Internal Facility Management Software

Computer Science NEA

Name: *Benjamin Thomas Ellison*
Candidate Number: *5828*

Centre Name: Barton Peveril College
Centre Number: 58231

0 Contents

0 Contents **2**

1 Analysis **5**

1.1 Statement of Problem 5

1.2 Background 5

1.3 End User 5

1.4 Initial Research 5

1.4.1 Existing, similar programs 5

1.4.2 Potential abstract data types / algorithms 6

1.4.2.1 Graph traversal (depth, breadth, etc.) to find the shortest route between two places 6

1.4.2.2 Priority queues so that clients can be ordered by FIFO by default, unless they are priority clients 6

1.4.2.3 SQL database to keep information about timings, distances, clients, etc. 7

1.4.3 First interview 7

1.4.4 Key requirements 8

1.5 Further Research 8

1.5.1 Prototype 8

1.5.1.1 Thesis: 8

1.5.1.2 Code: 9

1.5.1.3 Reflection on Prototype 13

1.6 Objectives 14

1.7 Modelling 15

2 Design **17**

2.1 Functional Components 17

2.1.1 Companies 17

2.1.2 Users 17

2.1.2.1 Admins 17

2.1.2.2 Employees 18

2.1.2.2.1 Qualifications 18

2.1.3 Teams 19

2.1.4 Region 20

2.1.5 Clients 21

2.1.6 Jobs 22

2.1.6.1 Jobs (Unregistered) 22

2.1.6.2 Jobs (Unassigned) 23

2.1.6.3 Jobs (Assigned) 24

2.1.6.4 Jobs (Completed) 24

2.2 Database Entity Relationships 26

2.3 User System 26

2.4 Menu Structure 27
2.4.1 Main Menu 28
2.4.2 Menu E 29
2.4.3 Menu A 30
2.4.4 Menu Pre T and Menu T 31
2.4.5 Menu R 32
2.4.6 Menu Q 33
2.5 Algorithms 33
2.5.1 Inputs 34
2.5.1.1 Matrices (Regions) 34
2.5.1.2 Nodes & lists 34
2.5.2.1 Demonstration 35
2.5.2.2 Efficiency 35
2.5.4.3 Next Location Pathfinding 36

3 Testing 36

4 Evaluation 38
4.1 Reflection 38
4.2 End-User Feedback 40
4.3 Improvements 40
4.3.1 Database changes 40
4.3.2 UI changes 40

5 Technical Solution 41
Contents page for code 42
Code 43
5.1 Classes 43
5.1.1 User Class 43
5.1.2 Node Class 43
5.2 SQL connection setup 47
5.3 SQL injection prevention using RegEx 49
5.4 Matrix Subroutines 49
5.4.1 Matrix Creation 49
5.4.2 Matrix Validation 50
5.5 Dijkstra's Algorithm 52
5.5.1 ID-Integer 2-way conversion for dijkstra 54
5.5.2 Dijkstra demonstration initiation 54
5.5.3 Dijkstra algorithm in practice 56
5.6 Company Creation 59
5.7 User Management 62
5.7.1 Creation 62
5.7.2 Credential Checker 63
5.8 Client Creation 64
5.9 Team management 65
5.9.1 Creation 65

5.9.2 Viewing and editing 67
5.10 Region Creation 68
5.11 Job Management 74
5.11.1 Creation 74
5.11.2 Marking Completion 76
5.11.3 Viewing 78
5.11.3.1 Employee level 78
5.11.3.2 Admin Level (Productivity Check) 79
5.12 Qualification management (Viewing and Assigning) 82
5.13 Efficiency Algorithm 86
5.14 Top Level Menus (Main, A & E) 87
5.15 SQL Database Creation Statements 91

1 Analysis

1.1 Statement of Problem

Many professions depend on the professional's ability to arrive at multiple locations within a day, which inevitably begets transport. Transport has its own mess of traffic, route-finding, and areas of operation, and having to take into account visiting multiple clients, multiple professionals and keeping everyone satisfied and running optimally, requires a level of precision that is, suffice it to say, beyond just human management. This can provide issues both financially and in regards to satisfaction of clients, employees and administrators.

1.2 Background

The issue of poorly managed transport routes is often caused by people all arriving at different times and taking different times to do the tasks they do. This combined with different skill sets and areas of origin make the routes, if managed by an unassisted dispatcher, or even more so if just left to the professionals will inevitably lead to potential errors over poor communication, or at the very least suboptimal routes. This can cause unnecessary delays for clients, and unnecessary fuel costs and time spent on the road, which are expensive and dangerous respectively. As a result of all these factors, suboptimal professional pathfinding is a lose-lose-lose situation.

1.3 End User

My end user would be a mobile professional who works in a team. In order to have a more advanced scope of the issue, I will be asking Stuart, a mobile engineer who has used a similar app as the one I am looking to create. He will give me some perspective and advise me on which features could be implemented, avoided, or improved upon from other similar programs

1.4 Initial Research

1.4.1 Existing, similar programs

Google Maps
Google maps is akin to the pathfinding aspect of this program. It uses A* search (along with a neural net predicting traffic issues and route speed, but that's excessive for these purposes) to find the optimal route from one place to another in the shortest possible time. This will end up being very similar to my pathfinding aspect, since it will likely also use A* search.
Sources: https://www.quora.com/How-does-the-algorithm-of-Google-Maps-work

https://www.quora.com/What-algorithms-are-used-by-Google-Maps-to-make-route-finding-so-fast

Joblogic
Joblogic has three aspects that my program will need, being a timetable that logs jobs, including the detail of said jobs, the ability to track and log job completion, and a database full of addresses, so that professionals can know where they need to go.

Source: https://www.joblogic.com/

1.4.2 Potential abstract data types / algorithms

1.4.2.1 Graph traversal (depth, breadth, etc.) to find the shortest route between two places

Graph traversal would be particularly essential in regards to the pathfinding aspect of this program. There are multiple ways one can go about this, each with varying degrees of efficiency.
Depth-first traversal (searching each path until its conclusion consecutively) would require strict limiting of area size in order to not be grossly inefficient, however a great deal of efficiency would likely be lost since with an area of any respectable size, many roads diverge in many ways, leading to many time consuming route calculations that may not even come close to the location you want to arrive at, and if it does, it may not be the optimal way of getting there.

Breadth-first traversal (searching all paths simultaneously in a circular motion) would be an algorithm that is more suited to an area of a map than depth traversal due to its inability to “chase down rabbit holes” so to speak, as well as its consistency in time consumption, however it still wastes a significant amount of resources checking routes that are not in the direction of the destination, leading to only a small amount of the time going to “useful” calculations. It will, however, find the shortest path.

Dijkstra's algorithm, an algorithm that takes into account the cost of each node to be travelled to from the present node and uses this to calculate the distance of each node from one another. This makes it a best-first traversal algorithm, which is best suited of all the traversal algorithms I've seen to this task. It requires an adjacency matrix or adjacency graph in order to function, so programming one of the two of those will be necessary

1.4.2.2 Priority queues so that clients can be ordered by FIFO by default, unless they are priority clients

Queues, from the get go, are best suited to this task, since the alternative, a stack, would be LIFO (last in, first out), and generally, that does not gel well with clients’ views on fairness. Queues, on the other hand, are FIFO (first in, first out), which is more culturally understood as a way of managing clients.

As well as queues being the optimal sorting method, a priority system would definitely be called for, since in many professions, some jobs are more time-sensitive

than others (for example, emergency paramedics will need to get to high risk of death situations faster than low risk of death ones). Therefore a priority queue is in order

1.4.2.3 SQL database to keep information about timings, distances, clients, etc.

A lot of jobs require a log of tasks completed for statistical analysis. This sort of info would be helpful to store in a database, and in order to utilise this database to the best of its ability, SQL (search query language) would be helpful to navigate this log. It may also be helpful to log the amount of travel time and task completion rate, so that an average of the two may be calculated for a personal routine to be created.

1.4.3 First interview

Pre-interview: Program is described as "A program that guides teams of mobile professionals to to their client (based on priority and first-come-first-serve) via the fastest available route, bearing in mind their specialities, and keeps track of what jobs you've completed, and are going to complete."

1. What do you think are the core, most important features of a route-planning program?

The program must know average travel times and assign calls to professionals based on whether or not they can get there within a set response time.
On top of this it must be aware of specialised training professionals have.

2. If you have experienced programs that do similar things (pathfinding, routine management), what were their pitfalls?

If an engineer gets held up then this can confuse the system and result in 2 engineers going to the same postcode to cover another machine.

3. What, to you, are some non-core, extra features that you think may be helpful?

Average point to point travel time
On-site time
Expected End time

4. What features do you think are important to allow user manoeuvrability on?

Ability to change expected end time.

5. What are some of the problems you've experienced with human dispatching methods?

Not aware of geography and what engineers are trained on.

5b .How do you think automation can resolve some of these issues?

Using postcodes and a database of levels of training .

6. What other time management hurdles may this program encounter?

Traffic jams, specific people being off sick. Clients who've booked jobs that require more time than can fit into a day.

Conclusion:

The program needs to specifically account for geographical knowhow, collisions in planning, and potential overflows in time management to be compliant with fixed-hour contracts. It is also important that the machine doesn't overestimate the speed professionals work at and promise clients unfulfillable response speed. The program also needs to be flexible in real time when a planned action cannot be fulfilled.

1.4.4 Key requirements

Judging by the thesis and interview, the main key objectives are as follows:
A pathfinding algorithm that takes professionals via the best route using average travel time.
A User-Interface that allows for different levels of modification by different parties.
A database logging addresses and details of all involved parties.

1.5 Further Research

1.5.1 Prototype

1.5.1.1 Thesis:

For my prototype I wanted to determine whether dijkstra's algorithm or an A* search algorithm would be best for optimal pathfinding. According to my research, they are both

virtually identical, bar the fact that Dijkstra's algorithm lacks the heuristic algorithm A* search uses. As a result of this, i will code a version of Dijkstra's algorithm, and test its efficiency without the heuristic algorithm that would make it A* search, in order to see if it would be viable, and how it would scale when nodes are added/removed

1.5.1.2 Code:

```
Module Module1

    Public Class Node
        Private Neighbourhood As New List(Of Char)
        Private costFromStart As Integer
        Private name As Char
        Private PreviousNode As Char


        Sub New(ByVal identity As Char, ByRef town As Integer(,), ByRef
townSize As Integer)
            name = identity
            Dim whereInMatrix As Integer = Asc(identity) - Asc("A")
            If whereInMatrix <= townSize And whereInMatrix >= 0 Then
                For i = 0 To townSize - 1
                    If town(whereInMatrix, i) <> Integer.MaxValue Then
                        If Chr(Asc("A") + i) <> name Then
                            Neighbourhood.Add(Chr(Asc("A") + i))
                        End If
                    End If
                Next
            End If
            costFromStart = Integer.MaxValue
            PreviousNode = "_"
        End Sub

        Sub NodeCopy(ByVal Swap As Node)
            Neighbourhood = Swap.Neighbours
            name = Swap.ID
            costFromStart = Swap.Cost
            PreviousNode = Swap.Previous
        End Sub
        Sub Newcost(ByVal costUpdate)
            costFromStart = costUpdate
        End Sub
        Sub SetStartNode()
            name = "A"
```

```
            costFromStart = 0
            PreviousNode = "_"
        End Sub
        Sub SetDefaultNode()
            name = "."
            costFromStart = Integer.MaxValue
            Neighbourhood.Clear()
            PreviousNode = "."
        End Sub
        Sub ChangePrevious(ByVal neighbourino As Node)
            PreviousNode = neighbourino.ID
        End Sub


        Function Neighbours() As List(Of Char)
            Return Neighbourhood
        End Function
        Function ID() As Char
            Return name
        End Function
        Function Cost() As Integer
            Return costFromStart
        End Function

        Function Previous() As Char
            Return PreviousNode
        End Function

    End Class

    Sub MatrixMaker(ByVal nodetotal As Integer, ByRef adjacencymatrix(,)
As Integer)

        For i = 0 To nodetotal - 1
            For j = 0 To nodetotal - 1
                If i = j Then
                    adjacencymatrix(i, j) = Integer.MaxValue
                Else
                    If Int(Rnd() * 10) + 1 < 9 Then
                        If adjacencymatrix(i, j) = 0 Then
                            adjacencymatrix(i, j) = Int(Rnd() * 100) + 2
                            adjacencymatrix(j, i) = adjacencymatrix(i,
j)
                        End If
                    Else
                        adjacencymatrix(i, j) = Integer.MaxValue
                        adjacencymatrix(j, i) = adjacencymatrix(i, j)
                    End If
```

```
                End If
            Next
        Next

        adjacencymatrix(0, nodetotal - 1) = Integer.MaxValue
        adjacencymatrix(nodetotal - 1, 0) = Integer.MaxValue
        adjacencymatrix(0, 0) = 0

    End Sub


    Sub DijkstraTimeV2(ByRef unvisited As Dictionary(Of Char, Node),
ByVal visited As Dictionary(Of Char, Node), ByRef nodetotal As Integer,
ByRef adjacencymatrix(,) As Integer)
        Dim placeholderNode As New Node(".", adjacencymatrix, nodetotal)
        Dim currentNode As New Node(".", adjacencymatrix, nodetotal)
        Dim comparisonNode As New Node(".", adjacencymatrix, nodetotal)
        Dim counter As Integer = 0
        Dim tempchar As Char
        Dim listForVal As New List(Of Node)

        While unvisited.Count <> 0

            currentNode = New Node(".", adjacencymatrix, nodetotal)
            For Each key In unvisited.Keys
                If currentNode.Cost > unvisited(key).Cost Then
                    currentNode.NodeCopy(unvisited(key))
                End If
            Next
            listForVal.Add(currentNode)
            For Each neighbour In currentNode.Neighbours
                If unvisited.ContainsKey(neighbour) Then
                    If currentNode.Cost +
adjacencymatrix(Asc(currentNode.ID) - Asc("A"), Asc(neighbour) -
Asc("A")) < (unvisited(neighbour).Cost) Then
                        unvisited(neighbour).Newcost(currentNode.Cost +
adjacencymatrix(Asc(currentNode.ID) - Asc("A"), Asc(neighbour) -
Asc("A")))
                        unvisited(neighbour).ChangePrevious(currentNode)
                        If unvisited(neighbour).Cost <
comparisonNode.Cost Then

comparisonNode.NodeCopy(unvisited(neighbour))
                            tempchar = neighbour
                        End If
                    End If
                End If
            Next
```

```
            If Not visited.ContainsKey(listForVal(counter).ID) Then
                visited.Add(listForVal(counter).ID, listForVal(counter))
                unvisited.Remove(currentNode.ID)
            End If
            counter += 1

            If counter > 50 Then
                counter = "A"
            End If

        End While
        Console.WriteLine("")
        Dim path As New List(Of String)
        path.Add(visited(Chr(64 + nodetotal)).ID)
        placeholderNode.NodeCopy(visited(Chr(64 + nodetotal)))

        While placeholderNode.Previous <> "_"
            path.Add(placeholderNode.Previous)
            placeholderNode.NodeCopy(visited(placeholderNode.Previous))
            counter += 1
        End While

        For i = path.Count - 1 To 0 Step -1
            Console.WriteLine(path(i))
        Next

    End Sub


    Sub Main()

        Randomize()

        Dim unvisited As New Dictionary(Of Char, Node)
        Dim visited As New Dictionary(Of Char, Node)
        Dim counter As Integer
        Console.WriteLine("how many nodes do you want?")
        Dim nodeTotal As Integer = Console.ReadLine

        Dim adjacencyMatrix(nodeTotal, nodeTotal) As Integer
        MatrixMaker(nodeTotal, adjacencyMatrix)
        Dim currentNode As New Node("A", adjacencyMatrix, nodeTotal)
        For i = 1 To nodeTotal - 1
            unvisited.Add((Chr(Asc("A") + i)), New Node((Chr(Asc("A") +
i)), adjacencyMatrix, nodeTotal))
        Next
        unvisited.Add("A", currentNode)
```

```
            unvisited("A").SetStartNode()

            For i = 0 To nodeTotal - 1
                For j = 0 To nodeTotal - 1
                    If adjacencyMatrix(i, j) = Integer.MaxValue Then
                        Console.Write("N/A, ")
                    Else
                        Console.Write(adjacencyMatrix(i, j) & ", ")
                    End If
                Next
                Console.WriteLine()
            Next
            DijkstraTimeV2(unvisited, visited, nodeTotal, adjacencyMatrix)

            Console.WriteLine("The Shortest Distance from the first node to
the last node is " & visited(Chr(Asc("A") + (visited.Count - 1))).Cost &
" and the node journey is as follows")

            Console.ReadKey()
        End Sub

End Module
```

1.5.1.3 Reflection on Prototype

Having completed this prototype I have learned a few things about how both the dijkstra's algorithm and adjacency matrix generation could be effectively implemented into the overall solution.

The first thing I learned is that the matrix generation in my prototype handles the addition of nodes differently than I'd expected. In my prototype, a matrix that is bigger than another would not lead to locations being further apart (as expanding a map would), but would instead cover a similar area more densely. It would provide many more ways to arrive at each node, and this meant that the bigger the matrix, in general, the shorter the shortest path. This is an issue in respect to emulation of a real map. The way that I project I can solve this issue in the end solution is to make both the odds of 2 nodes being connected and the odds of the weight of this connection being small are both inversely proportional to the number of nodes in the matrix. This can be done in a few ways, but I project that the most effective could be to make the odds of a connection be calculated with an equation making the odds of nodes being connected being inversely proportional to some root of the number of nodes

I also learned that in order for the solution to run safely and reliably, I will need some measure to ensure that matrices generated will be robust. Whether or not this will be using error catching when a user tries to navigate to an unconnected node or by

regenerating a matrix until every node is somehow connected, or even editing existing nodes to ensure this is the case remains to be seen.

1.6 Objectives

My program will solve the issue of poor human resource management by finding the fastest route for multiple professionals to complete as many tasks as possible. This will be done using a combination of real-time algorithmic calculations
The program must know average travel times and assign calls to professionals based on whether or not they can get there within a set response time.
On top of this it must be aware of specialised training professionals have.

User Interaction

1. UI must allow for different roles, having different privileges
    a. Admin, who assigns, removes or reassigns jobs to be put into the schedules of professionals.
        i. Jobs will be made up of two variables: Type and Location.
            - The type will be a short description of the problem the client is having. It will also implicate the required qualifications the employee completing it must have
            - The location will be a location code, helping with navigation.
    b. Employees, who can see the schedules, navigate using the pathfinding algorithm, enter when an action has been completed, and enter when an action cannot be completed, which will notify the distributor so the appropriate action can be taken
        i. Employees, on completion of a job, will enter the start and finish times, and the whole job will be logged into the database, along with the professional's name/identifier.
            - This logged data, particularly the time the job took, will be used for tighter scheduling accuracy.
2. UI must be able to show AT ANY TIME the professional's path to their next position

Database Functionality

3. My program should be able to maintain its relations fully while taking in, storing, and displaying data surrounding
    a. Companies
    b. Teams
    c. Regions
    d. Users
    e. Clients
    f. Job Types
    g. Jobs
    h. Qualifications

Algorithm Efficiency

4. My program should be able to find the shortest route between two places.
5. My program should be able to chain multiple paths together the allow professionals to visit multiple places in as short a distance as possible
6. My program should have graphs to traverse with consistent “geography” (meaning clients and professionals are all based in the same place at the start of each day.

Algorithm Functionality

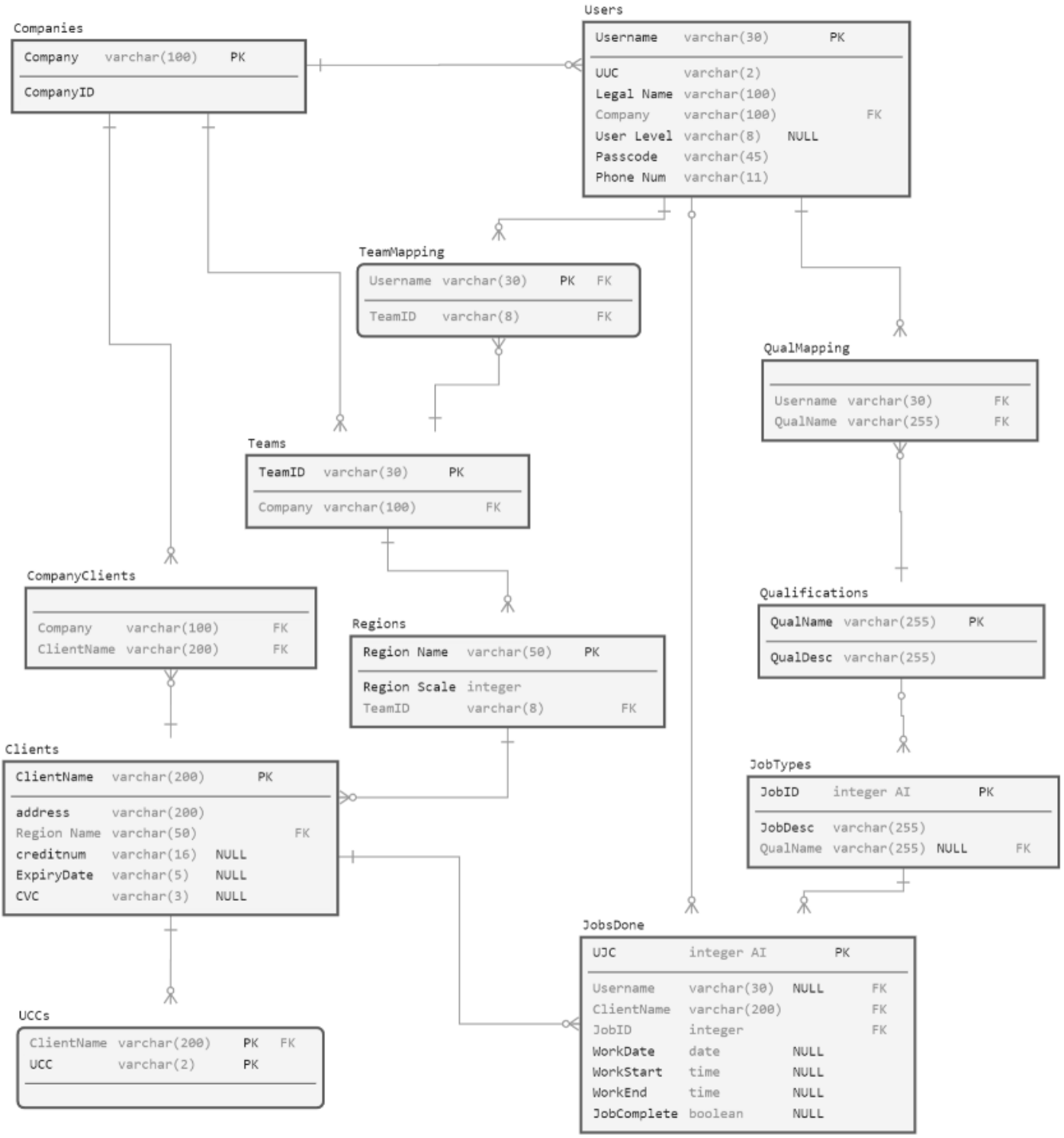
7. My program should be able to balance location of all employees to be optimal to minimise travel time and maximise productivity while not repeatedly missing clients
8. My program should be able to keep professionals from clashing at one client location
9. My program should be able to manage professionals that utilise different qualifications and remain optimal
10. My program should schedule no more than 8.5 hours of travel time in each employee schedule

1.7 Modelling

Much of my design will be based on containing consistent details about users, clients, regions, teams and jobs. This will require a strongly interlinked, normalised database. This database, in total will have to connect all the following details:

- Names of companies including a random, unique ID required to register users as employees under a company
- Details of a user, including username & password, name, user level & end-user-level details
- A generic type of job to be completed
- The qualifications required to complete the job
- Mapping which users have which qualifications
- Client details, including their name, address, which company they’re being served by, and which region they're in
- Teams including the users which are mapped to Regions as well as unique codes implying which user will originate from which point in the region (a “pseudo-home” for he user)
- The size and name of a region (so that it can be mapped to a binary file, which will contain its details, allowing it to remain consistent without burdening the database with each region’s data).
- Which jobs need to be done for each client
- Who will do each job for the client after it’s been registered as to be done
- When the user has started and finished each job

In my solution, this info will be contained in the following ER diagram



2 Design

2.1 Functional Components

The solution functions by and for many components. These are linked via an sql database in some cases, but also on a user level exists as self contained components. Some of these are intrinsically linked, some of them are not. They are as follows

2.1.1 Companies

Companies act as bases of operations for all other components. They require an admin to be instantiated, but apart from that they act more as categories allowing users/clients to only act within its jurisdiction. Companies have names and codes, the names are user defined, and the codes are unique and randomly generated. The code is the important aspect however, since it is required for a user to be instantiated.

Dependencies:

- 1 Admin (mutual)

Aspects

- Company name -> (String of length 100, user defined)
- Company ID -> (String of length 5, randomised, unique)

DDL

```
CREATE TABLE `Companies`(
`Company` VARCHAR(100) NOT NULL,
`CompanyID` VARCHAR(5) NOT NULL,
PRIMARY KEY (`Company`, `CompanyID`)
);
```

The above table contains the info about companies

2.1.2 Users

2.1.2.1 Admins

Admins are the users with the most independence and control. The majority of other components rely on admins to be created, but not to exist. Admins are created during the same process as companies, meaning the 2 are mutually dependent on one another. Their aspects are a unique user code (though this is currently unused) for admins, a username (which is unique to all other users)

Dependencies:

- 1 Company (mutual)

Aspects

- UUC (Unique user code) -> (String of length 2, user defined, unique)
- Username -> (String of length 30, user defined, unique)
- Passcode -> (String of length 45, user defined)
- User Level -> (String of length 8)
- Legal Name -> (String of length 100, user defined)
- Company Name (String of length 100)
- Phone Number - > (String of length 11, user defined)

2.1.2.2 Employees

Employees are the only other end user component.Their only write interaction with the database is marking jobs as complete, apart from that they simply read and interpret data from the database. As such their dependencies are low, requiring only a company for them to exist within to be created

Dependencies:

- 1 Company

Aspects

- UUC (Unique user code) -> (String of length 2, user defined, unique)
- Username -> (String of length 30, user defined, unique)
- Passcode -> (String of length 45, user defined)
- User Level -> (String of length 8)
- Legal Name -> (String of length 100, user defined)
- Company Name (String of length 100)
- Phone Number - > (String of length 11, user defined)

DDL

```
CREATE TABLE `Users` (
`UUC` VARCHAR(2) DEFAULT "A",
`Legal Name` VARCHAR(100) NOT NULL,
`Company` VARCHAR(100) NOT NULL,
`User Level` VARCHAR(8),
`Username` VARCHAR(30) NOT NULL,
`Passcode` VARCHAR(45) NOT NULL,
`Phone Num` VARCHAR(11) NOT NULL,
PRIMARY KEY (`Username`),
FOREIGN KEY (`Company`) REFERENCES `Companies`(`Company`)
);
```

The above table contains the info about all users

2.1.2.2.1 Qualifications

Qualifications act more as descriptors than components, but are still worth mentioning due to the important role they play in the efficiency and job registration systems. Qualifications are

linked to jobs so that only the appropriate employees can be assigned to specific job types. This is a vital argument for the efficiency algorithm, which in real terms would be the difference between serving customers effectively or wasting both employee and customer time.

Aspects

- Qualification Name -> (String of length 255)
- Qualification description -> (String of length 255)

DDL

```
CREATE TABLE `Qualifications` (
`QualName` VARCHAR(255) NOT NULL,
`QualDesc` VARCHAR(255) NOT NULL,
PRIMARY KEY (`QualName`)
);
```

The above table contains the info about which company each team is under

```
CREATE TABLE `QualMapping` (
`Username` VARCHAR(30) NOT NULL,
`QualName` VARCHAR(255) NOT NULL,
FOREIGN KEY (`Username`) REFERENCES `Users`(`Username`),
FOREIGN KEY (`QualName`) REFERENCES `Qualifications`(`QualName`)
);
```

The above table contains the info about which employees have each qualification, they can have any amount

2.1.3 Teams

Teams are collections of users that act as a unit of separate parts to serve a region (2.1.5) full of clients (2.1.6). They don't need a user to exist, technically, but they cannot function without users that have qualifications, so their relationship can be considered a dependency. They are required for the efficiency algorithm to function, and thus jobs to be completed, but don't need to perform these functions or any other to exist. They can be created or deleted at any time by an Admin (2.1.2)

Dependencies:

- 1 Company
- 1 User (with qualifications) [to function]
- 1 Admin {for creation only}

Aspects

- Team ID -> (String of length 8, user defined, unique)

DDL

```
CREATE TABLE `Teams`(
`Company` VARCHAR(100) NOT NULL,
`TeamID` VARCHAR(30) NOT NULL,
INDEX Teamorder(`TeamID`),
PRIMARY KEY (`TeamID`),
FOREIGN KEY (`Company`) REFERENCES `Companies`(`Company`)
);
```

The above table contains the info about which company each team is under

```
CREATE TABLE `TeamMapping`(
`TeamID` VARCHAR(8) NOT NULL,
`Username` VARCHAR(30) NOT NULL,
INDEX Teamorder(`TeamID`),
PRIMARY KEY (`Username`),
FOREIGN KEY (`Username`) REFERENCES `Users`(`Username`),
FOREIGN KEY (`TeamID`) REFERENCES `Teams`(`TeamID`)
);
```

The above table contains the info about which user is in which team

2.1.4 Region

Regions depend entirely on teams to exist, but have no other dependencies. They are composed of a name and a scale, both of which reference an adjacency matrix stored in a .region file in the program files, allowing the data from these files to be read. They also are linked to teams and clients once the matrix is loaded, as the user and the client both need positions within the matrix, but none of these relationships are dependencies for existence. Functionally, however, the region will require 1 client and 1 qualified employee.

Dependencies:

- 1 Company
- 1 Team
- 1 User (with qualifications) [to function]
- 1 Client [to function]
- 1 Admin {for creation only}

Aspects

- Team ID -> (String of length 8, user defined, unique)

DDL

```
CREATE TABLE `Regions`(
`Region Name` VARCHAR(50) NOT NULL,
`Region Scale` INTEGER NOT NULL,
`TeamID` VARCHAR(8) NOT NULL,
PRIMARY KEY (`Region Name`),
FOREIGN KEY (`TeamID`) REFERENCES `Teams`(`TeamID`)
);
```

The above table contains the info about regions for both identification and use in binary file interpretation

2.1.5 Clients

Clients are created by admins to be locations for employees to pathfind to. They can exist only when mapped to a region, but that is their only stipulation. They are composed of a UCC (Unique Client Code) dictating their position within that region, a Name, an Address, and any potential credit card info. Everything but the UCC is fundamentally user level, be it for pathfinding or job table display. The UCC system is just a unique 2 character code that can't be shared by any users or other clients within that region. They also exist under companies so that who is serving them within any given job is restricted to only users within that company.

Adding the credit card info is entirely user level and optional. Adding them is a process that instigates all of them sequentially, starting with the credit card number, then expiration date, then CVC. Despite being entirely composed of numerical values, these are still stored as strings since no calculations will include them.

Dependencies:

- 1 Company
- 1 Region
- 1 Admin {for creation only}

Aspects

- UCC (Unique Client Code) - > (String of length 2, user defined, unique)
- Client name -> (String of length 200, user defined)
- Client address -> (String of length 200, user defined)
- Credit card number (optional) -> (String of length 16, user defined)
- Credit card expiration date (optional) -> (String of length 5, user defined)
- CVC (optional) -> (String of length 3, user defined)

DDL

```
CREATE TABLE `Clients`(
`ClientName` VARCHAR(200),
`address` VARCHAR(200) NOT NULL,
`Region Name` VARCHAR(50) NOT NULL,
`creditnum` VARCHAR(16) DEFAULT NULL,
`ExpiryDate` VARCHAR(5) DEFAULT NULL,
`CVC` VARCHAR(3) DEFAULT NULL,
PRIMARY KEY (`ClientName`),
FOREIGN KEY (`Region Name`) REFERENCES Regions(`Region Name`)
);
```

The above table contains the info of each client about themselves and the region they're in

```
CREATE TABLE `CompanyClient`(
`Company` VARCHAR(100) NOT NULL,
`ClientName` VARCHAR(200) NOT NULL,
FOREIGN KEY (`Company`) REFERENCES `Companies`(`Company`),
FOREIGN KEY (`ClientName`) REFERENCES `Clients`(`ClientName`)
);
```

The above table contains the info about which company serves this particular client

```
CREATE TABLE `UCCs`(
`ClientName` VARCHAR(200),
`UCC` VARCHAR(2),
PRIMARY KEY (`UCC`, `ClientName`),
FOREIGN KEY (`ClientName`) REFERENCES `Clients`(`ClientName`)
);
```

The above table contains the info about the UCC (Unique Client Code) of each client, which describes its position in the region matrix

2.1.6 Jobs

2.1.6.1 Jobs (Unregistered)

Jobs (Unregistered) refers to the jobs in their theoretical state, existing in the database as descriptions of potential jobs that may need to be done. These only require at least one qualification to exist as they exist as a short description, a qualification requirement and an ID to signify them in other tables

Dependencies:

- Qualifications

Aspects

- Job ID -> (Auto increment integer)
- Job Description -> (String of length 255, user defined)

DDL

```
CREATE TABLE JobTypes(
`JobID` INTEGER AUTO_INCREMENT NOT NULL,
`JobDesc` VARCHAR(255) NOT NULL,
`QualName` VARCHAR(255),
PRIMARY KEY (`JobID`),
FOREIGN KEY (`QualName`) REFERENCES `Qualifications`(`QualName`)
);
```

The above table contains the info about types of jobs and the qualification they require

2.1.6.2 Jobs (Unassigned)

Jobs (unassigned) refers to the jobs once they've been registered to a client and a team. They act as an input into the efficiency algorithm, setting up the capability for jobs to be assigned to users after processing. An infinite number of these are able to be stacked up and due to the nature of the algorithm, only the jobs that are feasible to be completed will become assigned within a day cycle. Each of these jobs is assigned an automatically integrated UJC (unique job code) which allows each job to be referenced of a user level uniquely from one another if some recorded element is anomalous.

Dependencies:

- Job ID (necessitating 2.1.6.1)
- Client
- Team

Aspects

- UJC -> (Auto increment integer)
- Job completion -> (Boolean, set to false during this phase)

DDL

```
CREATE TABLE `JobsDone`(
`UJC` INTEGER AUTO_INCREMENT,
`Username` VARCHAR(30) DEFAULT NULL,
`ClientName` VARCHAR(200) NOT NULL,
`JobID` INTEGER NOT NULL,
`WorkDate` DATE DEFAULT NULL,
`WorkStart` TIME(0) DEFAULT NULL,
`WorkEnd` TIME(0) DEFAULT NULL,
`JobComplete` BOOLEAN DEFAULT FALSE,
PRIMARY KEY (`UJC`),
FOREIGN KEY (`Username`) REFERENCES `Users`(`Username`),
FOREIGN KEY (`ClientName`) REFERENCES `Clients`(`ClientName`),
FOREIGN KEY (`JobID`) REFERENCES `JobTypes`(`JobID`)
);
```

The above table contains all information about jobs that have actively been completed. It is the end point of the processing of all data

2.1.6.3 Jobs (Assigned)

This is the final state of an uncompleted job once it has gone through the efficiency algorithm. At this point, an employee has been assigned to the job, and as such the job and client are visible to the user from Menu E (2.3.3). This step can be used in its entirety to tell an employee the route to their next location (see 2.4.4.3).

Dependencies:

- Jobs (Unregistered)
- Client
- Team
- Employee

Aspects

- UJC -> (Auto increment integer)
- Job completion -> (Boolean, set to false during this phase)

2.1.6.4 Jobs (Completed)

This is the final state of a job once it has been marked as complete by an employee. This is completed via a subroutine, but requires a start time to be written by an employee. The current system date and time are entered as date and time fields so that an approximate time taken to do the job is calculated, and so that the system time can be compared to this timestamp in the future to tell whether an employee needs to pathfind from a client location (indicated by their UCC), or from their home (indicated by their UUC) respectively.

Dependencies:

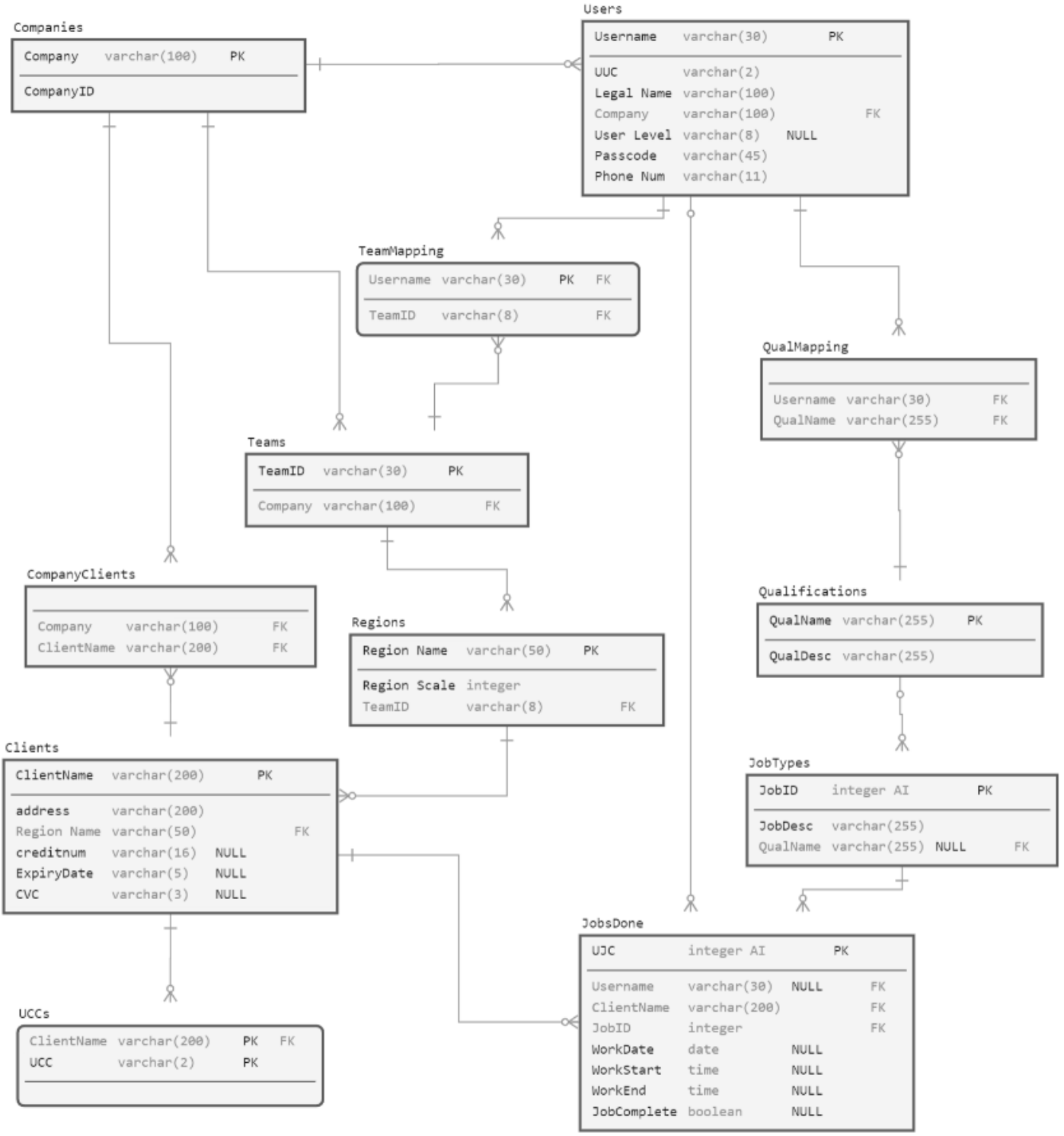
- Jobs (assigned)

Aspects

- UJC -> (Auto increment integer)
- Job completion -> (Boolean, set to true during this phase)
- Job Start time
- Job End time
- Date of completion

2.2 Database Entity Relationships

A lot of my solution is dependent on my SQL database. It is normalised to reduce redundancies, and as such, it's split into 11 tables. The main structure of this database allows primarily for data to exist on Team level, Region level, User level and Job level, all of which being given unique codes to make them distinct, and all of which interact either directly (such as which user is in which team) or by creating rules (such as jobs can only be completed in a region by a user in the team that covers the region) to keep data consistent The entity-relationship diagram can be seen below.



2.3 User System

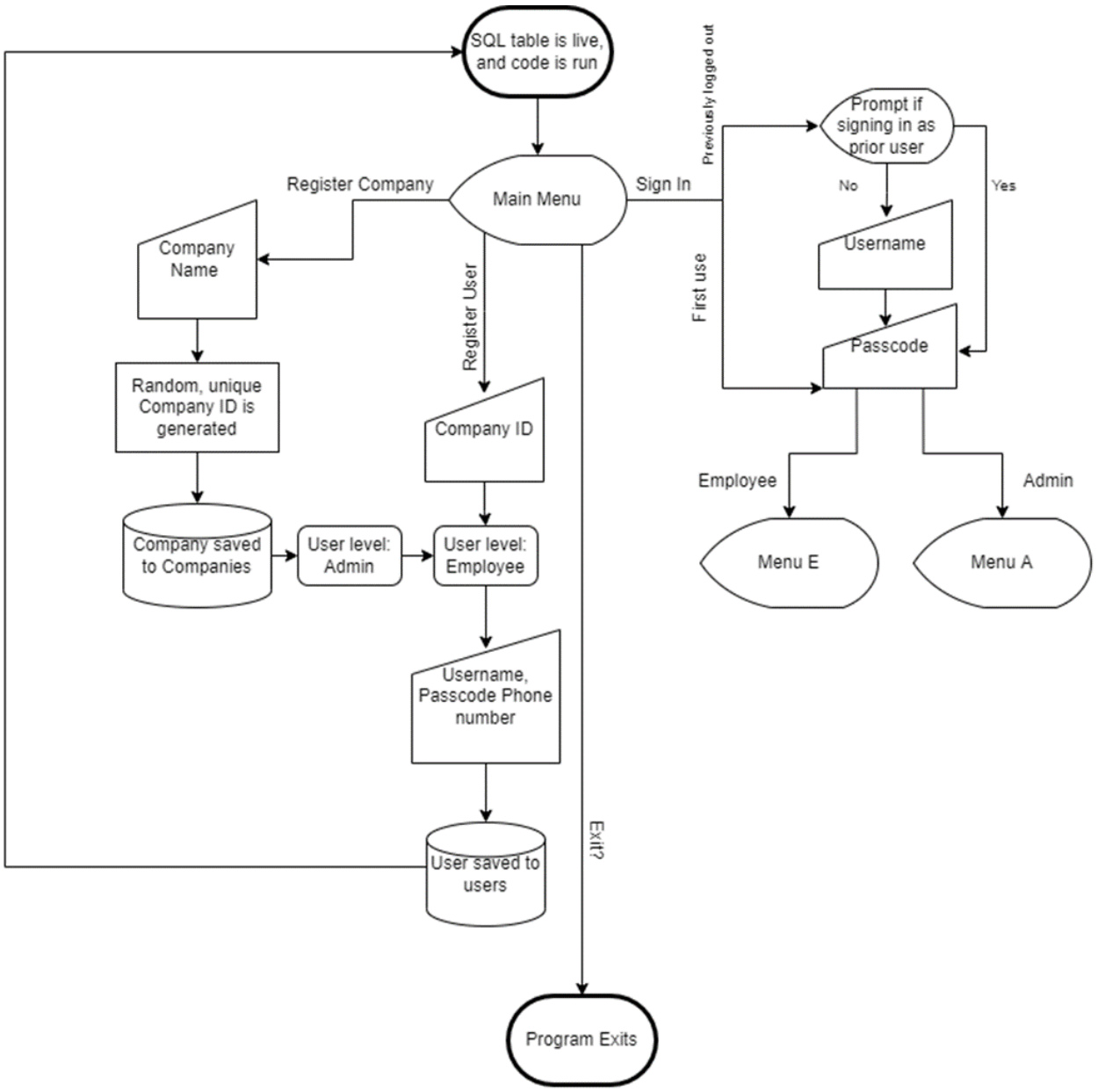
In order to make the program work on multiple levels, I've implemented a table in my sql database to connect each user to a user level which dictates what they can do. The hierarchy is such that the admin can set jobs, calculate who does the jobs, and the employee can view them and mark them as done. The admin is also able to change variables such as the employees qualifications, adding a client or region, adding or editing a team. The admin can also view the dijkstra's algorithm acting on an existing or randomly generated map matrix.
I've created this user hierarchy by using a structure, since that allows all the details to be taken from the database and held in one place, and it allows for a signing in protocol to act as a gateway for security and identification purposes
These user levels are the primary key to navigation through the menu system I've implemented. The structure of these menus is detailed in the following section, as well as a brief summary of the implications of each path, the majority of which simply add to, display, or update the SQL tables, but some of which are more complex, and will give references to other parts of the design description instead.

2.4 Menu Structure

The ethos of the menu structure I've decided to use is to keep matters surrounding any certain component localised, and to allow for the most branches in each menu to cut down on the number of menus, for clutter purposes. This is to say that if a user wishes to view or change teams, the team view will be given before the user advances to the next step, and the user is prompted to return to the prior menu or continue to the next. This is a linearly designed way to make viewing and altering teams a part of the same branch, which both keeps team-related matters localised and allows for multiple branching paths if the user wishes to continue.

It's examples such as this that underpin my menu structure, and allow users to find their way to entering whatever information is relevant, which the UI will prompt them to do one by one so as to not overwhelm the system or the user.

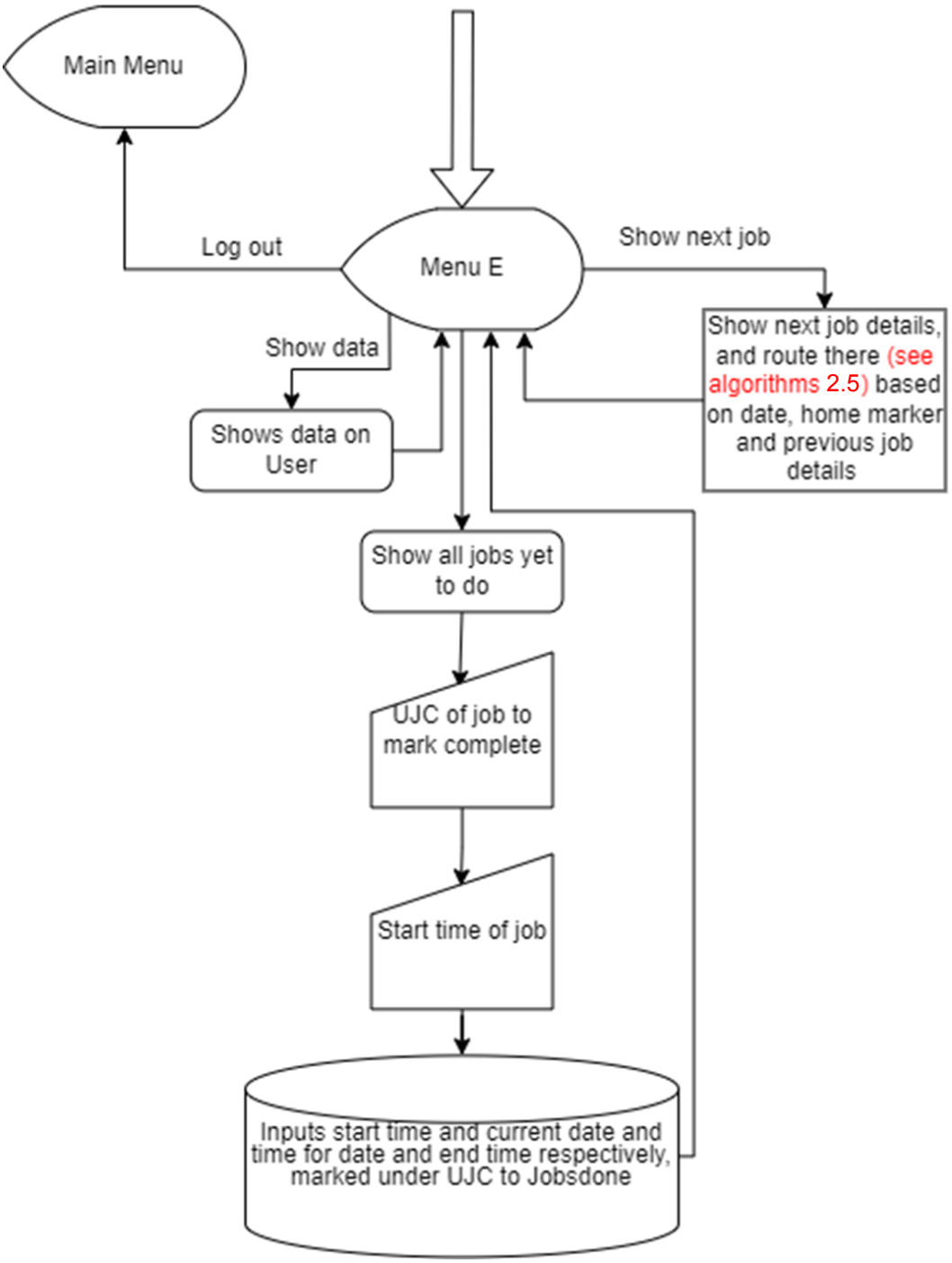
The specific design of the menu as a whole is split into 6 sub-menus creating one structured whole. The six components can be represented as flowcharts, which can be seen on the following pages, and any given point in the menu can be reached from any other.



2.4.1 Main Menu

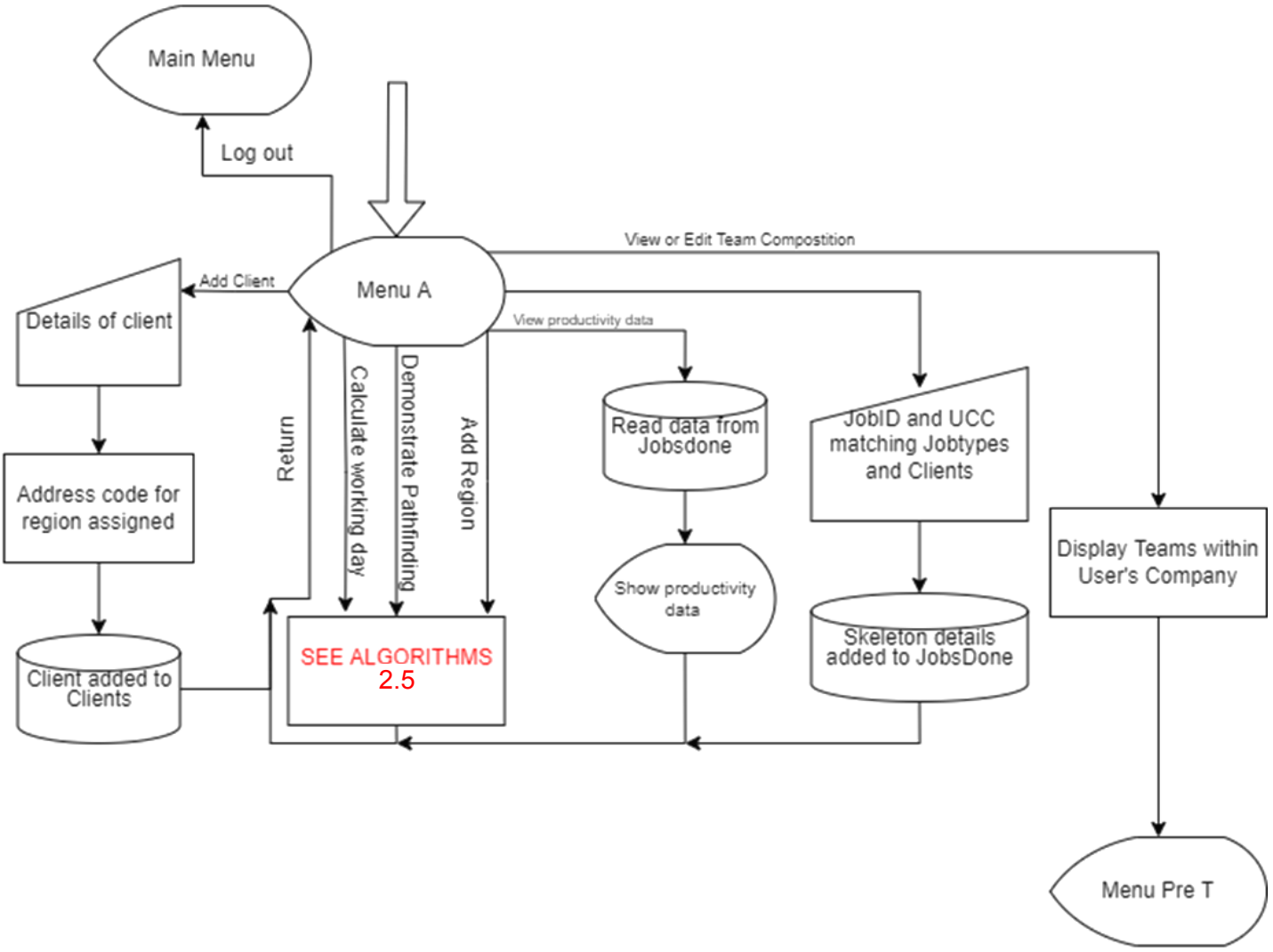
This menu acts as a way to create a user, create a company and a user to administer said company, or for an existing user to sign in. This signing-in process allows both for authentication of an existing user, and a way to indicate the user's access capabilities, namely which company they can access and whether or not they have admin or employee privileges. It also acts as the final gateway of sorts to exiting the program (bar of course force stopping it in the console window). This means that a user will be signed out before they quit.

Another consideration, purely for user convenience sake, is that the previous user signed in within an instance of the program is stored, meaning they need only enter their password to sign back in.



2.4.2 Menu E

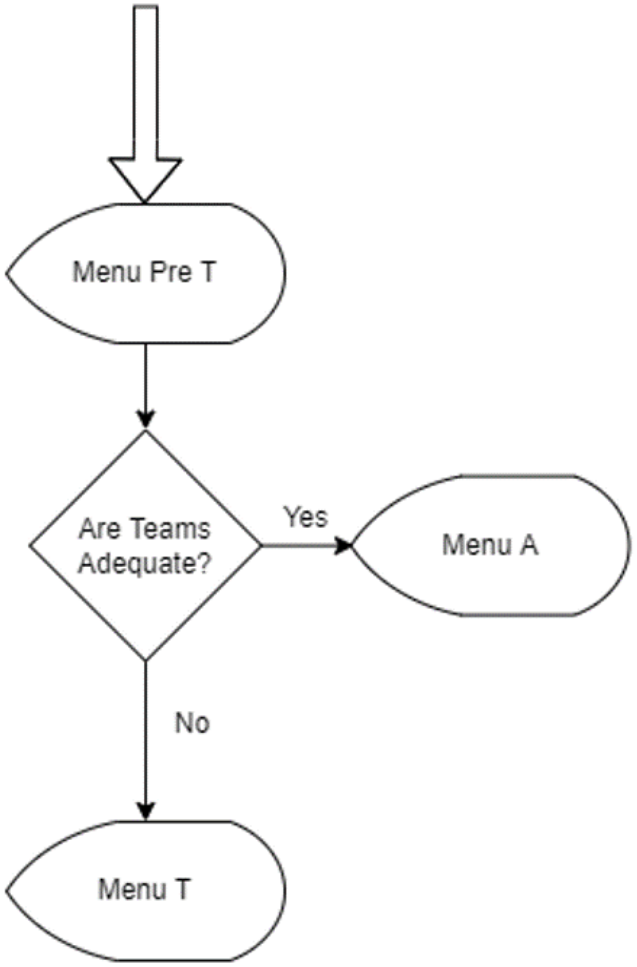
This menu acts as a gateway for employee level users. It enables them to access their personal data, show the order of the jobs they need to do, mark jobs as complete and after they're complete, sign out. Its main priority, however, is the efficiency algorithm (see 2.5.2.2).



2.4.3 Menu A

This is the hub of activity for Admin level users. This acts as a way for most forms of general database modification. It allows for clients to be added, pathfinding to be demonstrated (see 2.3.2.1), productivity data to be shown and contains an entrance to team data, which is handled in a different menu. The purpose of pathfinding demonstration is so that admins can trace journeys and check region info. This is important for validation that a generated map matrix is reproducible and consistently diverse, which of course is also important for showing that the pathfinding, and greater efficiency algorithms both work as intended.

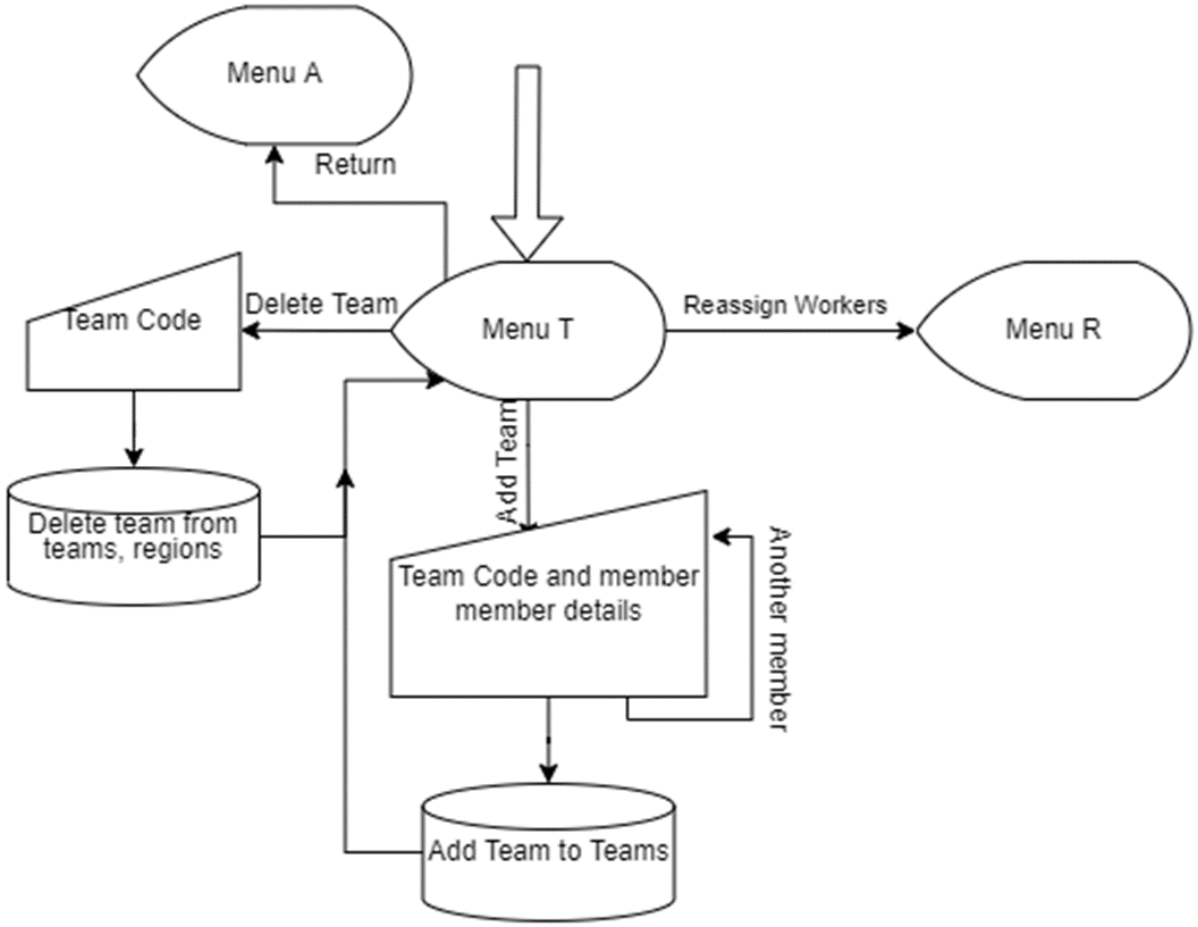
Productivity data, in this context, details how long employees take to do tasks, what jobs are registered, assigned and completed, and who has done what and when (both dates and times are considered). This, in real terms, would allow for admins to contact underperforming employees, acknowledge overachieving employees, and notice any other abnormalities that may indicate errors in the efficiency algorithm

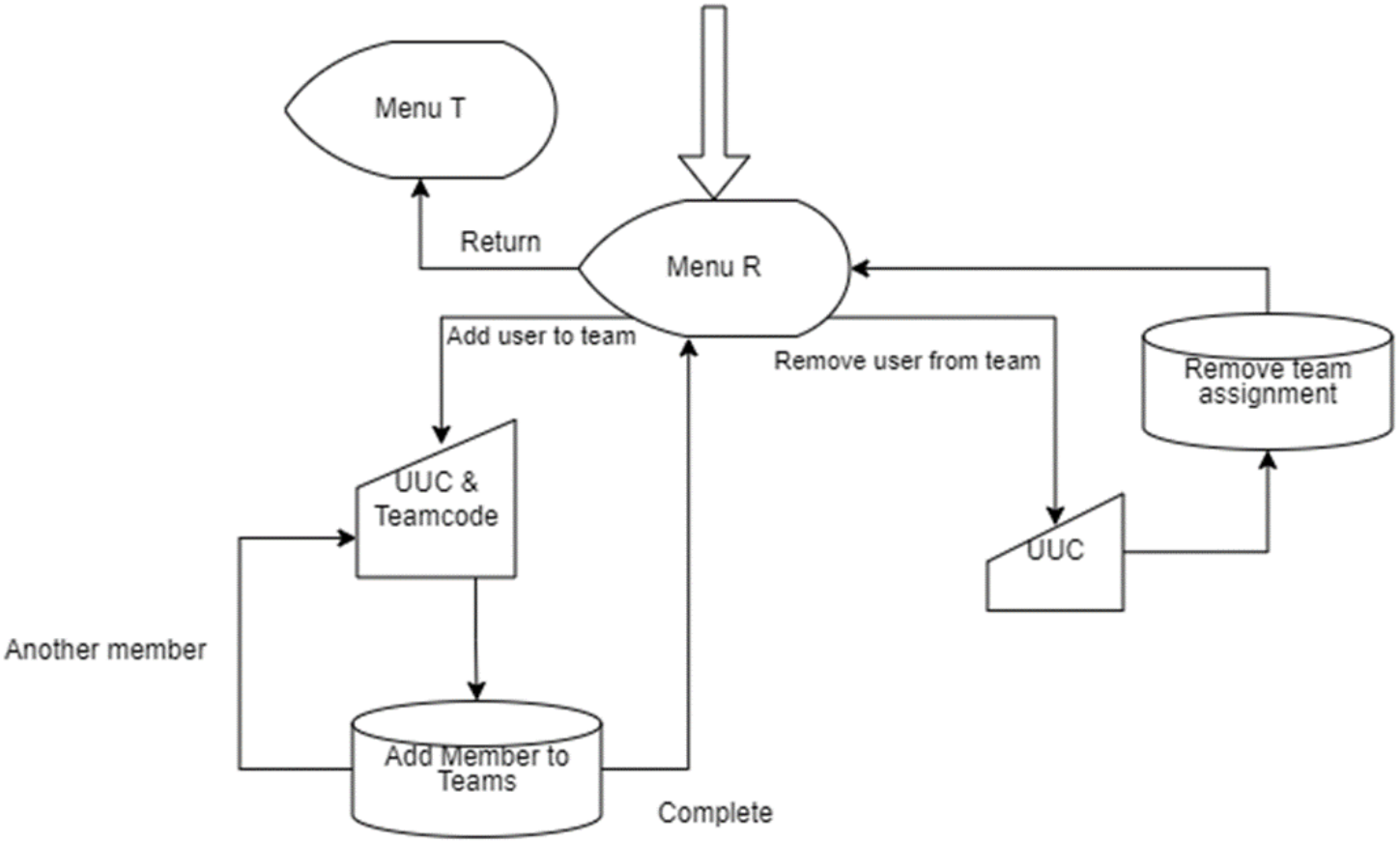


2.4.4 Menu Pre T and Menu T

These menus allow for team modification. Menu pre T exists simply to display which users are in which teams, acting as a go-between between menu A and menu T for users that simply wish to see the team info but don't want to edit anything. This acts simply to streamline the UI somewhat.

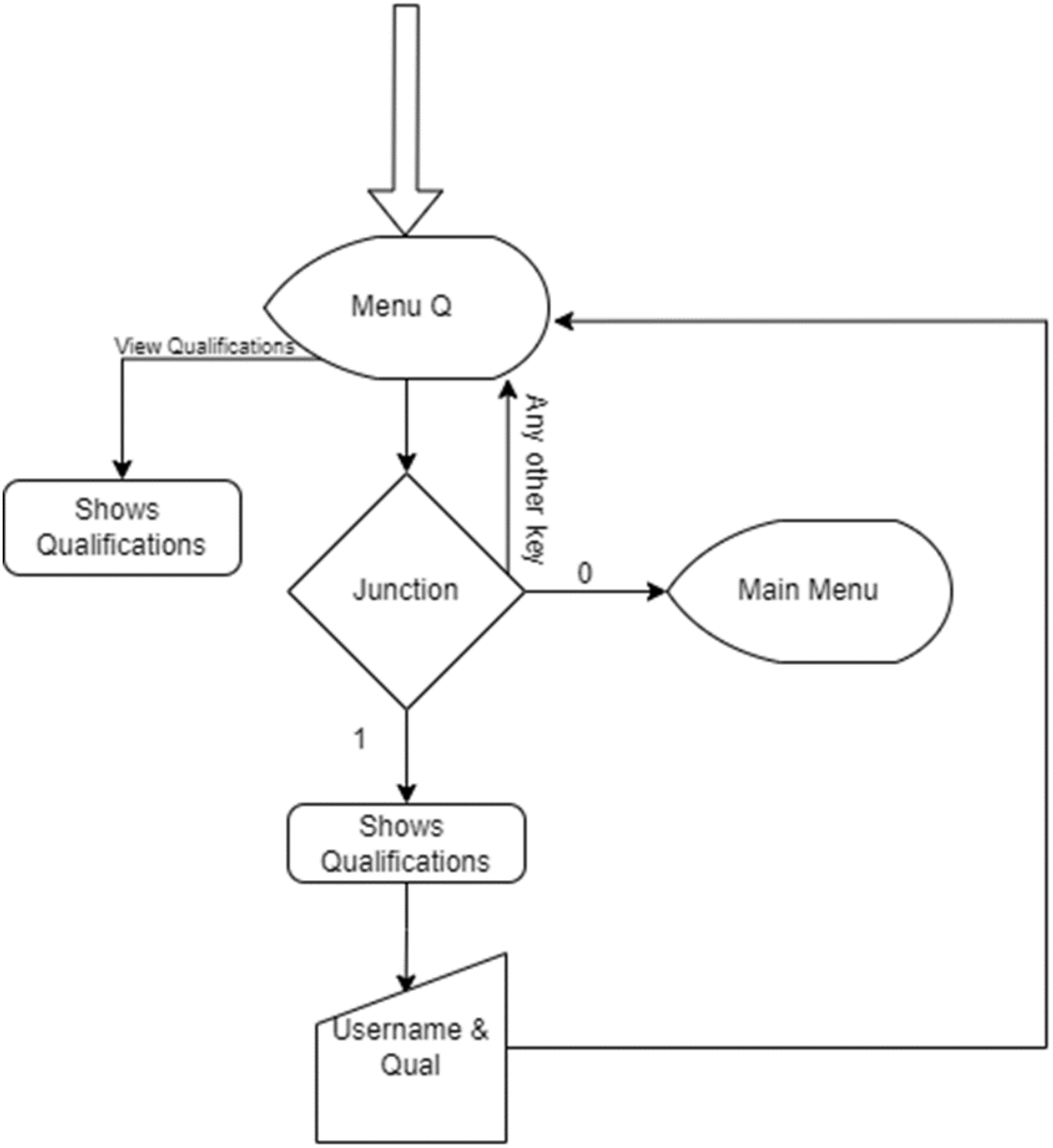
The team interface is split into Menu T and Menu R. Menu T deals with teams at a more abstract, collective level, adding or removing entire teams as one unit using SQL commands to remove all instances of said team, or to add a team user by user. A team can be empty, but a team cannot be created empty, necessitating at least one member to be created. Despite menu Pre T leading to menu T, menu T leads back to Menu T leads back to Menu A to keep the visuals of the teams separate from any of the rest of the code, for UI cleanliness reasons.





2.4.5 Menu R

Menu R allows for user level modification of teams on a user level, wherein users can freely be added or removed from teams. Teams and users are defined by team code and UUC respectively, streamlining the process as much as possible. You can also return to menu T when the job is done



2.4.6 Menu Q

Menu Q is the menu wherein qualifications can be viewed or added. The qualifications involved cannot be removed, as a large quantity of qualifications can exist in the database (though currently they must be added directly to the database, see 4.3.2 for more on this). This means that if qualifications are temporary (need refresher courses, etc.), another qualification can be made representing the dates that qualification covers.

2.5 Algorithms

2.5.1 Inputs

The algorithm I've used is Dijkstra's shortest path algorithm. I've used this because it works ideally in the situation that I encounter, wherein one path is checked at a time, and all nodes can be checked within a relatively short span of time. The implementation of the algorithm I've created allows for use with or without interacting with the UI, multiple searches between nodes and multiple data types to be returned based on the "mode" the subroutine is set to (one being a user friendly list of the distance and path of the shortest distance between two or more nodes, and one simply being the output weight of the shortest distance between two nodes.)

2.5.1.1 Matrices (Regions)

The structure I've used as an input representing the pseudo-map graph is an adjacency matrix. This is because it's simpler to randomly generate, and if the dimensions are known (which they are, as they are committed as an integer value into the Regions table of the database), can be easily committed to and read from binary files, which is the method I ended up using for meaningfully reproducible results. The files are stored in the standard subdirectory in the visual studio file, and are labelled as .region files.

The matrix also undergoes a subroutine that runs a modified version of Dijkstra's algorithm, which checks that every single node can connect via some path to every single other node before being approved for use as a region, or in pathfinding demonstration. If it isn't approved, then it's scrapped, and a new one is generated. The matrices also undergo processes to simulate an expansion of the pseudo-map along with the set scale of the matrix, with the odds of nodes being connected calculated using the following probability equation

$$1\ in\ \frac{1}{\sqrt[x]{n}} \times 100\ (\ 0\ d.p.\ )$$

Where x is some constant (likely 2 or 3, whichever causes the least issues generating a robust matrix) and n is the number of nodes. For input validation, and to make this equation consistent, users will be capped between 3 and 500.

The ranges of the weights between two nodes are calculated using the same calculation, but with the randomly generated number producing a result between 2 numbers when a certain threshold is passed. The higher the threshold, the higher the randomly generated number.

All of this in tandem results in a larger map generating larger shortest paths on average, mimicking a map increasing in size.

2.5.1.2 Nodes & lists

I used a Class to hold the details of each node (such as their current distance from the start and their ID), which enabled me to easily manipulate them by renewing them and putting them into the Lists to mark them as visited and unvisited within a Dijkstra algorithm cycle. Giving them IDs also helped with the inputs as, since I know the IDs will follow a naming convention (A-Z, AA-AZ, BA-BZ etc.), I can accurately create a list containing all nodes to navigate to through multiple uses of the dijkstra algorithm. This allows for tracing of an employee's path entirely within the subroutine just by decreasing the list. The use of a list here is ideal since it can vary in size based on user inputs and a list can change in size dynamically if required.

Because the classes used are not connected to anything via composition, aggregation or inheritance, this cannot be considered object oriented programming as a whole, just very selective use of its principals in ways that structures struggle to achieve.

2.5.2 Implementation

2.5.2.1 Demonstration

Since certain liberties have had to have been taken leaving the pseudo-map dissimilar to a real map in presentation (nodes not being named after locations, map being randomly generated, distances being random etc.), the dijkstra's algorithm can be freely used by the user to model potential paths without necessitating active functionality. It allows the user to create a matrix of any number of nodes between 3 and 500, or to use an existing one, and to create however many journeys it wants using the list described in 2.3.1.2. This is useful to check that the efficiency algorithm is working appropriately, and to experiment with the logistics of potential clients without assigning them yet. It's also useful to have a model of how the size of the matrix influences its contents, as discussed in 2.3.1.1.

2.5.2.2 Efficiency

The algorithm for the efficiency of job assignment (referred to elsewhere as the "efficiency algorithm") is the core function of the program. In effect, it ensures that the first available qualified person to a client will take that job, maximising efficiency in effect. The clients are served on a "first come first served" basis, so that further away customers from employees homes won't be shirked in favour of closer ones.

On a more in-depth level, structures are used for the employees and jobs. The structure for the employee details the employees home nodes, their qualifications, and their

current position. The job structure details the required qualification and the location node. All of this data is accessed from the database. The algorithm will add all the employees to a list, with their current nodes set to their home nodes, and all the jobs that need to be done to another list. A loop that cycles until either all jobs are complete, or there are no more available employees is instantiated, and within so, each employee that has the required qualification to complete the job is ran through the dijkstra algorithm once, and the distance between the employee's current position and the client's position is calculated. This time is added to the amount of time the employee is projected to have already worked, and then the employee who turns out to have the lowest output is assigned to that job. After this, each employee has the return journey modelled, and if the time meets or slightly exceeds 8 hours, the employee is sent home and disregarded from the future cycles.

All in all this means that the code maximises efficiency within the boundary that each customer must be served in the order that the task was assigned, meaning that each customer is met as efficiently as possible.

2.5.4.3 Next Location Pathfinding

The third and final implementation of dijkstra's algorithm was in assisting employees in finding their next location. The algorithm first takes in the date of the most recently completed job, and compares it with the current system date. If they are different, it can be assumed that a new work day is being calculated, and the origin is the employees home node (UUC). If they are the same, then the client node (UCC) is used as the origin. Then the next job that is assigned to them is examined for its client node, and the pathfinding takes these codes, converts them into integers that indicate their positions on the region matrix, and pathfinders from the origin to the destination. This is a fairly run-of-the-mill implementation of Dijkstra's algorithm, but worthy of mentioning separately from the others nonetheless.

3 Testing

Testing Video:
https://youtu.be/EQ_lqfbrLjc

Test Number	Objective	Test description	Time-stamp	Pass or Fail
1	3a.	Test that a company can be registered	0:43	Pass
2	3d, 1a	Test that an admin is assigned to each company on register	1:53	Pass
3	3a	Show that the companyID is randomly generated	0:43	Pass
4	3d, 3b	Test that a user can be registered under a company	2:37	Pass
5	General	Test that Menu won't accept invalid inputs for key inputs	17:30	Pass
6	General	Test that Menu won't accept invalid inputs for string inputs	1:14	Pass
7	General	Test that logging in works	3:50	Pass
8	1	Test that logging in as users of different levels effects UI	3:50, 17:43	Pass
9	General	Test that users can be changed during runtime	17:43	Pass
10	General	Test that if a user's credentials are invalid, the user will not be signed in	18:29	Pass
11	3f, 3g	Test that Job Types can be seen	10:45	Pass
12	3c	Test that Regions can be added, and put on a binary file	6:25 6:37	Pass
13	4	Test that Implementation of dijkstra's algorithm produces accurate results in a path with from one node to another	12:36	Pass
14	4, 5	Test that Implementation of dijkstra's algorithm produces accurate results in a path visiting multiple nodes	12:36	Pass

15	3c, 6	Test that saved regions can be read and produce reproducible data	13:34, 14:15	Pass
16	3b	Test that users can be placed in teams	4:31	Pass
17	3g	Test that productivity data can be shown	20:24	Pass
18	3e	Test that Clients can be added	7:20	Pass
19	3b	Test that teams can be added	4:31	Pass
20	3h	Test that Qualifications can be viewed and assigned	9:17, 9:57	Pass
21	3b	Test users can be removed from teams	5:02	Pass
22	3g	Test that users can be assigned jobs	10:54	Pass
23	7, 8, 9,	Test that users assigned to jobs are the most efficient at that time	16:22	Pass
24	3g, 1bi	Test that jobs can be marked as complete by the user	20:09	Pass
25	3g	Test that the database of jobs can be viewed live at any time	12:07	Pass
26	2	Test that users can see their next journey at any time and that it's accurate to their place on the rota	19:41	Pass
27	10	Test that employees only have to work 8.5 hours per day	N/A	FAIL
28	General	Test that sql injections via ‘, \, ‘ or # do not pass	1:44	Pass

All tests were successful except 27. In none of my attempts could reproducible results be observed confirming correct implementation of objective 10, due to time constraints.

4 Evaluation

4.1 Reflection

All in all, I am fairly satisfied with the way that my program meets its objectives when the variables are ideal. The algorithms are all fully functional, bar a minor hitch in keeping within a set time, and the database integration works fundamentally perfectly, even in its heavily normalised state. I think some robustness and options to edit the table more finely could be added given more time, but as for each aspect of the solution on a micro and macro scale work perfectly as intended, and with excellent efficiency, so I would classify this outcome as a success. Both potential end users' needs are met and their interactions function entirely as intended.

The list of objectives I've met can be seen in the testing section above. All objectives were met fully successfully, some such as 15, repeatedly, except for 27, which unfortunately didn't behave as I'd expected and as such couldn't produce the reproducible results required for me to day in good faith that I'd passed

Objective	Reflection
1a	This admin user type has been implemented nigh untouched from my original plan, and as such I think this objective has been met, if not exceeded
1ai	This objective has been met, and even exceeded, using client details as a way to describe location, minimising redundant data
1b	This employee user type is slightly different from what I expected to need, but I think that that cut for time (namely not allowing for notification of inability to complete jobs) was harmless, since there are user-level interactions in the real world that can solve this, as no authentication beyond initial sign-in is needed to mark jobs as complete.
1bi	This step was met and exceeded, but slightly differently than I'd first thought. At the beginning of the project I'd assumed that the user would be entering the start time at the start and the end time at the end along with the client details. As it happened in practice, however, only the start time was needed, the rest being handled by the assigned phase of a job and SQL timestamping
2	This objective has been met as planned
3	These objectives have been met perfectly as planned unless otherwise stated
3e	This, while being met, was fairly optimally split throughout my database, with 2 tables built to be normalised, which worked fine, but doesn't match uniformly with the other major components
3h	This, while being met, is met suboptimally in the sense that the admins cannot currently create new qualifications. Given more time, they would be able to.

4	This objective has been met as planned
5	This objective has been met as planned
6	This objective has been met well. It allows for different region sizes, stores the regions in a way that is not only consistent, but can technically be shared, however it still isn't perfect because the region size is limited, the file location is never explicitly explained
7	This objective hasn't quite been met as I'd hoped. Originally I wanted for a priority system to be used based on claimed priority by admins, time the client has waited, and I wanted the program to account for the time a job took to complete, by assigning each job type with the average time difference between the job being started and finished. This unfortunately was not completed due to insufficient MySQL knowledge and time constraints. This I believe is the project's biggest growth area.
8	This objective has been met as planned
9	This objective has been met as planned, as explained above
10	This objective has not been met, due to time constraints.

4.2 End-User Feedback

"The program's features were successful, and what the program attempts to do it does well. From both the admin and employee perspectives the UI is mostly easy to follow and shows the info it shows in an understandable manner. It definitely leaves some aspects out that should be included, the live database was viewable for me during beta testing, but some information could not be added or viewed, such as qualifications and client data respectively, which would require the user to note down these details, which negates the point of the database. However, the database does maintain and store all the existing data effectively. Improvements beyond the UI could be accounting for the time the job takes, which currently the main algorithm does not."

4.3 Improvements

4.3.1 Database changes

If I were to recreate this database again from the ground up, I'd likely execute it in a similar way in regards to the relationship between components (2.1) in a similar way in terms of algorithm functionality, however, I'd like to implement a column in the jobID table that takes the average time it's taken to do a job of that type from the jobsDone table to take into account when calculating the efficiency algorithm, as I think for utility purposes it would be pertinent to account for. Normalisation is also lacking in some areas. At the beginning of the task I intended to have the database in 6NF, but that quickly become a more significant task than I'd expected, and the database lacked a few aspects that were a necessity for the

algorithms to function in the way I wanted them to, but would require adding one or more new tables. This was particularly the case with the UXCs (unique X codes) Where in the case of clients a new table was added, but for users and jobs it was not, despite the username implying it by needing to be unique. The database is still separated in a near-normalised state, and is fully functional with very few redundancies, but is far from perfect in its design due to time constraints.

4.3.2 UI changes

As evident in the feedback, the UI could use some work, as it does not effectively show all potentially relevant info. This, while a design oversight, would only be a matter of time to implement, not necessitating the changing of any existing code to create. As well as this, however, in some processes it is impossible to navigate backwards through the menus, and in a perfect program, a user wouldn't have to close it to cancel an activity. I also would have liked to portray the regions in a more intuitive way, as a matrix is difficult to portray and perceive, and as a user, in practice, oftentimes you'd just have to trust the system, which isn't ideal. Both the previous two issues could be solved by use of a GUI, instead of using a CLI for convenience, and as such, that's likely the approach I'd take.

5 Technical Solution

Contents page for code

5 Technical Solution **42**
Contents page for code 42
Code 44
5.1 Classes 44
5.1.1 User Class 44
5.1.2 Node Class 48
5.2 SQL connection setup 50
5.3 SQL injection prevention using RegEx 51
5.4 Matrix Subroutines 51
5.4.1 Matrix Creation 51
5.4.2 Matrix Validation 53
5.5 Dijkstra's Algorithm 55
5.5.1 ID-Integer 2-way conversion for dijkstra 55
5.5.2 Dijkstra demonstration initiation 57
5.5.3 Dijkstra algorithm in practice 60
5.6 Company Creation 63
5.7 User Management 64
5.7.1 Creation 64
5.7.2 Credential Checker 66
5.8 Client Creation 67
5.9 Team management 68
5.9.1 Creation 68
5.9.2 Viewing and editing 70
5.10 Region Creation 75
5.11 Job Management 77
5.11.1 Creation 77
5.11.2 Marking Completion 79
5.11.3 Viewing 81
5.11.3.1 Employee level 81
5.11.3.2 Admin Level (Productivity Check) 83
5.12 Qualification management (Viewing and Assigning) 87
5.13 Efficiency Algorithm 89
5.14 Top Level Menus (Main, A & E) 92
5.15 SQL Database Creation Statements 96

Code

5.1 Classes

5.1.1 User Class

```
Public Class User
    Private entry As Boolean
    Private username, passcode, legalName, company, userLevel,
phoneNumber, Team As String
    Private conn As New System.Data.Odbc.OdbcConnection("DRIVER={MySQL
ODBC 5.3 ANSI
Driver};SERVER=localhost;PORT=3306;DATABASE=ProjDb;USER=root;PASSWORD=root
;OPTION=3;")

    Sub SignIn()

        conn.Open()
        username = ""
        Dim usernameInput, password As String
        Dim subj As ConsoleKey
        Dim reader, reader2 As Odbc.OdbcDataReader
        Dim checkForUser As New Odbc.OdbcCommand("", conn)
        Dim assignTeam As New Odbc.OdbcCommand("", conn)
        Do
            Console.Write("Sign in to initiate use
username: ")
            usernameInput = SQLmaintenance(Console.ReadLine(), 30)
            Console.Write("password: ")
            password = ""
            Console.ForegroundColor = Console.BackgroundColor
            subj = Console.ReadKey.Key
            Console.ForegroundColor = ConsoleColor.White
            While subj <> ConsoleKey.Enter
                If subj = ConsoleKey.Backspace And password.Length > 0
```

```
Then
                    password = password.Remove(password.Length - 1)
                    Console.Clear()
                    Console.Write("Sign in to initiate use
username: " & usernameInput & "
password: ")
                    For i = 0 To password.Length - 1
                        Console.Write(".")
                    Next
                    Console.SetCursorPosition(password.Length + 9, 2)
                ElseIf subj > 62 And subj < 91 Or subj > 47 And subj < 58
Then
                    If subj > 62 And subj < 91 Then
                        password &= subj.ToString
                    ElseIf subj > 47 And subj < 58 Then
                        password &= Mid(subj.ToString, 2, 1)
                    End If
                    Console.Clear()
                    Console.Write("Sign in to initiate use
username: " & usernameInput & "
password: ")
                    For i = 0 To password.Length - 1
                        Console.Write(".")
                    Next
                    Console.SetCursorPosition(password.Length + 9, 2)
                End If
                Console.ForegroundColor = Console.BackgroundColor
                subj = Console.ReadKey.Key
                Console.ForegroundColor = ConsoleColor.White
            End While
            subj = ConsoleKey.A
            Console.SetCursorPosition(0, 3)
            checkForUser.CommandText = ("SELECT * FROM Users WHERE
Username = '" & usernameInput & "' AND Passcode = '" & LCase(password) &
"';")
            reader = checkForUser.ExecuteReader
            If reader.Read() Then
```

```
                legalName = reader("Legal name")
                company = reader("Company")
                phoneNumber = reader("Phone Num")
                userLevel = reader("User Level")
                username = usernameInput
                entry = True
                assignTeam.CommandText = ("SELECT TeamID FROM TeamMapping
WHERE Username = '" & usernameInput & "';")
                reader2 = assignTeam.ExecuteReader
                Team = reader2.Read
                Console.WriteLine("Welcome " & legalName & ". Press any
key to return to the main menu.")
                Console.ReadKey()
                Console.Clear()
            Else
                Console.WriteLine("No user found! Check credentials and
try again.
Press any key to restart
Press Backspace to return")
                reader.Close()
                subj = Console.ReadKey.Key
                Console.Clear()
            End If
        Loop Until subj = ConsoleKey.Backspace Or entry
        conn.Close()
    End Sub

    Function Greet() As String
        Return legalName
    End Function
    Function SignedIn() As Boolean
        Return entry
    End Function

    Function GiveCompany()
        Return company
    End Function
```

```
Function PowerLevel() As Integer
    If userLevel = "Admin" Then
        Return 3
    ElseIf userLevel = "Manager" Then
        Return 2
    ElseIf userLevel = "Employee" Then
        Return 1
    Else
        Return 0
    End If
End Function
Function returnUsername()
    Return username
End Function
Function checkPassword()
    Dim password As String
    Dim subj As ConsoleKey
    Console.Write("Enter password: ")
    password = ""
    Console.ForegroundColor = Console.BackgroundColor
    subj = Console.ReadKey.Key
    Console.ForegroundColor = ConsoleColor.White
    While subj <> ConsoleKey.Enter

        If subj = ConsoleKey.Backspace And password.Length > 0 Then
            password = password.Remove(password.Length - 1)
            Console.Clear()
            Console.Write("Enter password: ")
            For i = 0 To password.Length - 1
                Console.Write(".")
            Next
            Console.SetCursorPosition(password.Length + 9, 2)
        ElseIf subj > 62 And subj < 91 Then
            password = password & subj.ToString
            Console.Clear()
            Console.Write("Enter password: ")
            For i = 0 To password.Length - 1
```

```
                    Console.Write(".")
                Next
                Console.SetCursorPosition(password.Length + 9, 2)
            End If
            Console.ForegroundColor = Console.BackgroundColor
            subj = Console.ReadKey.Key
            Console.ForegroundColor = ConsoleColor.White
        End While
        If password = UCase(passcode) Then
            Return True
        Else
            Return False
        End If
    End Function

End Class
```

5.1.2 Node Class

```
Public Class Node
    Private Neighbourhood As New List(Of String)
    Private costFromStart As Integer
    Private name As String
    Private PreviousNode As String


    Sub New(ByVal identity As String, ByVal town As Integer(,), ByVal
townSize As Integer, ByVal distanceFromStartinput As Integer, ByVal
prevnode As String)
        name = identity
        If identity <> "." Then
            Dim whereInMatrix As Integer = IDtoInteger(identity) - 1
            If whereInMatrix <= townSize Then
                For i = 0 To townSize - 1
                    If town(whereInMatrix, i) <> Integer.MaxValue Then
                        If IntegertoID(i + 1) <> name Then
                            Neighbourhood.Add(IntegertoID(i + 1))
                        End If
```

```
                End If
            Next
        End If
    End If
    PreviousNode = prevnode
    costFromStart = distanceFromStartinput
End Sub

Sub NodeCopy(ByVal Swap As Node)
    Neighbourhood = Swap.Neighbours
    name = Swap.ID
    costFromStart = Swap.Cost
    PreviousNode = Swap.Previous
End Sub
Sub Newcost(ByVal costUpdate)
    costFromStart = costUpdate
End Sub
Sub SetStartNode()
    costFromStart = 0
    PreviousNode = "_"
End Sub
Sub SetDefaultNode()
    name = "."
    costFromStart = Integer.MaxValue
    Neighbourhood.Clear()
    PreviousNode = "."
End Sub
Sub ChangePrevious(ByRef neighbourino As String)
    PreviousNode = neighbourino
End Sub


Function Neighbours() As List(Of String)
    Return Neighbourhood
End Function
Function ID() As String
    Return name
```

```
    End Function
    Function Cost() As Integer
        Return costFromStart
    End Function

    Function Previous() As String
        Return PreviousNode
    End Function

End Class
```

5.2 SQL connection setup

```
     Dim conn As New System.Data.Odbc.OdbcConnection("DRIVER={MySQL ODBC
5.3 ANSI
Driver};SERVER=localhost;PORT=3306;DATABASE=ProjDb;USER=root;PASSWORD=root
;OPTION=3;")
```

5.3 SQL injection prevention using RegEx

```
    Function SQLmaintenance(ByRef inputString As String, ByVal lengthlimit
As Integer) As String
        Dim outputstring As String = inputString
        Dim noInjection As New Regex("'+|\\+|#+|`+")
        Dim stringPasses As Boolean
        Do
            If noInjection.IsMatch(outputstring) Or outputstring.Length >
lengthlimit Then
                Console.WriteLine("Error Occured. Please re-enter this
field. Entries must meet the following requirements
entry must be no longer than " & lengthlimit & "
The following characters are not permtted:
` ' \ #")
                outputstring = Console.ReadLine
            Else
```

```
                stringPasses = True
            End If
        Loop Until stringPasses
        Return outputstring
    End Function
```

5.4 Matrix Subroutines

5.4.1 Matrix Creation

```
    Function MatrixMaker(ByVal nodetotal As Integer) As Integer(,)
        Console.WriteLine("Generating Matrix...")
        Dim matrixpasses, notAlone As Boolean
        Dim adjacencymatrix(nodetotal, nodetotal) As Integer
        Do

            For i = 0 To nodetotal - 1
                Do

                    For j = 0 To nodetotal - 1

                        If i = j Then
                            adjacencymatrix(i, j) = Integer.MaxValue
                        Else
                            If Int(Rnd() * Int(Sqrt(nodetotal) / 4) + 1) =
1 Then
                                If adjacencymatrix(i, j) = 0 Then
                                    Select Case Int(Rnd() *
Int(Sqrt(nodetotal)) + 1)
                                        Case = 1
                                            adjacencymatrix(i, j) =
Int(Rnd() * 8) + 2
                                        Case < 5
                                            adjacencymatrix(i, j) =
Int(Rnd() * 88) + 11
                                        Case Else
```

```
                                        adjacencymatrix(i, j) =
Int(Rnd() * 899) + 100
                                    End Select
                                    adjacencymatrix(j, i) =
adjacencymatrix(i, j)
                                End If
                            Else
                                adjacencymatrix(i, j) = Integer.MaxValue
                                adjacencymatrix(j, i) = adjacencymatrix(i,
j)
                            End If
                        End If


                    Next
                    For x = 0 To nodetotal - 1
                        notAlone = False
                        For y = 0 To nodetotal - 1
                            If adjacencymatrix(x, y) <> Integer.MaxValue
Then
                                notAlone = True
                            End If
                        Next
                        If notAlone = False Then
                            Exit For
                        End If
                    Next
                Loop Until notAlone
            Next

            matrixpasses = MatrixValid2(nodetotal, adjacencymatrix)
        Loop Until matrixpasses
        Console.Clear()
        Return adjacencymatrix
    End Function
```

5.4.2 Matrix Validation

```
 Function MatrixValidV2(ByRef nodetotal As Integer, ByRef
adjacencymatrix(,) As Integer)

        Dim unvisited As New Dictionary(Of String, Node)
        Dim visited As New Dictionary(Of String, Node)
        Dim nodetorestart As String = "A"
        Dim placeholderNode, nodeForPathMaking, currentnode As Node
        Dim charOfNextNodeToCheck As String
        Dim path As New List(Of String)
        currentnode = New Node(nodetorestart, adjacencymatrix, nodetotal,
Integer.MaxValue, "_")
        unvisited.Add(nodetorestart, currentnode)
        unvisited(nodetorestart).SetStartNode()
        For i = 0 To nodetotal - 1
            If Not unvisited.ContainsKey(IntegertoID(i + 1)) Then
                unvisited.Add(IntegertoID(i + 1), New Node((IntegertoID(i
+ 1)), adjacencymatrix, nodetotal, Integer.MaxValue, "_"))
            End If
        Next
        While unvisited.Count <> 0
            path.Clear()
            For i = 1 To nodetotal
                If unvisited.ContainsKey(IntegertoID(i)) Then
                    charOfNextNodeToCheck = IntegertoID(i)
                End If
            Next
            While unvisited.Count <> 0
                nodeForPathMaking = New Node(".", adjacencymatrix,
nodetotal, Integer.MaxValue, "_")
                placeholderNode = New Node(".", adjacencymatrix,
nodetotal, Integer.MaxValue, "_")
                For Each key In unvisited.Keys
                    If placeholderNode.Cost > unvisited(key).Cost Then
                        placeholderNode.NodeCopy(unvisited(key))
                    End If
```

```
                Next

                For Each neighbour In placeholderNode.Neighbours
                    If unvisited.ContainsKey(neighbour) Then
                        If placeholderNode.Cost +
adjacencymatrix(IDtoInteger(placeholderNode.ID) - 1,
IDtoInteger(neighbour) - 1) < (unvisited(neighbour).Cost) Then

unvisited(neighbour).Newcost(placeholderNode.Cost +
adjacencymatrix(IDtoInteger(placeholderNode.ID) - 1,
IDtoInteger(neighbour) - 1))

unvisited(neighbour).ChangePrevious(placeholderNode.ID)
                        End If
                    End If
                Next

                visited.Add(placeholderNode.ID, New
Node(placeholderNode.ID, adjacencymatrix, nodetotal, placeholderNode.Cost,
placeholderNode.Previous))

                unvisited.Remove(placeholderNode.ID)
            End While
        End While

        For i = 1 To nodetotal
            If visited(IntegertoID(i)).Cost = Integer.MaxValue Then
                Console.WriteLine(visited(IntegertoID(i)))
                Console.ReadKey()
                Return False
            End If
        Next
        Return True

    End Function
```

5.5 Dijkstra's Algorithm

5.5.1 ID-Integer 2-way conversion for dijkstra

```
  Function IDtoInteger(ByVal id As String) As Integer
        Dim runningInteger As Integer
        For i = 1 To id.Length
            runningInteger += (Asc(Mid(id, i, 1)) - 64) * 26 ^ (id.Length
- i)
        Next
        Return runningInteger
    End Function

 Function IntegertoID(ByVal startingnum As Integer)
        Dim runningString As String = ""
        Dim runningInteger As Integer = startingnum
        Dim keepInOrder As Integer = 0
        Dim stringfinished As Boolean
        While Not stringfinished
            If runningInteger < 26 ^ 7 And runningInteger > 26 ^ 6 Then
                If runningInteger Mod (26 ^ 6) = 0 Then
                    keepInOrder = 1
                End If
                runningString &= Chr(Int(runningInteger / 26 ^ 6) + 64 -
keepInOrder)
                runningInteger = (runningInteger + keepInOrder) Mod (26 ^
6)
                keepInOrder = 0
            ElseIf runningInteger < 26 ^ 6 And runningInteger > 26 ^ 5
Then
                If runningInteger Mod (26 ^ 5) = 0 Then
                    keepInOrder = 1
                End If
                runningString &= Chr(Int(runningInteger / 26 ^ 5) + 64 -
keepInOrder)
                runningInteger = (runningInteger + keepInOrder) Mod (26 ^
```

```
5)
                keepInOrder = 0
            ElseIf runningInteger < 26 ^ 5 And runningInteger > 26 ^ 4
Then
                If runningInteger Mod (26 ^ 4) = 0 Then
                    keepInOrder = 1
                End If
                runningString &= Chr(Int(runningInteger / 26 ^ 4) + 64 -
keepInOrder)
                runningInteger = (runningInteger + keepInOrder) Mod (26 ^
4)
                keepInOrder = 0
            ElseIf runningInteger < 26 ^ 4 And runningInteger > 26 ^ 3
Then
                If runningInteger Mod (26 ^ 3) = 0 Then
                    keepInOrder = 1
                End If
                runningString &= Chr(Int(runningInteger / 26 ^ 3) + 64 -
keepInOrder)
                runningInteger = (runningInteger + keepInOrder) Mod (26 ^
3)
                keepInOrder = 0
            ElseIf runningInteger < 26 ^ 3 And runningInteger > 26 ^ 2
Then
                If runningInteger Mod (26 ^ 2) = 0 Then
                    keepInOrder = 1
                End If
                runningString &= Chr(Int(runningInteger / 26 ^ 2) + 64 -
keepInOrder)
                runningInteger = (runningInteger + keepInOrder) Mod (26 ^
2)
                keepInOrder = 0
            ElseIf runningInteger < 26 ^ 2 And runningInteger > 26 Then
                If runningInteger Mod 26 = 0 Then
                    keepInOrder = 1
                End If
                runningString &= Chr(Int(runningInteger / 26) + 64 -
```

```
keepInOrder)
                runningInteger = (runningInteger) Mod 26
            ElseIf runningInteger <= 26 Then
                If runningInteger = 0 Then
                    runningString &= "Z"
                Else
                    runningString &= Chr(runningInteger + 64)
                End If
                stringfinished = True
            End If
        End While
        Return runningString
    End Function
```

5.5.2 Dijkstra demonstration initiation

```
    Sub DijkstraExample()

        Randomize()
        Dim choice1Passed As Boolean
        Dim choice As ConsoleKey
        Dim listToMatricize As New List(Of Integer)
        Dim scale, iterations, tempInt As Integer
        Dim unvisited As New Dictionary(Of String, Node)
        Dim visited As New Dictionary(Of String, Node)
        Dim adjacencymatrix(,) As Integer
        Dim regionName, tempstring As String
        Dim nodelist As New List(Of String)
        Dim abstractText As New List(Of String)


        Console.WriteLine("Would you like to
1. Generate a random region
2. Use an existing region")

        Do
            choice = Console.ReadKey.Key
            If choice = ConsoleKey.D1 Or choice = ConsoleKey.NumPad1 Then
```

```
                Do
                    Console.WriteLine("How many nodes do you want
(3-500)")
                    scale = Console.ReadLine
                    If Not (scale > 2 And scale < 501) Then
                        Console.WriteLine("Invalid. Please enter a value
between 3 and 500 inclusive.")
                    End If
                Loop Until scale > 2 And scale < 501
                Console.WriteLine("How many iterations do you want to
test?")
                iterations = Console.ReadLine
                For i = 1 To iterations
                    Console.WriteLine("What node number would you like to
be destination number " & i & "?")
                    Console.WriteLine("Remember to keep it between 1 and "
& scale)
                    Do
                        tempInt = Console.ReadLine()
                    Loop Until tempInt > 0 And tempInt < scale + 1
                    nodelist.Add(IntegertoID(tempInt))

                Next
                adjacencymatrix = MatrixMaker(scale)
                choice1Passed = True
            ElseIf choice = ConsoleKey.D2 Or choice = ConsoleKey.NumPad2
Then
                Console.WriteLine("What's the name of the region you want
to use?")
                regionName = Console.ReadLine
                Dim workersearch As New Odbc.OdbcCommand("SELECT `Region
Scale` FROM `Regions` WHERE `Region Name` = '" & regionName & "';", conn)
                Dim check = workersearch.ExecuteReader
                check.Read()
                scale = check("Region Scale") * 40 + 100
                ReDim adjacencymatrix(scale, scale)
                Using reader As BinaryReader =
```

```
                    New BinaryReader(File.Open((regionName & ".region"),
FileMode.Open))
                    For i = 0 To scale - 1
                        For j = 0 To scale - 1
                            adjacencymatrix(i, j) = reader.ReadInt32
                        Next
                    Next
                End Using
                Console.WriteLine("How many iterations do you want to
test?")
                iterations = Console.ReadLine
                For i = 1 To iterations
                    Console.WriteLine("What node number would you like to
be destination number " & i & "?")
                    Console.WriteLine("Remember to keep it between A and "
& IntegertoID(scale) & ")")
                    Do
                        tempstring = Console.ReadLine()

                    Loop Until IDtoInteger(tempstring) > 0 And
IDtoInteger(tempstring) < scale + 1
                    nodelist.Add(tempstring)

                Next
                choice1Passed = True
            End If
        Loop Until choice1Passed

        abstractText = DijkstraChain(unvisited, visited, scale,
adjacencymatrix, iterations, nodelist, "A", 1)


        For i = 0 To scale - 1
            Console.Write(IntegertoID(i + 1) & ":   ")
            For j = 0 To scale - 1
                If adjacencymatrix(i, j) = Integer.MaxValue Then
                    Console.Write("N/A, ")
```

```
                Else
                    Console.Write(adjacencymatrix(i, j) & ", ")
                End If
            Next
            Console.WriteLine()
        Next

        For i = 0 To abstractText.Count - 1
            Console.WriteLine(abstractText(i))
        Next
        Console.WriteLine("Press any key to end the simulation")

        Console.ReadKey()
        Console.Clear()
    End Sub
```

5.5.3 Dijkstra algorithm in practice

```
    Function DijkstraChain(ByRef unvisited As Dictionary(Of String, Node),
ByRef visited As Dictionary(Of String, Node), ByRef nodetotal As Integer,
ByRef adjacencymatrix(,) As Integer, ByRef iterations As Integer, ByRef
nodelist As List(Of String), ByVal StartNode As String, ByVal mode As
Integer)
        Dim nodetorestart As String = StartNode
        Dim currentnode, placeholderNode As Node
        Dim tempid As String
        Dim counter As Integer
        Dim abstractText As New List(Of String)
        Dim path As New List(Of String)
        While iterations > 0 And nodelist.Count <> 0
            path.Clear()
            unvisited.Clear()
            visited.Clear()
            currentnode = New Node(nodetorestart, adjacencymatrix,
nodetotal, Integer.MaxValue, "_")
            unvisited.Add(nodetorestart, currentnode)
            unvisited(nodetorestart).SetStartNode()
            For i = 0 To nodetotal - 1
```

```
                tempid = IntegertoID(i + 1)
                If Not unvisited.ContainsKey(tempid) Then
                    unvisited.Add(tempid, New Node((tempid),
adjacencymatrix, nodetotal, Integer.MaxValue, "_"))
                End If
            Next

            For i = 0 To nodetotal - 1
                For j = 0 To nodetotal - 1
                    If adjacencymatrix(i, j) = 0 Then
                        adjacencymatrix(i, j) = Integer.MaxValue
                    End If
                Next
            Next


            While unvisited.Count <> 0 And nodelist.Count <> 0
                placeholderNode = New Node(".", adjacencymatrix,
nodetotal, Integer.MaxValue, "_")
                For Each key In unvisited.Keys
                    If placeholderNode.Cost > unvisited(key).Cost Then
                        placeholderNode.NodeCopy(unvisited(key))
                    End If
                Next

                For Each neighbour In placeholderNode.Neighbours
                    If unvisited.ContainsKey(neighbour) Then
                        If placeholderNode.Cost +
adjacencymatrix(IDtoInteger(placeholderNode.ID) - 1,
IDtoInteger(neighbour) - 1) < (unvisited(neighbour).Cost) Then

unvisited(neighbour).Newcost(placeholderNode.Cost +
adjacencymatrix(IDtoInteger(placeholderNode.ID) - 1,
IDtoInteger(neighbour) - 1))

unvisited(neighbour).ChangePrevious(placeholderNode.ID)
                        End If
```

```
                    End If
                Next
                visited.Add(placeholderNode.ID, New
Node(placeholderNode.ID, adjacencymatrix, nodetotal, placeholderNode.Cost,
placeholderNode.Previous))

                unvisited.Remove(placeholderNode.ID)


            End While
            If mode = 2 Then
                Return visited(nodelist(0)).Cost
            End If
            abstractText.Add("The Shortest Distance from node " &
nodetorestart & " to node " & nodelist(0) & " is " &
visited(nodelist(0)).Cost & " and the node journey is as follows")

            path.Add(visited(nodelist(0)).ID)
            placeholderNode.NodeCopy(visited(nodelist(0)))

            While placeholderNode.Previous <> "_"
                path.Add(placeholderNode.Previous)

placeholderNode.NodeCopy(visited(placeholderNode.Previous))
                counter += 1
            End While
            For i = path.Count - 1 To 0 Step -1
                abstractText.Add(path(i))
            Next
            iterations -= 1
            If nodelist.Count > 0 Then
                nodetorestart = nodelist(0)
            End If
            nodelist.RemoveAt(0)

        End While
        Return abstractText
```

```
    End Function
```

5.6 Company Creation

```
    Sub NewCompany()
        Randomize()
        Dim codeUsed As Boolean
        Dim query As New Odbc.OdbcCommand("SELECT CompanyID FROM
Companies", conn)
        Dim checkunused As Odbc.OdbcDataReader
        Dim randomcode As String = ""
        Dim companyName As String
        Console.WriteLine("What is the name of your Company?")
        companyName = SQLmaintenance(Console.ReadLine, 100)
        Do
            For i = 1 To 5
                Select Case (Int(Rnd() * 30) + 1) Mod 6
                    Case 0
                        randomcode &= Int(Rnd() * 10)
                    Case Else
                        randomcode &= Chr(Int(Rnd() * 25) + 65)
                End Select
            Next
            checkunused = query.ExecuteReader
            While checkunused.Read
                If checkunused("CompanyID") = randomcode Then
                    codeUsed = True
                End If
            End While
            checkunused.Close()
        Loop Until Not codeUsed
        Dim nonquery As New Odbc.OdbcCommand("INSERT INTO
Companies(CompanyID,Company) VALUES ('" & randomcode & "', '" &
companyName & "');", conn)
        nonquery.ExecuteNonQuery()
        Console.WriteLine("Company creation successful, your company ID is
" & randomcode & ". KEEP THIS SAFE! You will need it to register
```

```
employees.")
        Console.WriteLine()
        Console.WriteLine("An Admin account must now be registered to the
company, please enter your details")

        AddEmployee("Admin")
    End Sub
```

5.7 User Management

5.7.1 Creation

```
 Sub AddUser(ByVal userLevel As String)

        Dim truth1 As Boolean
        Dim query As String = ""
        Dim companycode As String
        Dim check, check2 As Odbc.OdbcDataReader
        Console.WriteLine("What is your Legal Name?")
        query &= ("'" & SQLmaintenance(Console.ReadLine, 100) & "', ")
        Dim compsearch As New Odbc.OdbcCommand("SELECT CompanyID FROM
Companies", conn)
        check = compsearch.ExecuteReader

        Do
            Console.WriteLine("What is your company ID (Receive this via
your company's administrative team")
            companycode = Console.ReadLine
            While check.Read And Not truth1
                If check("CompanyID") = companycode Then
                    truth1 = True
                End If
            End While
            If truth1 Then
                Dim compnameget As New Odbc.OdbcCommand("SELECT Company
FROM Companies WHERE CompanyID = '" & companycode & "'", conn)
                check2 = compnameget.ExecuteReader
```

```
                check2.Read()
                query = query & ("'" & check2("Company") & "', ")
            Else
                Console.WriteLine("Company not found. Retry code or
contact company representative.")
                check.Close()
                check = compsearch.ExecuteReader
            End If
        Loop Until truth1
        query &= ("'" & userLevel & "', ")
        Console.WriteLine("Please enter a Username")
        query &= ("'" & SQLmaintenance(Console.ReadLine, 30) & "', ")
        Console.WriteLine("Please enter a Passcode")
        query &= ("'" & SQLmaintenance(Console.ReadLine, 45) & "', ")
        Console.WriteLine("Please enter a Unique User Code (1-2 letters,
signifies your home address)")
        query &= ("'" & UCase(SQLmaintenance(Console.ReadLine, 2)) & "',
")
        Console.WriteLine("Please enter a Phone number")
        query &= ("'" & SQLmaintenance(Console.ReadLine, 11) & "'")
        Dim command As New Odbc.OdbcCommand("INSERT INTO Users(`Legal
Name`,`Company`,`User Level`,`Username`,`Passcode`, `UUC`, `Phone Num`)
Values (" & query & ");", conn)
        command.ExecuteNonQuery()
        Console.WriteLine("Successfully Added! Press any key to return to
Main Menu.")
        Console.ReadKey()
        Console.Clear()
    End Sub
```

5.7.2 Credential Checker

```
   Sub SearchUsers(ByVal username As String)
        Dim passcode As String
        Dim query As String = "SELECT * From Users WHERE Username = "
        Dim checker As Odbc.OdbcDataReader
        query &= ("'" & username & "' AND Passcode = ")
        Console.WriteLine("Please enter your Passcode")
```

```
        passcode = SQLmaintenance(Console.ReadLine, 45)
        query &= ("'" & passcode & "';")
        Dim searchTerm As New Odbc.OdbcCommand(query, conn)
        checker = searchTerm.ExecuteReader
        If checker.Read() Then
            Console.WriteLine("
            " & checker("UUC") & " | " & checker("Legal Name") & " | " &
checker("Company") & " | " & checker("User Level") & " | " &
checker("Username") & " | " & checker("Passcode") & " | " & checker("Phone
Num"))
            Console.WriteLine("
Press any key to return to main menu")
            Console.ReadKey()
        Else
            Console.WriteLine("No match for these credentials. Check your
credentials and try again")
            Console.ReadKey()
        End If
    End Sub
5.7
```

5.8 Client Creation

```
 Sub AddClient(ByVal company As String)
        Dim choice As ConsoleKey
        Dim query As String = ""
        Console.WriteLine("What is the Client's Company Name?")
        Dim client As String = SQLmaintenance(Console.ReadLine, 200)
        query &= ("'" & client & "', ")
        Console.WriteLine("What is the Client's address?")
        query &= ("'" & SQLmaintenance(Console.ReadLine, 200) & "', ")
        Console.WriteLine("What is the Name of the Region the client is
positioned in?")
        query &= ("'" & SQLmaintenance(Console.ReadLine, 50) & "'")
        Console.WriteLine("Do you have the clients credit details, and
permission to hold them? (Y/N)")
        While choice <> ConsoleKey.Y And choice <> ConsoleKey.N
```

```
            choice = Console.ReadKey.Key
            If choice = ConsoleKey.Y Then
                Console.WriteLine("What is the Client's Credit Card
Number?")
                query &= (", '" & SQLmaintenance(Console.ReadLine, 16) &
"', ")
                Console.WriteLine("What is the Client's Credit Card
Expiration Date?")
                query &= ("'" & SQLmaintenance(Console.ReadLine, 5) & "',
")
                Console.WriteLine("What is the Client's CVC?")
                query &= ("'" & SQLmaintenance(Console.ReadLine, 3) & "'")
            End If
        End While
        Dim command As New Odbc.OdbcCommand("INSERT INTO
Clients(`ClientName`, `address`, `Region Name`) Values (" & query & ");",
conn)

        command.ExecuteNonQuery()
        Dim command2 As New Odbc.OdbcCommand("INSERT INTO
CompanyClient(`Company`, `clientname`) Values ('" & company & "', '" &
client & "');", conn)
        command2.ExecuteNonQuery()
        Console.WriteLine("Please enter a Unique Client Code (1-2 letters,
signifies their map address)")
        Dim ucc As String = SQLmaintenance(Console.ReadLine(), 2)
        Dim command3 As New Odbc.OdbcCommand("INSERT INTO `UCCs`(`UCC`,
`clientname`) Values ('" & ucc & "', '" & client & "');", conn)
        command3.ExecuteNonQuery()
        Console.WriteLine("Client Successfully Added!")
        Console.WriteLine("Press any key to return to your hub")
        Console.ReadKey()
    End Sub
```

5.9 Team management

5.9.1 Creation

```
Sub NewTeam(ByVal company As String)
        Console.Clear()
        Dim choice As ConsoleKey
        Dim userInput, teamCodeInput As String
        Dim userValid As Odbc.OdbcDataReader
        Dim userFound, reconsidered As Boolean
        Dim searchQuery As New Odbc.OdbcCommand("", conn)
        Dim TeamAdd As New Odbc.OdbcCommand("", conn)
        Do
            Console.WriteLine("What is the Legal Name of the first
employee in your new team?")
            userInput = SQLmaintenance(Console.ReadLine, 30)
            searchQuery.CommandText = "SELECT `Username`, `Legal Name`
FROM Users WHERE `Legal Name` = '" & userInput & "';"
            userValid = searchQuery.ExecuteReader
            userValid.Read()
            If userValid("Legal Name") = userInput Then
                Console.WriteLine("User Found")
                userFound = True
                Console.WriteLine("Please enter your 8 character team
code?")
                teamCodeInput = SQLmaintenance(Console.ReadLine, 8)
                TeamAdd.CommandText = "INSERT INTO Teams Values ('" &
company & "', '" & teamCodeInput & "');"
                TeamAdd.ExecuteNonQuery()
                TeamAdd.CommandText = "INSERT INTO TeamMapping Values ('"
& teamCodeInput & "', '" & userValid("Username") & "');"
                TeamAdd.ExecuteNonQuery()
            Else
                Console.WriteLine("User not found. Press Esc to quit team
making, or any other key to reenter the teammate's Legal Name.")
                choice = Console.ReadKey.Key

            End If
        Loop Until userFound Or reconsidered Or choice = ConsoleKey.Escape
```

```
        If choice <> ConsoleKey.Escape Then
            choice = ConsoleKey.A
            Do
                Console.WriteLine("Press Esc to quit team making, or any
other key to add another team member")
                choice = Console.ReadKey.Key
                If choice = ConsoleKey.D1 Or choice = ConsoleKey.NumPad1
Then


                    Console.WriteLine("What is the Legal Name of the next
employee in your team?")
                    userInput = SQLmaintenance(Console.ReadLine, 30)
                    searchQuery.CommandText = "SELECT ' FROM Users Where
`Legal Name` = '" & userInput & "';"
                    userValid = searchQuery.ExecuteReader
                    If userValid("Legal Name") = userInput Then
                        Console.WriteLine("User Found")
                        userFound = True
                        TeamAdd.CommandText = "INSERT INTO TeamMapping
Values ('" & teamCodeInput & "', '" & userInput & "');"
                        TeamAdd.ExecuteNonQuery()
                    Else
                        Console.WriteLine("User not found. Press Esc to
quit team making, or any other key to reenter the teammate's Legal Name.")
                        choice = Console.ReadKey.Key
                    End If
                End If
            Loop Until choice = ConsoleKey.Escape
        End If
    End Sub
```

5.9.2 Viewing and editing

```
Sub TeamComp(ByVal company As String)
        Dim goAhead As Boolean
        Dim teamsame As String = ""
        Dim choice, answer As ConsoleKey
        Dim tempstring As String
```

```
        Dim nonquery As String = "INSERT INTO `TeamMapping`(`Username`,
`TeamID`) Values ("
        Dim TeamGatherer As New Odbc.OdbcCommand("SELECT `TeamID` FROM
TeamMapping ORDER BY TeamID DESC", conn)
        Dim teamReader As Odbc.OdbcDataReader = TeamGatherer.ExecuteReader
        Dim MemberGatherer As New Odbc.OdbcCommand("SELECT `TeamID`,
`UUC`, `Legal Name`, `User Level`, `Phone Num` FROM Users, TeamMapping
WHERE Users.username = TeamMapping.username and company = '" & company &
"' ORDER BY 'User Level' ASC;", conn)
        Dim memberReader As Odbc.OdbcDataReader =
MemberGatherer.ExecuteReader
        Dim TeamlessGatherer As New Odbc.OdbcCommand("SELECT `UUC`, `Legal
Name`, `User Level`, `Phone Num` FROM `users` WHERE Users.username NOT
IN(SELECT `Username` FROM `TeamMapping`);", conn)
        Dim teamlessReader As Odbc.OdbcDataReader =
TeamlessGatherer.ExecuteReader
        Dim TempCommand As New Odbc.OdbcCommand("", conn)
        Dim tempreader As Odbc.OdbcDataReader
        While teamReader.Read
            If teamsame <> teamReader("teamID") Then
                goAhead = True
                Console.WriteLine(teamReader("teamID"))
                teamsame = teamReader("teamID")
            Else
                goAhead = False
            End If
            While memberReader.Read And goAhead
                If memberReader("TeamID") = teamReader("TeamID") Then
                    Console.WriteLine(memberReader("UUC") & "  |  " &
memberReader("User Level") & "  |  " & memberReader("Legal Name") & "  |
" & memberReader("Phone Num") & "  |  ")
                End If
            End While
            memberReader.Close()
            memberReader = MemberGatherer.ExecuteReader
        End While
        teamReader.Close()
```

```
        teamReader = TeamGatherer.ExecuteReader
        While teamlessReader.Read
            If Not goAhead Then
                Console.WriteLine("TEAMLESS")
                goAhead = True
            End If
            Console.WriteLine(teamlessReader("UUC") & "  |  " &
teamlessReader("User Level") & "  |  " & teamlessReader("Legal Name") & "
|  " & teamlessReader("Phone Num") & "  |  ")
        End While
        Console.WriteLine()
        Console.WriteLine("Are These Teams Adequate? (Y/N)")
        Console.ForegroundColor = Console.BackgroundColor
        answer = Console.ReadKey.Key
        Console.ForegroundColor = ConsoleColor.White
        Console.WriteLine()

        If answer = ConsoleKey.N Then
            Do
                Console.WriteLine("Do you wish to
1. Add a new Team
2. Reassign workers to another Team
0. Return to menu")
                choice = Console.ReadKey.Key
                If choice = ConsoleKey.D1 Or choice = ConsoleKey.NumPad1
Then
                    NewTeam(company)
                ElseIf choice = ConsoleKey.D2 Or choice =
ConsoleKey.NumPad2 Then
                    Console.WriteLine()
                    While teamReader.Read
                        If teamsame <> teamReader("teamID") Then
                            goAhead = True
                            Console.WriteLine(teamReader("teamID"))
                            teamsame = teamReader("teamID")
                        Else
                            goAhead = False
```

```
                        End If
                        While memberReader.Read And goAhead
                            If memberReader("TeamID") =
teamReader("TeamID") Then
                                Console.WriteLine(memberReader("UUC") & "
|  " & memberReader("User Level") & "  |  " & memberReader("Legal Name") &
"  |  " & memberReader("Phone Num") & "  |  ")
                            End If
                        End While
                        memberReader.Close()
                        memberReader = MemberGatherer.ExecuteReader
                    End While
                    teamReader.Close()
                    teamReader = TeamGatherer.ExecuteReader
                    Do
                        Console.WriteLine("Do you wish to
1. Add a user to a team
2. Remove a user from an existing team
0. Return to menu")
                        choice = Console.ReadKey.Key
                        If choice = ConsoleKey.D1 Or choice =
ConsoleKey.NumPad1 Then
                            goAhead = False
                            Do
                                Console.WriteLine("Write the UUC of the
user you'd like to add to a team")
                                tempstring = Console.ReadLine
                                TempCommand.CommandText = ("SELECT `UUC`
FROM `Users`")
                                tempreader = TempCommand.ExecuteReader
                                While tempreader.Read And goAhead = False
                                    If tempreader("UUC") = tempstring Then
                                        goAhead = True
                                    End If
                                End While
                                If Not goAhead Then
                                    Console.WriteLine("User not found. Try
```

```
again")
                                End If
                            Loop Until goAhead
                            tempreader.Close()
                            nonquery &= "(SELECT Username FROM Users WHERE
UUC = '" & tempstring & "'), '"
                            goAhead = False
                            Do
                                Console.WriteLine("Write the Team ID of
the team you'd like to add t")
                                tempstring = Console.ReadLine
                                TempCommand.CommandText = ("SELECT
`TeamID` FROM `Teams`")
                                tempreader = TempCommand.ExecuteReader
                                While tempreader.Read And goAhead = False
                                    If tempreader("TeamID") = tempstring
Then
                                        goAhead = True
                                    End If
                                End While
                                If Not goAhead Then
                                    Console.WriteLine("Team ID not found.
Try again")
                                End If
                                tempreader.Close()
                            Loop Until goAhead
                            tempreader.Close()
                            nonquery &= tempstring & "');"
                            TempCommand.CommandText = nonquery
                            TempCommand.ExecuteNonQuery()
                            Console.WriteLine("Complete. Press any key to
return to menu.")
                            Console.ReadKey()
                            choice = ConsoleKey.D0
                        ElseIf choice = ConsoleKey.D2 Or choice =
ConsoleKey.NumPad2 Then
                            goAhead = False
```

```
                            Do
                                Console.WriteLine("Please enter the UUC of
the user you'd like to leave teamless")
                                tempstring = Console.ReadLine
                                TempCommand.CommandText = ("SELECT `UUC`
FROM `Users`")
                                tempreader = TempCommand.ExecuteReader
                                While tempreader.Read And goAhead = False
                                    If tempreader("UUC") = tempstring Then
                                        goAhead = True
                                    End If
                                End While
                                If Not goAhead Then
                                    Console.WriteLine("User not found. Try
again")
                                End If
                            Loop Until goAhead
                            tempreader.Close()
                            TempCommand.CommandText = "DELETE FROM
TeamMapping WHERE Username = (SELECT `Username` FROM `Users` WHERE `UUC` =
'" & tempstring & "');"
                            TempCommand.ExecuteNonQuery()
                            Console.WriteLine("Complete. Press any key to
return to menu.")
                            Console.ReadKey()
                        End If
                    Loop Until choice = ConsoleKey.D0 Or choice =
ConsoleKey.NumPad0
                End If
            Loop Until choice = ConsoleKey.D0 Or choice =
ConsoleKey.NumPad0
        End If
    End Sub
```

5.10 Region Creation

```
    Sub RegionCreator()

        Dim team, regionName As String
        Dim regionMatrix(,) As Integer
        Dim query As String = ""
        Dim scale As Integer
        Dim isUnique As Boolean
        Dim choice As ConsoleKey
        Do
            Console.WriteLine("What would you like to call this region?")
            regionName = SQLmaintenance(Console.ReadLine, 50)
            Try
                isUnique = False
                Using reader As BinaryReader =
            New BinaryReader(File.Open((regionName & ".region"),
FileMode.Open))
                End Using
            Catch noFile As FileNotFoundException
                isUnique = True
            End Try
            If Not isUnique Then
                Console.WriteLine("Region Already exists. Do you want to
1. Update Region
2. Enter new Region Name

Or press Backspace to go back")
                choice = Console.ReadKey.Key
                If choice = ConsoleKey.D1 Or choice = ConsoleKey.NumPad1
Then
                    isUnique = True
                End If
            ElseIf choice = ConsoleKey.Backspace Then
                Exit Sub
            End If
```

```
        Loop Until isUnique

        query &= "'" & regionName & "', "
        Console.WriteLine("What Scale (Between 1 and 10) do you want your
region to be?")
        Do
            scale = Console.ReadLine
            If scale < 1 Or scale > 10 Then
                Console.WriteLine("Please stay within the bounds of 1 and
10")
            End If
        Loop Until scale > 0 And scale < 11
        query &= "'" & scale & "', "
        Console.WriteLine("What's the Team Code of the Team that you want
to cover this region?")
        team = SQLmaintenance(Console.ReadLine, 8)
        query &= "'" & team & "'"
        Dim command2 As New Odbc.OdbcCommand("INSERT INTO Regions Values
(" & query & ");", conn)
        command2.ExecuteNonQuery()
        scale = 100 + scale * 40
        regionMatrix = MatrixMaker(scale)
        Using writer As BinaryWriter =
                New BinaryWriter(File.Open(regionName & ".region",
FileMode.OpenOrCreate))
            For i = 0 To scale - 1
                For j = 0 To scale - 1
                    writer.Write(regionMatrix(i, j))
                Next
            Next
        End Using
        Console.WriteLine("Region Created. Press any key to exit")
        Console.ReadKey()
    End Sub
```

5.11 Job Management

5.11.1 Creation

```
  Sub AddJob()
        Dim choice1Passes, choice2Passes As Boolean
        Dim jobID, codeRead, client, uccRead, spacer As String
        Dim IDTerm As New Odbc.OdbcCommand("SELECT `JobID`, `JobDesc` FROM
`JobTypes`;", conn)
        Dim UCCTerm As New Odbc.OdbcCommand("SELECT `ClientName`, `UCC`
FROM `UCCs`;", conn)
        Dim teamTerm As New Odbc.OdbcCommand("SELECT `TeamID` FROM
`Regions`;", conn)
        Dim searcher As Odbc.OdbcDataReader
        Do
            searcher = IDTerm.ExecuteReader
            Console.WriteLine("Add the job type that the job is, or enter
? to see all job types.")
            jobID = SQLmaintenance(Console.ReadLine, 8)
            If jobID = "?" Then
                While searcher.Read
                    spacer = "       "
                    codeRead = searcher("JobID")
                    spacer = codeRead & Mid(spacer, codeRead.Length, 8 -
codeRead.Length)
                    Console.WriteLine(spacer & searcher("JobDesc"))
                End While
                searcher.Close()
                searcher = IDTerm.ExecuteReader
            Else
                While searcher.Read
                    If jobID = searcher("JobID") Then
                        choice1Passes = True
                    End If
                End While
                If Not choice1Passes Then
                    Console.WriteLine("Job type not found. Press any key
```

```
to try again.")
                    Console.ReadKey()
                End If
            End If
            searcher.Close()
        Loop Until choice1Passes
        Do
            searcher = UCCTerm.ExecuteReader
            Console.WriteLine("Enter the UCC of the client to be served or
press ? to see all UCCs")
            uccRead = SQLmaintenance(Console.ReadLine, 8)
            If uccRead = "?" Then
                While searcher.Read
                    spacer = "    "
                    uccRead = searcher("UCC")
                    spacer = uccRead & spacer
                    Console.WriteLine(spacer & searcher("ClientName"))
                End While
                searcher.Close()
                searcher = UCCTerm.ExecuteReader
            Else
                While searcher.Read
                    If uccRead = searcher("UCC") Then
                        choice2Passes = True
                    End If
                End While
                If Not choice2Passes Then
                    Console.WriteLine("Client not found. Press any key to
try again.")
                    Console.ReadKey()
                End If
            End If
            searcher.Close()
        Loop Until choice2Passes

        searcher = UCCTerm.ExecuteReader
        While searcher.Read
```

```
            If uccRead = searcher("UCC") Then
                client = searcher("ClientName")
            End If
        End While

        Dim jobEntry As New Odbc.OdbcCommand("INSERT INTO `JobsDone`
(`JobID`, `ClientName`) VALUES(" & jobID & ", '" & client & "');", conn)
        jobEntry.ExecuteNonQuery()
        Console.WriteLine("Job Added Successfully. Press any key to return
to the main menu.")
        Console.ReadKey()
        Console.Clear()
    End Sub
```

5.11.2 Marking Completion

```
    Sub JobLog(ByVal username As String)
        Dim doYouEvenWorkHere As Boolean
        Dim UJCtoMark As Integer
        Dim jobstart As String
        Dim line As String = ""
        Console.WriteLine("Please enter the UJC number of the job you've
completed")
        Console.WriteLine("UJC       |JobID|Client Name")
        Dim jobstoMark As New Odbc.OdbcCommand("SELECT `UJC`, `JobID`,
`ClientName` FROM `JobsDone` WHERE `Username` = '" & username & "' AND
`JobComplete` = FALSE;", conn)
        Dim searcher As Odbc.OdbcDataReader = jobstoMark.ExecuteReader
        While searcher.Read
            doYouEvenWorkHere = True
            line = searcher("UJC")
            For i = 1 To 10 - Convert.ToString(searcher("UJC")).Length
                line &= " "
            Next
            line &= "|" & searcher("JobID")
            For i = 1 To 5 - Convert.ToString(searcher("JobID")).Length
                line &= " "
            Next
```

```
            line &= "|" & searcher("ClientName")
            Console.WriteLine(line)

        End While
        searcher.Close()
        If doYouEvenWorkHere Then
            UJCtoMark = Console.ReadLine
            jobstoMark.CommandText = "SELECT `UJC` FROM `JobsDone` WHERE
`Username` = '" & username & "' AND `JobComplete` = FALSE AND `UJC`= " &
UJCtoMark & ";"
            searcher = jobstoMark.ExecuteReader
            searcher.Read()
            Console.WriteLine("What time did you start the job (in the
format XX:XX)?")
            jobstart = Console.ReadLine
            Dim jobCompleter As New Odbc.OdbcCommand("UPDATE `JobsDone`
SET `JobComplete` = TRUE, `WorkDate` = CAST(CURRENT_TIMESTAMP As Date),
`WorkStart` = '" & jobstart & "',`WorkEnd` = CAST(CURRENT_TIMESTAMP As
Time(0)) WHERE `UJC` = " & searcher("UJC") & ";", conn)
            jobCompleter.ExecuteNonQuery()
            Console.WriteLine("Job Completion Logged! Press any key to
return to main menu.")
        Else
            Console.Clear()
            Console.WriteLine("You currently have no assigned jobs. Press
any key to return to main menu.")
        End If
        Console.ReadKey()
        Console.Clear()
    End Sub
```

5.11.3 Viewing

5.11.3.1 Employee level

```
  Sub ShowJobs(ByVal username As String)
        Dim adjacencymatrix(,) As Integer
        Dim TempCommand As New Odbc.OdbcCommand("", conn)
```

```
        Dim tempreader As Odbc.OdbcDataReader
        Dim tempCurrentPos As String
        Dim tempDestinationlist As New List(Of String)
        tempDestinationlist.Add("")
        TempCommand.CommandText = "SELECT UUC FROM `Users` WHERE
`Username` = '" & username & "';"
        tempreader = TempCommand.ExecuteReader
        tempreader.Read()
        tempCurrentPos = tempreader("UUC")
        tempreader.Close()
        TempCommand.CommandText = "SELECT * FROM `JobsDone` WHERE Username
= '" & username & "' AND Jobcomplete = FALSE;"
        tempreader = TempCommand.ExecuteReader
        If tempreader.Read Then
            Console.WriteLine("UJC| JobID |  Client Name")
        End If
        tempreader.Close()
        tempreader = TempCommand.ExecuteReader
        While tempreader.Read

            Console.WriteLine(tempreader("UJC") & "|  " &
tempreader("JobID") & "  |  " & tempreader("ClientName"))
        End While
        tempreader.Close()
        TempCommand.CommandText = "SELECT `UCC` FROM `UCCs`, `JobsDone`
WHERE `JobsDone`.`Username` = '" & username & "' AND `JobsDone`.`WorkDate`
= CAST(CURRENT_TIMESTAMP As Date) AND `JobsDone`.`ClientName` =
`UCCs`.`ClientName` AND `JobsDone`.`JobComplete` = TRUE;"
        tempreader = TempCommand.ExecuteReader
        While tempreader.Read
            tempCurrentPos = tempreader("UCC")
        End While
        tempreader.Close()
        TempCommand.CommandText = "SELECT `UCC` FROM `UCCs`, `JobsDone`
WHERE `JobsDone`.`Username` = '" & username & "' AND
`JobsDone`.`ClientName` = `UCCs`.`ClientName` AND `JobsDone`.`JobComplete`
= FALSE;"
```

```
        tempreader = TempCommand.ExecuteReader
        tempreader.Read()
        tempDestinationlist(0) = tempreader("UCC")
        tempreader.Close()
        TempCommand.CommandText = "SELECT `Region Scale`, `Region Name`
FROM `Regions`, `TeamMapping` WHERE `TeamMapping`.`Username` = '" &
username & "' AND `TeamMapping`.`TeamID` = `Regions`.`TeamID`;"
        tempreader = TempCommand.ExecuteReader
        tempreader.Read()
        Dim scale As Integer = tempreader("Region Scale") * 40 + 100
        ReDim adjacencymatrix(scale, scale)
        Using reader As BinaryReader =
            New BinaryReader(File.Open((tempreader("Region Name") &
".region"), FileMode.Open))
            For i = 0 To scale - 1
                For j = 0 To scale - 1
                    adjacencymatrix(i, j) = reader.ReadInt32
                Next
            Next
        End Using
        If tempDestinationlist(0) = "" Then
            tempDestinationlist(0) = tempCurrentPos
        End If
        If DijkstraChain(New Dictionary(Of String, Node), New
Dictionary(Of String, Node), scale, adjacencymatrix, 1,
tempDestinationlist, tempCurrentPos, 2) <> 0 Then
            Console.WriteLine("Your next journey is as follows:

")
            Dim abstractText As List(Of String) = DijkstraChain(New
Dictionary(Of String, Node), New Dictionary(Of String, Node), scale,
adjacencymatrix, 1, tempDestinationlist, tempCurrentPos, 1)
            For i = 0 To abstractText.Count - 1
                Console.WriteLine(abstractText(i))
            Next
            Console.WriteLine("
```

```
Press any key to return to the main menu")
            Console.ReadKey()
            Console.Clear()
        Else
            Console.WriteLine("You currently have nowhere to go to.")
            Console.WriteLine("

Press any key to return to the main menu")
            Console.ReadKey()
            Console.Clear()
        End If
    End Sub
```

5.11.3.2 Admin Level (Productivity Check)

```
    Sub ProductivityCheck(ByVal company As String)
        Dim choice As ConsoleKey
        Dim TempCommand As New Odbc.OdbcCommand("", conn)
        Dim tempreader As Odbc.OdbcDataReader
        Do
            Console.WriteLine("Would you like to view:
1. Completed Jobs
2. Uncompleted Assigned Jobs
3. Uncompleted Unassigned Jobs
0 Return To Main Menu")
            choice = Console.ReadKey.Key
            Console.Clear()

            If choice = ConsoleKey.D1 Or choice = ConsoleKey.NumPad1 Then
                TempCommand.CommandText = "SELECT `UJC`, `Legal Name`,
`ClientName`, `JobID`, `WorkDate`, `WorkStart`, `WorkEnd` FROM `JobsDone`,
`Users` WHERE `JobsDone`.`Username` = `Users`.`Username` AND
`JobsDone`.`JobComplete` = True;"
                tempreader = TempCommand.ExecuteReader
                If tempreader.Read Then
                    tempreader.Close()
                    tempreader = TempCommand.ExecuteReader
                    Console.WriteLine("Guide: The table will be arranged
```

```
as follows
Unique Job Code  |  Legal Name  |  Client Name |  JobID  |  Work Date  |
Work Start  |  Work End

Press any key to continue")
                    Console.ReadKey()
                    Console.Clear()
                    Console.WriteLine("Guide: The table will be arranged
as follows
Unique Job Code  |  Legal Name  |  Client Name |  JobID  |  Work Date  |
Work Start  |  Work End

Press any key to continue")
                    While tempreader.Read
                        Console.WriteLine(tempreader("UJC") & "  |  " &
tempreader("Legal Name") & "  |  " & tempreader("ClientName") & "  |  " &
tempreader("JobID") & "  |  " & Convert.ToString(tempreader("WorkDate")) &
"  |  " & Convert.ToString(tempreader("WorkStart")) & "  |  " &
Convert.ToString(tempreader("WorkEnd")))
                    End While
                Else
                    Console.WriteLine("No data to be shown")
                End If
                Console.WriteLine("


Press any key to return to the productivity menu")
                Console.ReadKey()
                Console.Clear()
                tempreader.Close()

            ElseIf choice = ConsoleKey.D2 Or choice = ConsoleKey.NumPad2
Then
                TempCommand.CommandText = "SELECT `UJC`, `Legal Name`,
`ClientName`, `JobID`FROM `JobsDone`, `Users` WHERE `JobsDone`.`Username`
= `Users`.`Username` AND `JobsDone`.`JobComplete` = False;"
                tempreader = TempCommand.ExecuteReader
```

```
                If tempreader.Read Then
                    tempreader.Close()
                    tempreader = TempCommand.ExecuteReader
                    Console.WriteLine("Guide: The table will be arranged
as follows
Unique Job Code  |  Legal Name  |  Client Name |  JobID

Press any key to continue")
                    Console.ReadKey()
                    Console.Clear()
                    Console.WriteLine("Guide: The table will be arranged
as follows
Unique Job Code  |  Legal Name  |  Client Name |  JobID

Press any key to continue")
                    While tempreader.Read
                        Console.WriteLine(tempreader("UJC") & "  |  " &
tempreader("Legal Name") & "  |  " & tempreader("ClientName") & "  |  " &
tempreader("JobID"))
                    End While
                Else
                    Console.WriteLine("No data to be shown")
                End If
                Console.WriteLine("


Press any key to return to the productivity menu")
                Console.ReadKey()
                Console.Clear()
                tempreader.Close()
            ElseIf choice = ConsoleKey.D3 Or choice = ConsoleKey.NumPad3
Then
                TempCommand.CommandText = "SELECT `UJC`, `ClientName`,
`JobID`FROM `JobsDone` WHERE Username IS NULL AND `JobsDone`.`JobComplete`
= False;"
                tempreader = TempCommand.ExecuteReader
                If tempreader.Read Then
```

```
                    tempreader.Close()
                    tempreader = TempCommand.ExecuteReader
                    Console.WriteLine("Guide: The table will be arranged
as follows
Unique Job Code  |  Client Name |  JobID

Press any key to continue")
                    Console.ReadKey()
                    Console.Clear()
                    Console.WriteLine("Guide: The table will be arranged
as follows
Unique Job Code  |  Client Name |  JobID

Press any key to continue")
                    While tempreader.Read
                        Console.WriteLine(tempreader("UJC") & "  |  " &
tempreader("ClientName") & "  |  " & tempreader("JobID"))
                    End While
                Else
                    Console.WriteLine("No data to be shown")
                End If
                Console.WriteLine("


Press any key to return to the productivity menu")
                Console.ReadKey()
                tempreader.Close()
                Console.Clear()
            End If

        Loop Until choice = ConsoleKey.D0 Or choice = ConsoleKey.NumPad0
        Console.Clear()
    End Sub
```

5.12 Qualification management (Viewing and Assigning)

```
    Sub QualificationHub(ByVal company As String)
```

```
        Dim choice As ConsoleKey
        Dim tempcommand As New Odbc.OdbcCommand("", conn)
        Dim tempreader As Odbc.OdbcDataReader
        Dim user, qual As String
        Do
            Console.WriteLine("Would You like to
1. View Qualifications
2. View or add Employee qualifications
0. Return to Main Menu")
            choice = Console.ReadKey.Key
            Console.WriteLine()
            If choice = ConsoleKey.D1 Or choice = ConsoleKey.NumPad1 Then
                tempcommand.CommandText = "SELECT * FROM Qualifications"
                tempreader = tempcommand.ExecuteReader
                While tempreader.Read
                    Console.WriteLine(tempreader("QualName") & "  |  " &
tempreader("QualDesc"))
                End While
                Console.WriteLine("Press 0 to return to main menu, or
press any other key to return to qualification menu")
                choice = Console.ReadKey.Key
                tempreader.Close()
                Console.Clear()
            ElseIf choice = ConsoleKey.D2 Or choice = ConsoleKey.NumPad2
Then
                tempcommand.CommandText = "SELECT `Legal name`, `Qualname`
FROM QualMapping, Users WHERE Users.username = qualmapping.username and
users.company = '" & company & "';"
                tempreader = tempcommand.ExecuteReader
                While tempreader.Read
                    Console.WriteLine(tempreader("Legal Name") & "  |  " &
tempreader("QualName"))
                End While
                tempreader.Close()
                Console.WriteLine("Press 1 to Add a qualification, press 0
to return to main menu, or press any other key to return to qualification
menu")
```

```
                choice = Console.ReadKey.Key
                Console.WriteLine()

                If choice = ConsoleKey.D1 Or choice = ConsoleKey.NumPad1
Then
                    Console.WriteLine("Which user would you like to add a
qualification to?")
                    user = SQLmaintenance(Console.ReadLine, 30)
                    Console.WriteLine("Which Qualification would you like
to add?")
                    qual = SQLmaintenance(Console.ReadLine, 30)
                    tempcommand.CommandText = "INSERT INTO qualmapping
Values((Select Username from users where `legal name` = '" & user & "' AND
company = '" & company & "'), '" & qual & "');"
                    tempcommand.ExecuteNonQuery()
                End If
                tempreader.Close()
                Console.WriteLine("Done

Press any key to return to qualification menu")
                Console.ReadKey()
                Console.Clear()
            End If
        Loop Until choice = ConsoleKey.D0 Or choice = ConsoleKey.NumPad0
    End Sub
```

5.13 Efficiency Algorithm

```
  Sub THE_BIG_ONE(ByVal company As String)
        Dim counter, tempcost, marker As Integer
        Dim employeeAccountedFor As Boolean
        Dim tempPos As String
        Dim unvisited, visited As New Dictionary(Of String, Node)
        Dim tempInputList As New List(Of String)
        Dim ToBeDone As New List(Of Job)
        Dim teamToDoIt As New List(Of Employee)
        Dim tempJob As Job = Nothing
```

```
        Dim tempEmployee As Employee = Nothing
        tempEmployee.Quals = New List(Of String)
        tempEmployee.NodePaths = New List(Of String)
        Dim UndoneTerm As New Odbc.OdbcCommand("SELECT `UCCs`.`UCC`,
`JobTypes`.`QualName`, `JobsDone`.`UJC` FROM `UCCs`, `JobsDone`,
`JobTypes`, `CompanyClient` WHERE `Uccs`.`ClientName` =
`JobsDone`.`ClientName` AND `JobsDone`.`JobComplete` = FALSE AND
`JobsDone`.`Username` IS NULL AND `JobsDone`.`JobID` = `JobTypes`.`JobID`
AND `UCCs`.`ClientName` = `CompanyClient`.`ClientName` and
`CompanyClient`.`Company` = '" & company & "' ORDER BY
`JobsDone`.`JobComplete` ASC;", conn)
        Dim UpdateTerm As New Odbc.OdbcCommand("", conn)
        Dim TeamAssembler As New Odbc.OdbcCommand("SELECT `UUC`,
`QualName` FROM `Users`, `TeamMapping`, `Regions`, `Clients`, `JobsDone`,
`Qualmapping`, `Teams` WHERE `JobsDone`.`ClientName` =
`Clients`.`ClientName` AND `Clients`.`Region Name` = `Regions`.`Region
Name` AND `Regions`.`TeamID` = `TeamMapping`.`TeamID` AND
`TeamMapping`.`Username` = `Users`.`Username` AND `JobsDone`.`JobComplete`
= 0 AND `Users`.`Username` = `Qualmapping`.`Username` AND `Users`.`User
Level` = 'Employee' AND `TeamMapping`.`TeamID` = `Teams`.`TeamID` AND
`Teams`.`Company` = '" & company & "' ORDER BY UUC ASC;", conn)
        Dim searcher As Odbc.OdbcDataReader
        Dim regionTerm As New Odbc.OdbcCommand("SELECT `Regions`.`Region
Name`, `Region Scale` FROM `Regions`, `Clients`, `JobsDone` WHERE
`Regions`.`Region Name` = `Clients`.`Region Name` and
`Clients`.`ClientName` = `JobsDone`.`ClientName` ORDER BY
`JobsDone`.`JobComplete` ASC;", conn)
        Dim regionLoad = regionTerm.ExecuteReader
        regionLoad.Read()
        Dim scale As Integer = regionLoad("Region Scale") * 40 + 100
        Dim regionName As String = regionLoad("Region Name")
        Dim adjacencymatrix(scale, scale) As Integer
        Using reader As BinaryReader =
                    New BinaryReader(File.Open((regionName & ".region"),
FileMode.Open))
            For i = 0 To scale - 1
                For j = 0 To scale - 1
```

```
                adjacencymatrix(i, j) = reader.ReadInt32
            Next
        Next
    End Using
    searcher = UndoneTerm.ExecuteReader
    While searcher.Read
        tempJob.Destination = searcher("UCC")
        tempJob.Qualification = searcher("QualName")
        tempJob.UJC = searcher("UJC")
        ToBeDone.Add(tempJob)
    End While
    searcher.Close()
    searcher = TeamAssembler.ExecuteReader
    While searcher.Read
        counter += 1

        employeeAccountedFor = False
        If teamToDoIt.Count > 0 Then
            For i = 0 To teamToDoIt.Count - 1
                If teamToDoIt(i).Home = searcher("UUC") Then
                    teamToDoIt(i).Quals.Add(searcher("QualName"))
                    employeeAccountedFor = True
                End If
            Next
        End If
        If Not employeeAccountedFor Then
            tempEmployee.Home = searcher("UUC")
            tempEmployee.PositionUpdate(searcher("UUC"))
            tempEmployee.NodePaths.Add(searcher("UUC"))
            tempEmployee.Quals.Add(searcher("QualName"))
            teamToDoIt.Add(tempEmployee)
        End If
        tempEmployee.Quals = New List(Of String)
        tempEmployee.NodePaths = New List(Of String)
    End While
    For i = 0 To ToBeDone.Count - 1
        tempcost = Integer.MaxValue
```

```
            For j = 0 To teamToDoIt.Count - 1
                If teamToDoIt(j).Quals.Contains(ToBeDone(i).Qualification)
Then
                    tempInputList.Clear()
                    tempInputList.Add(ToBeDone(i).Destination)
                    If tempcost > DijkstraChain(unvisited, visited, scale,
adjacencymatrix, 1, tempInputList, teamToDoIt(j).currentPosition, 2) Then
                        tempcost = DijkstraChain(unvisited, visited,
scale, adjacencymatrix, 1, tempInputList, teamToDoIt(j).currentPosition,
2)
                        tempPos = teamToDoIt(j).currentPosition
                    End If
                End If
            Next
            For k = 0 To teamToDoIt.Count - 1
                If teamToDoIt(k).currentPosition = tempPos Then
                    teamToDoIt(k).CostUpdate(tempcost)
                    teamToDoIt(k).PositionUpdate(tempPos)

teamToDoIt(k).NodePaths.AddRange(DijkstraChain(unvisited, visited, scale,
adjacencymatrix, 1, tempInputList, teamToDoIt(k).currentPosition, 1))
                    marker = k
                End If
            Next
            For l = 0 To teamToDoIt.Count - 1
                If teamToDoIt(l).currentPosition = tempPos Then
                    teamToDoIt(l).CostUpdate(tempcost)
                    teamToDoIt(l).PositionUpdate(tempPos)

teamToDoIt(l).NodePaths.Add(teamToDoIt(l).currentPosition)
                    l = marker
                End If
            Next
            tempInputList.Clear()
            tempInputList.Add(teamToDoIt(marker).Home)
            If teamToDoIt(marker).MinutesWorked + DijkstraChain(unvisited,
visited, scale, adjacencymatrix, 1, tempInputList,
```

```
teamToDoIt(marker).currentPosition, 2) > 480 Then
                teamToDoIt(marker).NodePaths.Add(DijkstraChain(unvisited,
visited, scale, adjacencymatrix, 1, tempInputList,
teamToDoIt(marker).currentPosition, 1))
                teamToDoIt(marker).PositionUpdate(teamToDoIt(marker).Home)
                teamToDoIt.RemoveAt(marker)
            End If
            UpdateTerm.CommandText = ("UPDATE `JobsDone` SET
`JobsDone`.`Username` = (SELECT `Username` FROM `Users` WHERE
`Users`.`UUC` =  '" & teamToDoIt(marker).Home & "') WHERE UJC = '" &
ToBeDone(i).UJC & "'; ")
            UpdateTerm.ExecuteNonQuery()

        Next
        Console.WriteLine("Done")
        Console.ReadKey()
    End Sub
```

5.14 Top Level Menus (Main, A & E)

```
    Sub Main()
        conn.Open()
        Dim signedIn As Integer = 0
        Dim choice As ConsoleKey
        Dim currentUser As New User()
        Do
            If currentUser.PowerLevel() = 0 Or signedIn = 0 Or signedIn =
2 Then
                Do

                    Console.Clear()
                    Console.WriteLine("1 to register
2 to register a company
3 to Sign In
0 to exit")
                    choice = Console.ReadKey.Key
                    Console.Clear()
```

```
                    If choice = ConsoleKey.D1 Or choice =
ConsoleKey.NumPad1 Then
                        AddUser("Employee")
                    ElseIf choice = ConsoleKey.D2 Or choice =
ConsoleKey.NumPad2 Then
                        NewCompany()
                    ElseIf choice = ConsoleKey.D3 Or choice =
ConsoleKey.NumPad3 Then
                        If signedIn = 0 Then
                            currentUser.SignIn()
                            If currentUser.returnUsername = "" Then
                                signedIn = 0
                            Else
                                signedIn = 1
                            End If
                        ElseIf signedIn = 1 Then
                            Console.WriteLine("You are already signed in")
                        ElseIf signedIn = 2 Then
                            Console.WriteLine("Sign in as " &
currentUser.Greet & "? (Y/N)")
                            choice = Console.ReadKey.Key
                            Console.Clear()

                            If choice = ConsoleKey.Y Then
                                currentUser.checkPassword()
                            ElseIf choice = ConsoleKey.N Then
                                currentUser.SignIn()
                                signedIn = 1
                            End If
                        End If

                    End If

                Loop Until choice = ConsoleKey.D0 Or choice =
ConsoleKey.NumPad0 Or signedIn = 1
```

```
            ElseIf currentUser.PowerLevel = 1 Or currentUser.PowerLevel =
2 And signedIn = 1 Then
                Do
                    If signedIn = 1 Then
                        Console.WriteLine("Welcome " & currentUser.Greet()
& ".")
                    End If
                    Console.WriteLine("
1 to view your personal data
2 to Log A Completed Job
3 to View Your Next Jobs
Backspace to Log Out")
                    choice = Console.ReadKey.Key
                    Console.Clear()

                    If choice = ConsoleKey.D1 Or choice =
ConsoleKey.NumPad1 Then
                        SearchUsers(currentUser.returnUsername)
                    ElseIf choice = ConsoleKey.D2 Or choice =
ConsoleKey.NumPad2 Then
                        JobLog(currentUser.returnUsername)
                    ElseIf choice = ConsoleKey.D3 Or choice =
ConsoleKey.NumPad3 Then
                        ShowJobs(currentUser.returnUsername)
                    ElseIf choice = ConsoleKey.Backspace Then
                        Console.WriteLine("You've been logged out.")
                        signedIn = 2
                        Console.ReadKey()
                    End If
                Loop Until choice = ConsoleKey.D0 Or choice =
ConsoleKey.NumPad0 Or signedIn = 2


            ElseIf currentUser.PowerLevel = 3 And signedIn = 1 Then
                If signedIn = 1 Then
                    Console.WriteLine("Welcome " & currentUser.Greet() &
".")
```

```
                End If
                Console.WriteLine("
1 to View or Edit Team Composition
2 to Add or Edit a Region
3 to add a Client
4 to Add a Job
5 to View Productivity Data
6 to Calculate working day
7 to Demonstrate Pathfinding
8 to Manage Qualifications
Backspace to Log Out")

                choice = Console.ReadKey.Key
                Console.Clear()

                If choice = ConsoleKey.D1 Or choice = ConsoleKey.NumPad1
Then
                    TeamComp(currentUser.GiveCompany)
                ElseIf choice = ConsoleKey.D2 Or choice =
ConsoleKey.NumPad2 Then
                    RegionCreator()
                ElseIf choice = ConsoleKey.D3 Or choice =
ConsoleKey.NumPad3 Then
                    AddClient(currentUser.GiveCompany)
                ElseIf choice = ConsoleKey.D4 Or choice =
ConsoleKey.NumPad4 Then
                    AddJob()
                ElseIf choice = ConsoleKey.D5 Or choice =
ConsoleKey.NumPad5 Then
                    productivitycheck(currentUser.GiveCompany)
                ElseIf choice = ConsoleKey.D6 Or choice =
ConsoleKey.NumPad6 Then
                    THE_BIG_ONE(currentUser.GiveCompany)
                ElseIf choice = ConsoleKey.D7 Or choice =
ConsoleKey.NumPad7 Then
                    DijkstraExample()
                ElseIf choice = ConsoleKey.D8 Or choice =
```

```
ConsoleKey.NumPad8 Then
                    QualificationHub(currentUser.GiveCompany)
                ElseIf choice = ConsoleKey.Backspace Then
                    Console.WriteLine("You've been logged out.")
                    signedIn = 2
                    Console.ReadKey()
                End If
            End If
        Loop Until choice = ConsoleKey.D0 Or choice = ConsoleKey.NumPad0
        conn.Close()

    End Sub

End Module
```

5.15 SQL Database Creation Statements

```
/*
SQLyog Community v11.5 (64 bit)
MySQL - 8.0.17 : Database - projdb
*********************************************************************
*/


/*!40101 SET NAMES utf8 */;

/*!40101 SET SQL_MODE=''*/;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO'
*/;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
CREATE DATABASE /*!32312 IF NOT EXISTS*/`projdb` /*!40100 DEFAULT
CHARACTER SET utf8 */ /*!80016 DEFAULT ENCRYPTION='N' */;

USE `projdb`;

/*Table structure for table `clients` */
```

```
DROP TABLE IF EXISTS `clients`;

CREATE TABLE `clients` (
  `ClientName` varchar(200) NOT NULL,
  `address` varchar(200) NOT NULL,
  `Region Name` varchar(50) NOT NULL,
  `creditnum` varchar(16) DEFAULT NULL,
  `ExpiryDate` varchar(5) DEFAULT NULL,
  `CVC` varchar(3) DEFAULT NULL,
  PRIMARY KEY (`ClientName`),
  KEY `Region Name` (`Region Name`),
  CONSTRAINT `clients_ibfk_1` FOREIGN KEY (`Region Name`) REFERENCES
`regions` (`Region Name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `clients` */

/*Table structure for table `companies` */

DROP TABLE IF EXISTS `companies`;

CREATE TABLE `companies` (
  `Company` varchar(100) NOT NULL,
  `CompanyID` varchar(5) NOT NULL,
  PRIMARY KEY (`Company`,`CompanyID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `companies` */

/*Table structure for table `companyclient` */

DROP TABLE IF EXISTS `companyclient`;

CREATE TABLE `companyclient` (
  `Company` varchar(100) NOT NULL,
  `ClientName` varchar(200) NOT NULL,
  KEY `Company` (`Company`),
  KEY `ClientName` (`ClientName`),
  CONSTRAINT `companyclient_ibfk_1` FOREIGN KEY (`Company`) REFERENCES
`companies` (`Company`),
  CONSTRAINT `companyclient_ibfk_2` FOREIGN KEY (`ClientName`)
REFERENCES `clients` (`ClientName`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `companyclient` */

/*Table structure for table `jobsdone` */
```

```
DROP TABLE IF EXISTS `jobsdone`;

CREATE TABLE `jobsdone` (
  `UJC` int(11) NOT NULL AUTO_INCREMENT,
  `Username` varchar(30) DEFAULT NULL,
  `ClientName` varchar(200) NOT NULL,
  `JobID` int(11) NOT NULL,
  `WorkDate` date DEFAULT NULL,
  `WorkStart` time DEFAULT NULL,
  `WorkEnd` time DEFAULT NULL,
  `JobComplete` tinyint(1) DEFAULT '0',
  PRIMARY KEY (`UJC`),
  KEY `Username` (`Username`),
  KEY `ClientName` (`ClientName`),
  KEY `JobID` (`JobID`),
  CONSTRAINT `jobsdone_ibfk_1` FOREIGN KEY (`Username`) REFERENCES
`users` (`Username`),
  CONSTRAINT `jobsdone_ibfk_2` FOREIGN KEY (`ClientName`) REFERENCES
`clients` (`ClientName`),
  CONSTRAINT `jobsdone_ibfk_3` FOREIGN KEY (`JobID`) REFERENCES
`jobtypes` (`JobID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `jobsdone` */

/*Table structure for table `jobtypes` */

DROP TABLE IF EXISTS `jobtypes`;

CREATE TABLE `jobtypes` (
  `JobID` int(11) NOT NULL AUTO_INCREMENT,
  `JobDesc` varchar(255) NOT NULL,
  `QualName` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`JobID`),
  KEY `QualName` (`QualName`),
  CONSTRAINT `jobtypes_ibfk_1` FOREIGN KEY (`QualName`) REFERENCES
`qualifications` (`QualName`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

/*Data for the table `jobtypes` */

insert  into `jobtypes`(`JobID`,`JobDesc`,`QualName`) values (1,'Fix an
A-Type Machine','Qualification 1'),(2,'Fix a B-Type
Machine','Qualification 2');

/*Table structure for table `qualifications` */
```

```
DROP TABLE IF EXISTS `qualifications`;

CREATE TABLE `qualifications` (
  `QualName` varchar(255) NOT NULL,
  `QualDesc` varchar(255) NOT NULL,
  PRIMARY KEY (`QualName`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `qualifications` */

insert  into `qualifications`(`QualName`,`QualDesc`) values
('Qualification 1','Can Fix A-Type Machines'),('Qualification 2','Can
Fix B-Type Machines');

/*Table structure for table `qualmapping` */

DROP TABLE IF EXISTS `qualmapping`;

CREATE TABLE `qualmapping` (
  `Username` varchar(30) NOT NULL,
  `QualName` varchar(255) NOT NULL,
  KEY `Username` (`Username`),
  KEY `QualName` (`QualName`),
  CONSTRAINT `qualmapping_ibfk_1` FOREIGN KEY (`Username`) REFERENCES
`users` (`Username`),
  CONSTRAINT `qualmapping_ibfk_2` FOREIGN KEY (`QualName`) REFERENCES
`qualifications` (`QualName`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `qualmapping` */

/*Table structure for table `regions` */

DROP TABLE IF EXISTS `regions`;

CREATE TABLE `regions` (
  `Region Name` varchar(50) NOT NULL,
  `Region Scale` int(11) NOT NULL,
  `TeamID` varchar(8) NOT NULL,
  PRIMARY KEY (`Region Name`),
  KEY `TeamID` (`TeamID`),
  CONSTRAINT `regions_ibfk_1` FOREIGN KEY (`TeamID`) REFERENCES `teams`
(`TeamID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `regions` */

/*Table structure for table `teammapping` */
```

```
DROP TABLE IF EXISTS `teammapping`;

CREATE TABLE `teammapping` (
  `TeamID` varchar(8) NOT NULL,
  `Username` varchar(30) NOT NULL,
  PRIMARY KEY (`Username`),
  KEY `Teamorder` (`TeamID`),
  CONSTRAINT `teammapping_ibfk_1` FOREIGN KEY (`Username`) REFERENCES
`users` (`Username`),
  CONSTRAINT `teammapping_ibfk_2` FOREIGN KEY (`TeamID`) REFERENCES
`teams` (`TeamID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `teammapping` */

/*Table structure for table `teams` */

DROP TABLE IF EXISTS `teams`;

CREATE TABLE `teams` (
  `Company` varchar(100) NOT NULL,
  `TeamID` varchar(30) NOT NULL,
  PRIMARY KEY (`TeamID`),
  KEY `Teamorder` (`TeamID`),
  KEY `Company` (`Company`),
  CONSTRAINT `teams_ibfk_1` FOREIGN KEY (`Company`) REFERENCES
`companies` (`Company`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `teams` */

/*Table structure for table `uccs` */

DROP TABLE IF EXISTS `uccs`;

CREATE TABLE `uccs` (
  `ClientName` varchar(200) NOT NULL,
  `UCC` varchar(2) NOT NULL,
  PRIMARY KEY (`UCC`,`ClientName`),
  KEY `ClientName` (`ClientName`),
  CONSTRAINT `uccs_ibfk_1` FOREIGN KEY (`ClientName`) REFERENCES
`clients` (`ClientName`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `uccs` */

/*Table structure for table `users` */
```

```
DROP TABLE IF EXISTS `users`;

CREATE TABLE `users` (
  `UUC` varchar(2) DEFAULT 'A',
  `Legal Name` varchar(100) NOT NULL,
  `Company` varchar(100) NOT NULL,
  `User Level` varchar(8) DEFAULT NULL,
  `Username` varchar(30) NOT NULL,
  `Passcode` varchar(45) NOT NULL,
  `Phone Num` varchar(11) NOT NULL,
  PRIMARY KEY (`Username`),
  KEY `Company` (`Company`),
  CONSTRAINT `users_ibfk_1` FOREIGN KEY (`Company`) REFERENCES
`companies` (`Company`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `users` */

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```