# TFL SHORTEST WALKING DISTANCE

Ifeoma Okafor, Yusuf Jimoh and Bethelham Demissie

# Introduction

Aim: Find the fastest walking route between any two stations within the TFL Zone 1 train lines using Dijstrka's Shortest Path Algorithm

Group: Three members

We developed two versions of the application:

- Version 2 - Hand coded using core C# classes
- Version 3 - Modified Version 2 using .NET library classes

# Overview of the Task

Used a GitHub repository when working on the implementation to share code and work collaboratively on the project. analysis.

The program is able to handle incidents that affect the walking route such as road closures, and delays.

We conducted application testing,  benchmark testing and consistency testing to ensure that our both versions run correctly, we recorded our findings from the analysis.

In this presentation, we will provide an overview of our groups work, providing an explanation on the design of the two versions developed, our testing and benchmarking as well as our finding from the project.

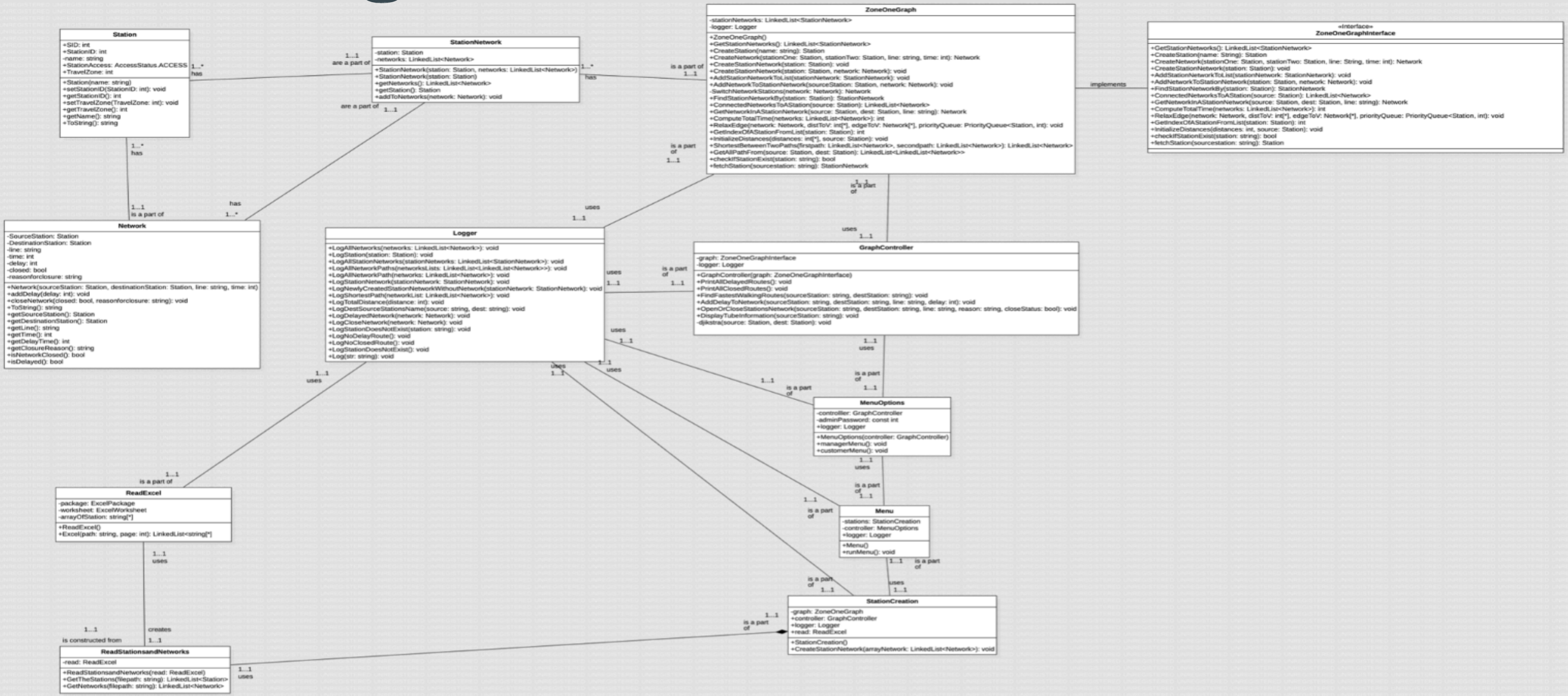
# Our Approach

Use a excel sheet found on the TFL website as a source for our stations and networks as well as weights, we considered this a scalable approach as it would be easier to modify the excel sheet then the hard code new station.

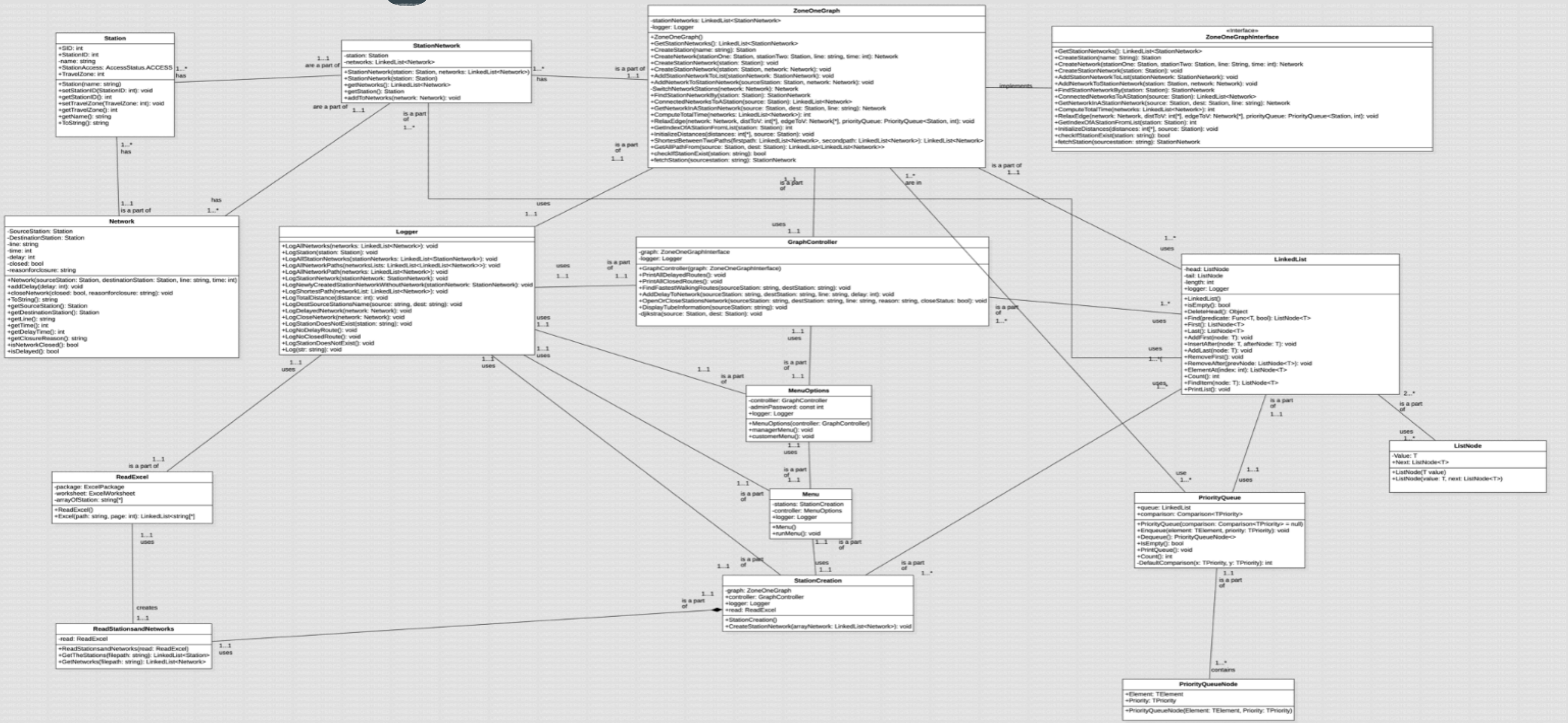
Our approach was to create a ‘StationNetwork’ class where we create the networks between the stations. This was done to ensure that the creation of vertices and edges for the graph were done faster.

We also used a Linked List for efficient insertion. The advantage of this is that it is easier to work with as we can add to both head and tail easily and there’s no memory wastage.

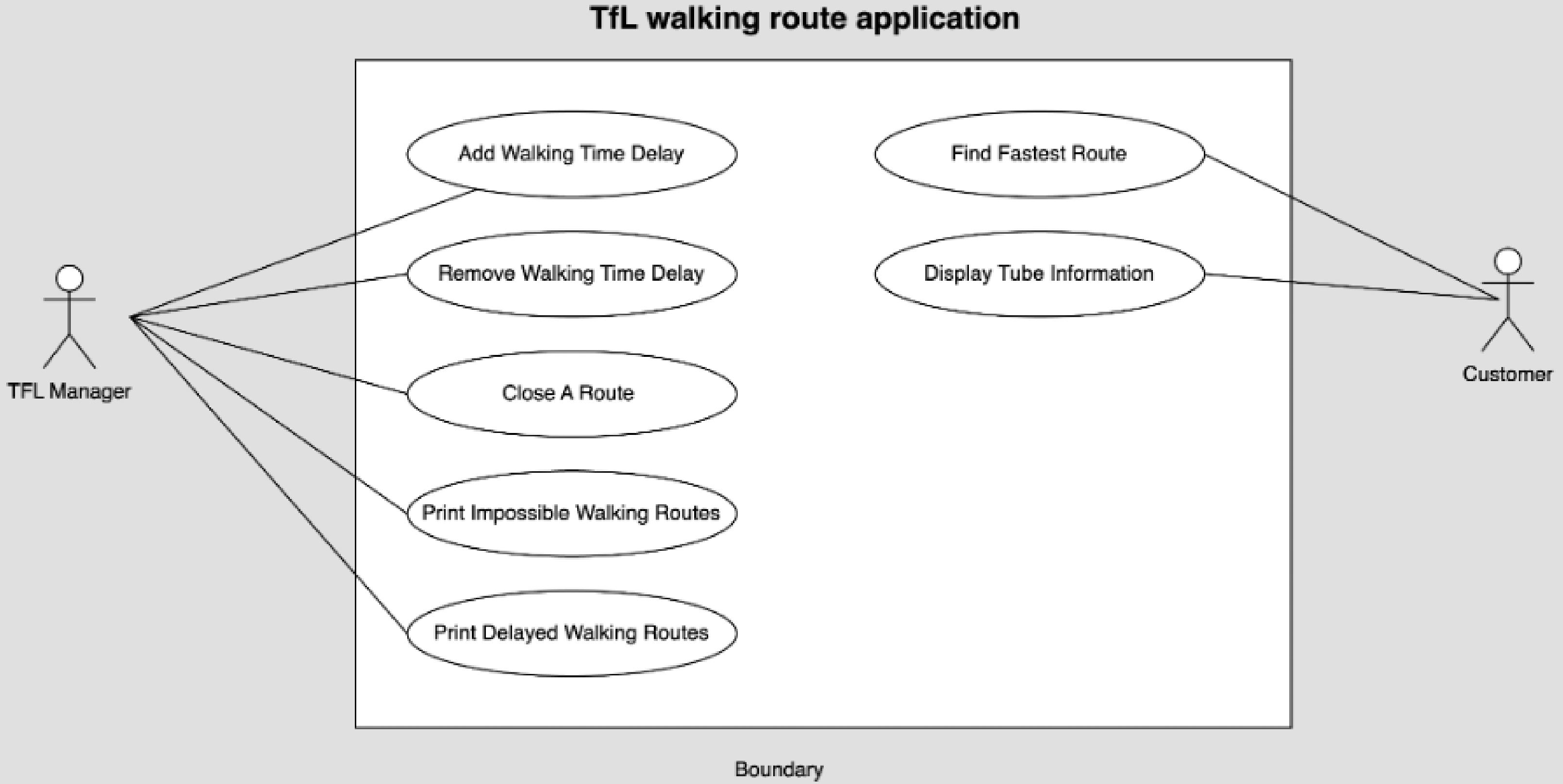
# UML Design - .NET



# UML Design - Hand Coded



# Use Case Diagram



# Implementation

StationNetwork:

- Represents a network of stations and contains a Station object and a LinkedList of Network objects
- It provides the program with methods to get the LinkedList of networks, get the Station object, and add a Network object to the networks LinkedList.

ZoneOneGraph:

- Represents a graph of stations and networks using a linked list data structure.
- Provides methods for creating and adding stations/networks, finding connected networks, computing total time, and pathfinding.

GraphController:

- Implements operations on the tube network graph using a ZoneOneGraphInterface.
- Provides methods for finding walking routes, adding delays, and opening/closing stations, and uses PriorityQueue and Djikstra's algorithm for pathfinding.

# Testing / Analysis

```
Beginning of Benchmarking Tests
-------------------------------

Test Name                       Station(s)                      VersionTwo Elapsed Time (ms)    VersionThree Elapsed Time (ms)
Dijsktra                        Baker Street - Goodge Street    16.8945                         16.6053
Dijsktra                        Temple - Lambeth North          10.7551                         14.9966
Dijsktra                        Green Park - Liverpool Street   10.913                          11.9925
Dijsktra                        Marble Arch - Cannon Street     15.6795                         11.0997
Dijsktra                        London Bridge - Marylebone      11.801                          16.6458
Dijsktra                        Paddington - Tower Hill         17.9979                         11.6504
Getting Tube Information        Tower Hill                      0.099                           0.0944
Add Delay To Network            Oxford Circus - Bond Street     0.1628                          0.1932
Print Closed Routes                                             0.31                            0.2943
Create Adjacency List                                           965.8164                        901.7815

Average ELapsed Time (FindShortestPath) for VersionTwo: 14.006833333333335(ms)
Average ELapsed Time (FindShortestPath) for VersionThree: 13.831716666666667(ms)

-------------------------------
End of Benchmarking Tests
```

# Comparison

Version two uses hand-coded data structures and algorithms, while version three uses software available in the .NET Framework Libraries for implementing Dijkstra's shortest path algorithm.

Version two implements the adjacency list using custom LinkedList, ListNode, PriorityQueueNode, and PriorityQueue classes, making the implementation relatively complex.

Version three simplifies the implementation by replacing custom data structures with built-in LinkedList and PriorityQueue data structures, making the code easier to understand and modify.

Version three's use of built-in libraries results in better performance, as they are optimised for performance and have been extensively tested and refined.

# Conclusion

Both versions implement Dijkstra's shortest path algorithm

Version three is a better implementation overall

Hand-coded data structures in version two are complex and prone to errors

Version three's implementation is simpler, more standardised, and more likely to perform better

We recommend using version three's implementation in future projects

# Thank you
# Any Questions?