with the user's username. This comparison ensures that access is granted only if the hashes match, thereby authenticating the user's identity without storing or transmitting the actual password.

## 3 Component Design

### 3.1 Sequence Diagrams

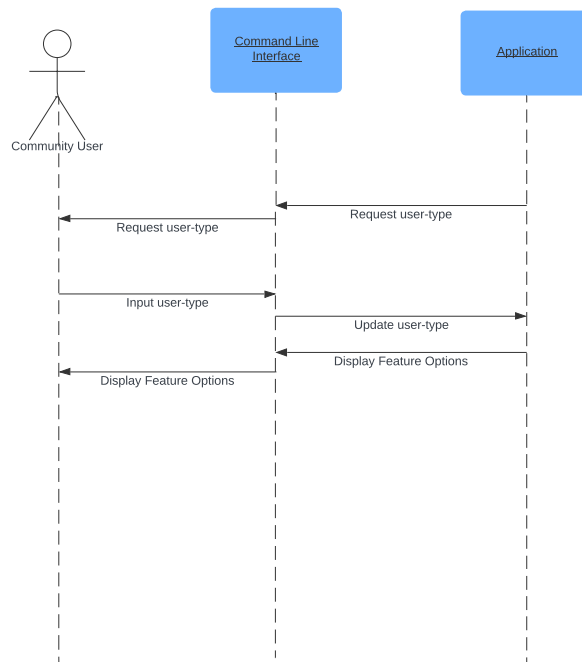


Figure 3.1: User-type Prompt Sequence Diagram

#### 3.1.1 User-type Prompt Sequence Diagram

This sequence diagram illustrates the series of events that occur at the start of the program. Initially, the application displays a welcome message, prompting the user



to enter their user type. Upon receiving the user type, the application updates the user's type to 'community'. If the user is an Encost employee, the 'user type' is set to 'encostUnverified'. After updating the user type, the application proceeds to the next prompt, which is determined by the user's type. In this scenario, as the user is identified as a community user, the application will present feature options that include 'graph visualization' only.

### 3.1.2 Employee Login Prompt Sequence Diagram

This diagram illustrates the sequence of events initiated by a user identified as an Encost Employee. Upon starting, the application requests the user's type. After updating this information, it prompts the user for their username and password. The user then provides their account information in response. The application verifies if there is a user with the given username in the employee list. Subsequently, it hashes the entered password to compare it with the stored password associated with that username. If the passwords match, the application proceeds to the next step. If not, it alerts the user that the username and/or password failed authentication and requests the re-entry of their account information.

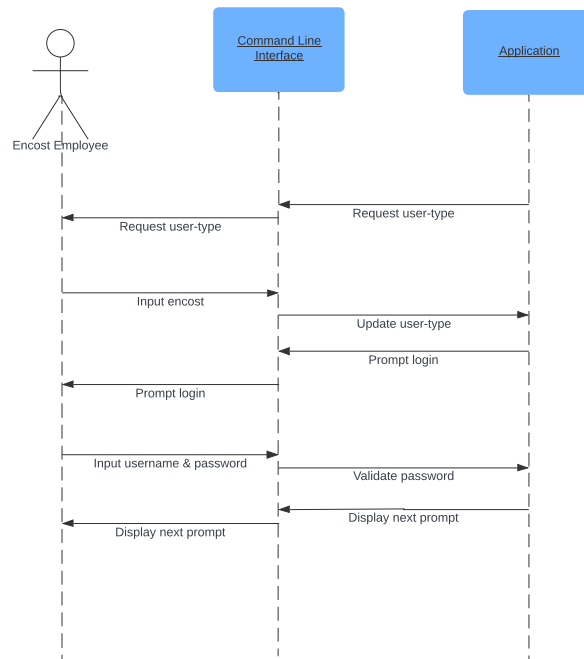


Figure 3.2: Employee Login Prompt Sequence Diagram

### 3.1.3 Graph Visualization Selection Process Sequence Diagram

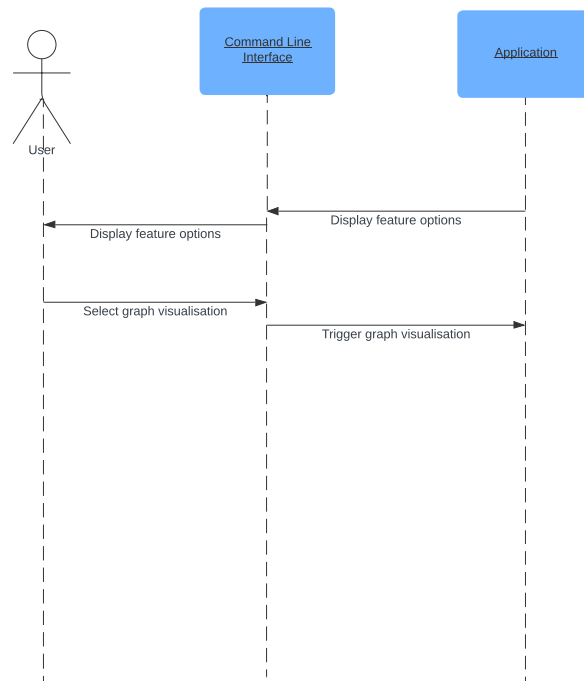


Figure 3.3: Graph Visualization Selection Process Sequence Diagram

This diagram illustrates the sequence of events that unfolds when a user selects "Graph Visualization" from a list of feature options, a process relevant to all users. After completing the preceding activity, the application presents the feature options to the user. Upon the user's selection of "Graph Visualization," the graph visualization process is initiated, leading to the opening of a new user interface (UI) where the graph is displayed.

### 3.1.4 Load Custom Dataset Selection Process Sequence Diagram

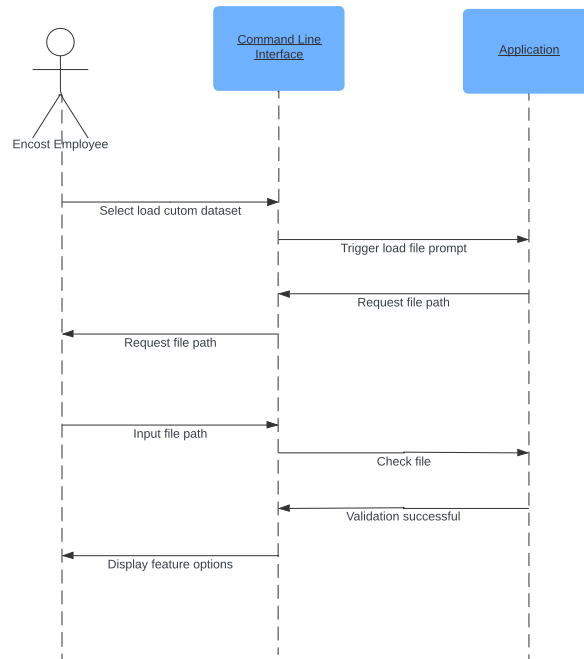


Figure 3.4: Load Custom Dataset Selection Process Sequence Diagram

This diagram illustrates the steps an Encost employee takes to load a custom dataset. The process begins when the user selects "Load Custom Dataset" from the feature options, prompting the application to display a file load dialogue that asks the user to enter the file path for the dataset they wish to upload. Upon receiving the file path, the application verifies the file's formatting to ensure compatibility for processing. If the file meets the necessary criteria, the application confirms the successful upload to the user by revisiting the feature options screen. Conversely, if the file format is incompatible, the application notifies the user and provides guidance to re-enter the file path. Additionally, it offers the option to proceed with a default dataset instead.

### 3.1.5 Summary Statistics Selection Process Sequence Diagram

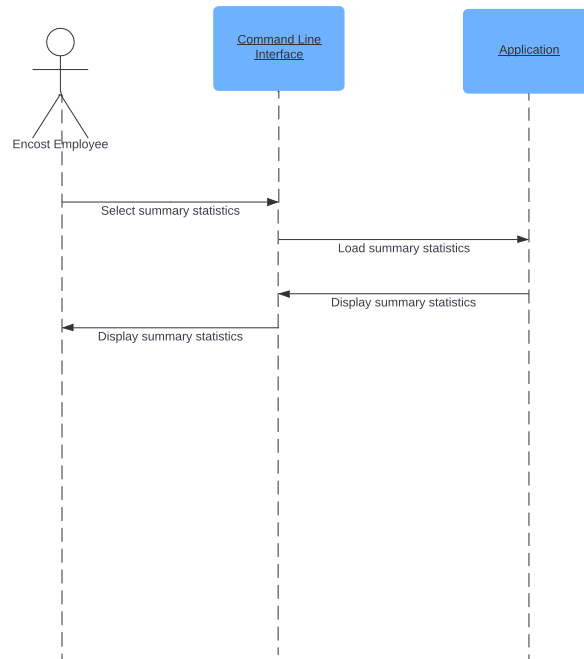


Figure 3.5: Summary Statistics Selection Process Sequence Diagram

This diagram illustrates the sequence of steps a user takes to select summary statistics from the feature options menu. Upon selecting summary statistics, the application begins loading them, a process that involves performing various calculations using the graph data structure. Once these calculations are complete, the application displays the summary statistics on the command line interface.

## 3.2 Use-case Diagram

This use-case diagram highlights the distinctions between two user categories: community users and Encost employee users. Community users possess fewer privileges compared to Encost employees. Encost employee users are required to log in, a process that ensures a necessary level of authentication for accessing more advanced application features. Unlike Encost employees, community users cannot load custom datasets or view summary statistics.

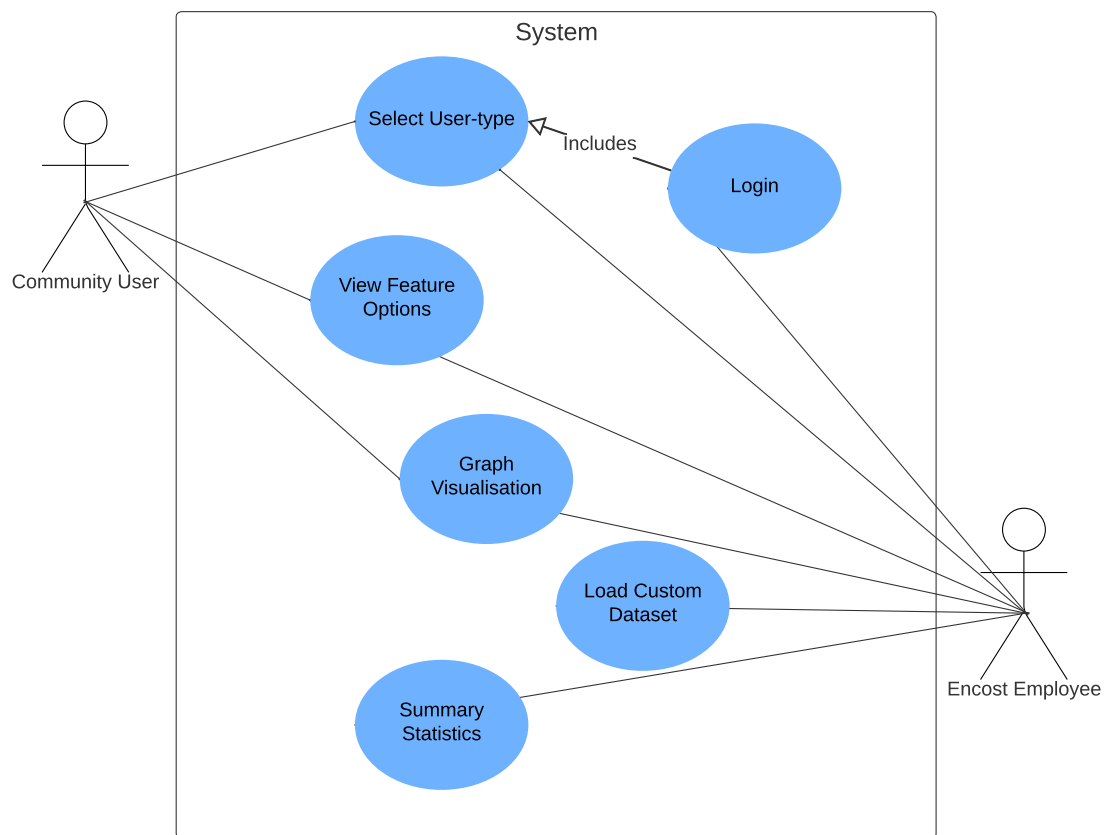


Figure 3.6: Use-case Diagram

## 3.3 User-Interface Design

### 3.3.1 Welcome prompt Wireframe Diagram

This application features a command-line user interface. Upon launching the application, users are greeted with a welcome message and prompted to specify their user type by entering either 'encost employee' or 'community'.

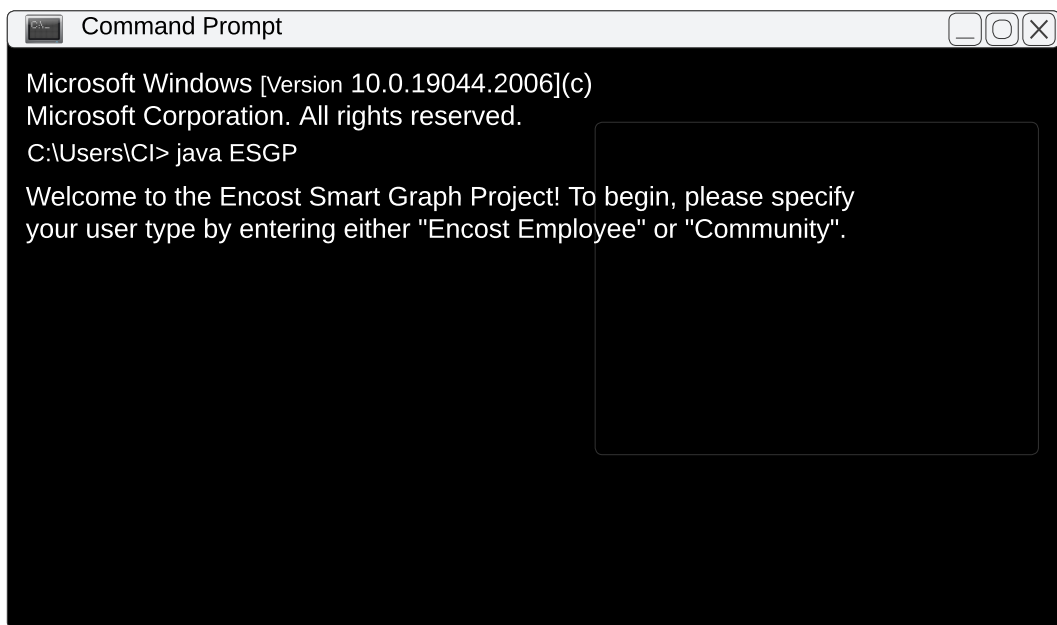


Figure 3.7: Welcome prompt Wireframe Diagram

### 3.3.2 Encost Employee Login Prompt Wireframe Diagram

This wireframe diagram illustrates the interface presented to an Encost employee upon specifying their user type. The application prompts the user through the command line interface to enter their username and password. Upon inputting their credentials, the application verifies their account and advances them to the subsequent phase.

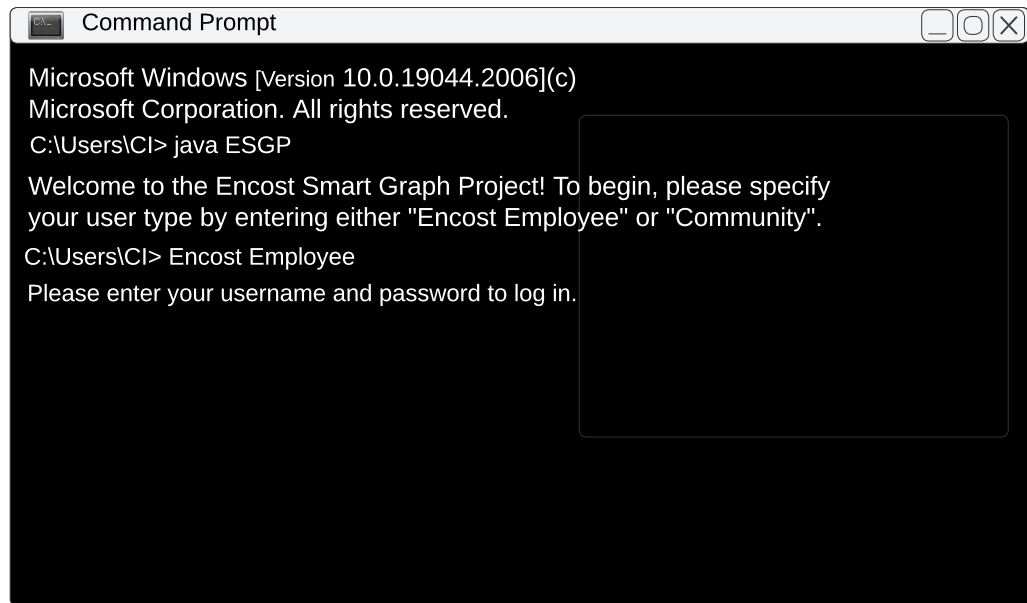
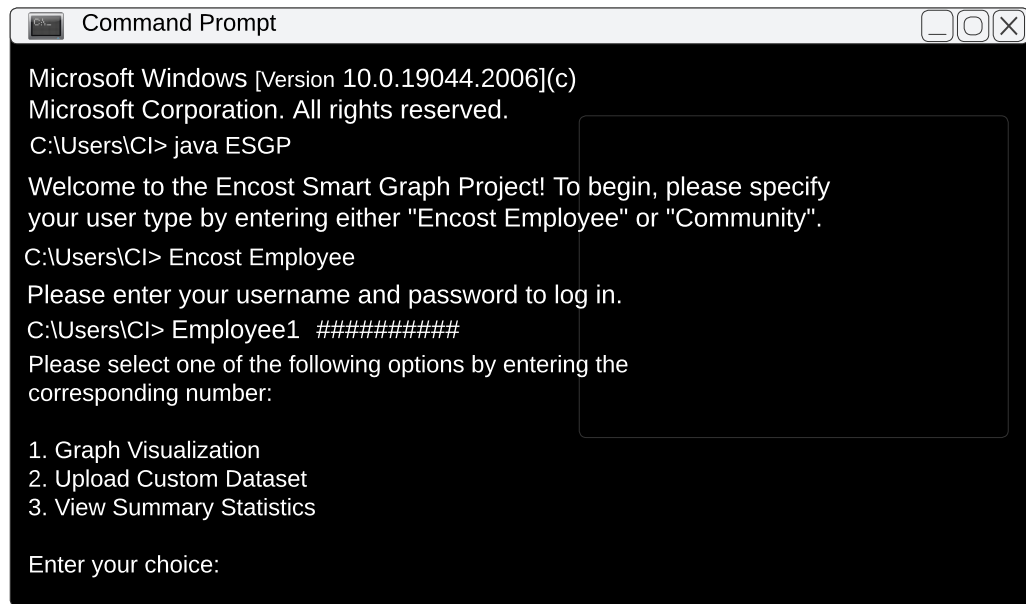


Figure 3.8: Employee Login Wireframe Diagram



### 3.3.3 Encost Employee Feature Options Wireframe Diagram

This wireframe diagram illustrates the display presented to an Encost employee after successfully entering their username and password. The application then shows available feature options and prompts the user to select one.



```
Microsoft Windows [Version 10.0.19044.2006](c)
Microsoft Corporation. All rights reserved.
C:\Users\CI> java ESGP

Welcome to the Encost Smart Graph Project! To begin, please specify
your user type by entering either "Encost Employee" or "Community".
C:\Users\CI> Encost Employee
Please enter your username and password to log in.
C:\Users\CI> Employee1 #####
Please select one of the following options by entering the
corresponding number:

1. Graph Visualization
2. Upload Custom Dataset
3. View Summary Statistics

Enter your choice:
```

Figure 3.9: Encost Employee Feature Options Wireframe Diagram

### 3.3.4 Graph Visualisation Wireframe Diagram

The wireframe diagram for the Graph Visualization User Interface presents a simplified depiction of the graph. Upon selection, the application launches a new UI window that presents the graph visualization. Labels are fixed to the graph, indicating devices that can send and those that can receive. Additionally, the nodes are color-coded to differentiate between different device categories.

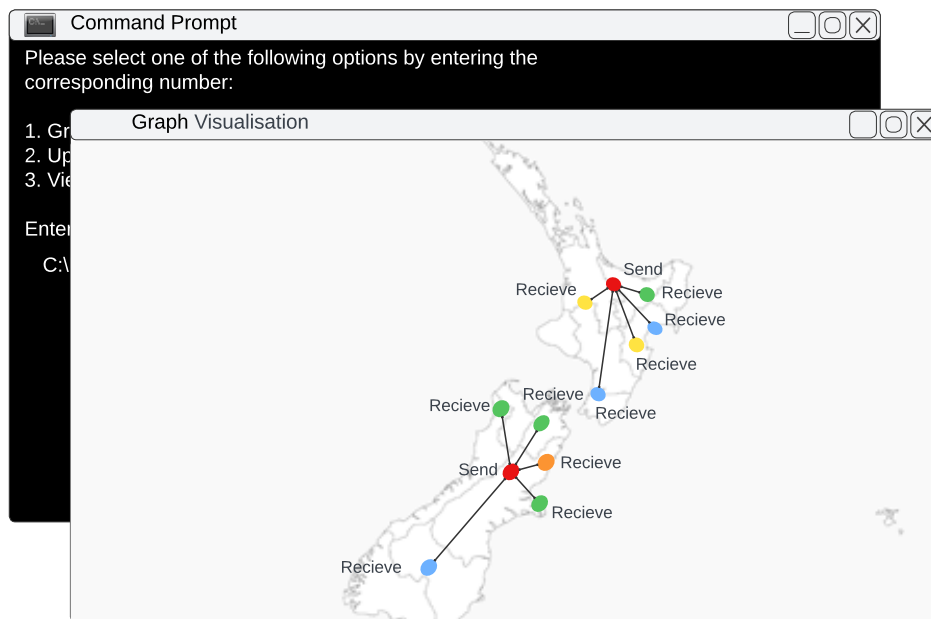


Figure 3.10: Graph Visualisation Wireframe Diagram

### 3.3.5 Upload Custom Dataset Prompt Wireframe Diagram

This wireframe diagram illustrates the output generated by the application when an Encost employee user opts to upload a custom dataset. The application prompts the user to input the file path for the dataset they intend to upload. It then verifies the file and notifies the user about its compatibility.

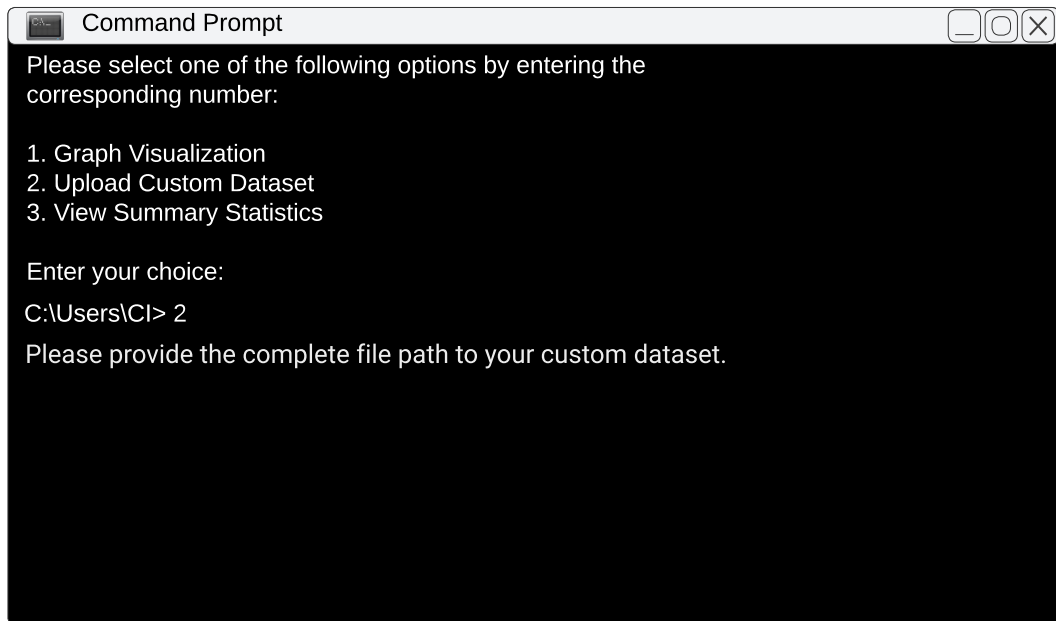


Figure 3.11: Upload Custom Dataset Wireframe Diagram

### 3.3.6 Summary Statistics Wireframe Diagram

This diagram illustrates the output presented when the user opts to view the summary statistics. The output is neatly formatted to enhance readability.

```
Command Prompt
SUMMARY STATISTICS OUTPUT
Distribution:
-----
Device Categories Summary:
- Smartphones: 320 devices
- Laptops: 215 devices
- Smart TVs: 89 devices
- IoT Devices: 560 devices
Device Type Distribution:
- Smartphones: [Android: 220, iOS: 100]
- Laptops: [Windows: 150, MacOS: 65]
- Smart TVs: [Android TV: 45, Other: 44]
- IoT Devices: [Thermostats: 150, Smart Lights: 410]
Location (New Zealand Regions):
-----
Households Summary:
- Auckland (NZ-AUK): 150 households
- Wellington (NZ-WGN): 120 households
- Canterbury (NZ-CAN): 90 households
Devices by Region:
- Auckland (NZ-AUK): 1020 devices
- Wellington (NZ-WGN): 850 devices
- Canterbury (NZ-CAN): 680 devices
Devices per Household by Region:
- Auckland (NZ-AUK): Average 6.8 devices/household
- Wellington (NZ-WGN): Average 7.1 devices/household
- Canterbury (NZ-CAN): Average 7.5 devices/household
Device Category Distribution per Region:
- Auckland (NZ-AUK): [Smartphones: 300, IoT Devices: 720]
- Wellington (NZ-WGN): [Laptops: 200, Smart TVs: 80]
- Canterbury (NZ-CAN): [Smartphones: 180, IoT Devices: 500]
Connectivity:
-----
Encost Wifi Router Connections:
- Average devices connected: 5
- Min/Max devices connected: 1 / 10
Encost Hubs/Controllers to Smart Device Connections:
- Average Hubs/Controllers per Smart Device: 2
- Min/Max Hubs/Controllers per Smart Device: 1 / 4
Smart Devices to Hubs/Controllers Connectivity:
- Average Smart Devices per Hub/Controller: 8
- Min/Max Smart Devices per Hub/Controller: 3 / 15
```

Figure 3.12: Summary Statistics Wireframe Diagram

## 4 Conclusion

This Software Design Specification for the Encost Smart Device Project outlines the application's design, including its architecture and the detailed design of its components. The component diagram offers a high-level view of the system's structure, while the activity diagram reveals insights into the system's control flow. The class diagram serves as a blueprint for the internal structure, guiding developers on how to implement the system. Additionally, the sequence diagrams detail the order in which activities will be accessed by users. The wireframe diagrams visually demonstrate user interaction with the system, and the use case diagram provides an overview of the system's use cases. Together, these elements form a robust foundation for the successful development and deployment of the Encost Smart Device Project.