Nicky Temperley

Nicky Temperley

E6155906

EMA



Contents

[1 Draft Project Report 2](#_Toc13919693)

[1.1 Problem Description 2](#_Toc13919694)

[1.1.1 Background 2](#_Toc13919695)

[1.1.2 The Problem 2](#_Toc13919696)

[1.1.3 My Proposed Solution 3](#_Toc13919697)

[1.1.4 Alternatives to my proposal 3](#_Toc13919698)

[1.1.5 Resources Used 4](#_Toc13919699)

[1.2 Account of related literature 5](#_Toc13919700)

[1.3 Account of project work and its outcome 5](#_Toc13919701)

[1.3.1 Stage 1 – Data Collection and Project Definition 5](#_Toc13919702)

[1.3.2 Stage 2 - First Iteration 10](#_Toc13919703)

[1.3.3 Stage 3 - Second Iteration 17](#_Toc13919704)

[2 Review 23](#_Toc13919705)

[2.1 Review of the current stage of project work 23](#_Toc13919706)

[2.2 Review Project Management 23](#_Toc13919707)

[2.3 Assess risks to project completion 24](#_Toc13919708)

[2.4 Review of personal development 25](#_Toc13919709)

[3 References 26](#_Toc13919710)

[4 Appendices 27](#_Toc13919711)

[4.1 The initial enquiry email from the client 28](#_Toc13919712)

[4.2 Initial Meeting with the Client 30](#_Toc13919713)

[4.3 Proposed designs for the Java GUI 32](#_Toc13919714)

[4.4 Agreed designs for the Java GUI 37](#_Toc13919715)

[4.5 JSwing pages 39](#_Toc13919716)

[4.6 Java Code 45](#_Toc13919717)

[4.7 Mobile App Protype 59](#_Toc13919718)

[4.8 Code for the Mobile App 60](#_Toc13919719)

[4.9 Proposed Design of the Mobile App v2 66](#_Toc13919720)

[4.10 Screenshots of the Mobile App 67](#_Toc13919721)

[4.11 PHP Code 70](#_Toc13919722)

[4.12 Protype for the new GUI 75](#_Toc13919723)

# 1 Project Report

## Problem Description

### 1.1.1 Background

I have been working for Northumberland County Council for 9 years now, and within it there are over 6000 employees across dozens of departments, each providing a role to the Council and the Community. One of the departments in the council is the planning department, who receive planning applications.

Planning is a customer facing Service. They deal with Planning applications from members of the Public, land owners and Developers. The type of planning applications can vary from small scale such as a simple householder extension, through to large scale residential developments, new commercial developments, opencast sites, etc. Planning Officers are given 'delegated authority' by the Chief Planning Officer which means they have authority to determine planning applications. Most applications are determined by a Planning Officer however some less straightforward or contentious applications may be referred to a Planning committee.

Planning also write Policies through consultation to guide future development of the County. This sets the framework for all planning officers to adhere to for all future applications for developments of towns and villages within the County.

Conservation areas also fall under the remit of Planning. The Conservation officers may be involved in a planning application to give opinion and guidance to a planning officer where a planning application/development is in a Conservation area. Likewise advise is also provided to residents who may wish to make changes to a historic building such as a grade II listed building.

Planning also deal with a number of planning breaches. This may be that building works are carried out without planning permission or not in compliance with their approved planning approval. The Enforcement officers will also get involved in reports from members of the public of issues such as untidy land, non-approved advertising, etc.

### 1.1.2 The Problem

The planning department have about 100 users who work out on site. Fiona Charlton, the client, is looking for the development of a system that will allow her and other managers to record which officers are on site, which site they are at, the time that they arrived and the time they estimated they will leave. The officer will time to leave the site, and if they haven’t checked in when they leave or before this time an alert needs to be sent to their line manager. See [Appendix 4.1](#_4.1_The_initial) for a brief description in the client’s words.

Enrich the body of the report by taking quotes and using them here.

Currently there is no standardised reporting procedures for this to happen, and all teams use a different way of recording the information. There is a system already available called Guardian24, but at over £5 per user per month, this isn’t a viable solution for the whole workforce and is only used for the most at-risk officers. Without a standardised solution it could be the case that a site officer has an accident and isn’t able to contact the office, and no-one would be aware that there was a problem.

The officers will need to be able to submit the check in and out times from their mobile phone, and the system must be able to handle a manual input by office staff in the case they are unable to use their phone for whatever reason.

### 1.1.3 My Proposed Solution

To solve the problem, I am planning on designing and implementing a system to record all the information required. I will use a Java GUI to make calls the SQL to the database, with a mobile app created in Cordova using HTML, CSS and JavaScript to communicate via a REST web service to the database.

The Java system will record the managers details, the officer’s details, the estimated and actual check in and out times (with the actual times being submitted through the mobile app or manually entered), the site location and any notes required for the site. Officers, or admin staff on their behalf, will selected the officer’s name, manually enter a site location, then enter estimated check in and out times for when they arrive and leave site. The actual times will be entered later through the mobile app. The GUI will check for users that have checked in at a site and haven’t checked out by the time they have estimated; in this instance, an alert will show on every running instance of the system. This will be a prompt for a manager to contact them and take further action if needed or update the system on their behalf.

The mobile interface will be simple, with only a dropdown box to select the site, and buttons for check in and check out. The user will be authenticated by the Google account that they are logged into on the phone, which is linked to their staff domain account, and the site dropdown will only show sites that they have previously typed into the system.

### 1.1.4 Alternatives to my proposal

|  |  |
| --- | --- |
| **Alternative** | **Justification** |
| Carry on the way they are currently working | Not really an option as it could be a health and safety risk if site operatives have an accident on site and no-one notices that they haven’t checked in |
| Use the app Guardian24 for ever officer | Not feasible due to the cost, it would work out over £500 per month |
| Ask a third party to develop a solution | Another cost implication, high initial cost although there shouldn’t be a monthly cost apart from maybe an annual licensing fee |
| Develop a system with different technologies to the one proposed | This is achievable, and up until recently it was a strong possibility. I have decided against this as Java is my strongest language and I feel I could deliver the best results with a language I am comfortable, rather than trying to learn a new language to achieve a suitable outcome. |
| The development team at Northumberland County Council to develop a solution | The development team are unable to develop a solution as it is outside the scope of the services they provide to the council. Resources may become available later, but currently this isn’t an option |

### 1.1.5 Resources Used

|  |  |  |
| --- | --- | --- |
| **Resource** | **Why is it needed? / What does it do?** | **How to acquire** |
| Java v8u11 | Enables use of the Java programming language, and the ability to compile/run programs | Downloaded for free from the Oracle website. |
| Netbeans v8.2 | An IDE – for writing the code | Included with the above download. |
| DB Browser for SQLite | Software to allow me to create a database for my GUI to connect to, as well as seeing the state of the database at any given time. | Downloaded from the portable apps website. <https://portableapps.com/> |
| Java ODBC Drivers v3.27.2.1 | To allow my Java code to communicate with the SQLite database. | Downloaded from a BitBucket repository, <https://bitbucket.org/xerial/sqlite-jdbc/src/default/> |
| Data from users | This will be collected by sending the form in Appendix 5.5 to each of the users intending to use the system. For the purpose of this iteration, I have generated random information from [Mockaroo](https://mockaroo.com/) to build my database | Form has been sent to users to complete for next iteration. Random test data was created using the website GUI. Any users that have responded will be prompted by their manager to fill it in |
| Microsoft Forms | To create the form to collect the data. MS Forms is part of the Office 365 suite, which I have access to as both part of my Student Membership and my work account. | The simple interface is used to create the form, with several options available for each question type |
| UMLet | My chosen application for drawing and editing UML diagrams. | Downloaded from the UMLet website |
| Node,js | Used for installing various packages and plugins to work with cordova | Downloaded for free from <https://nodejs.org/en/>. I am using the LTS version 10.16.0 |
| Cordova | Allows me to design mobile apps using HTML, CSS and JavaScript | Using Node.js, type npm install -g corodova from a command prompt |
| Gradle | Pre-requisite of using Cordova, used to build the apps | Downloaded from <https://gradle.org/releases/> |
| Android SDK | Allows development on the Android platform | Downloaded the command linr tools only from <https://developer.android.com/studio> |
| Java Development Kit | Prerequisite of Cordova, required for Cordova to run | Downloaded from [https://www.oracle.com/technetwork/ java/javase/downloads/jdk8-downloads-2133151.html](https://www.oracle.com/technetwork/%20java/javase/downloads/jdk8-downloads-2133151.html) |
| XAMPP | Allows me to run a server on my local host, giving me the features a server is able to offer | Downloaded from [https://www.apachefriends.org/ download.html](https://www.apachefriends.org/%20download.html) |
| phpMyAdmin | Handles the operations on MySQL databases | Included with XAMPP installation |
| MySQL | Provides database functionality, required for storing and retrieving data | Included with XAMPP installation |

## Account of related literature

As mentioned in my project work, a major issue I was having was connecting my Java program to my SQLite database, something I haven’t had to do as part of my OU studies thus far. After searching on the internet, I managed to find an article on SQLite Tutorial (n.d.), which gave me not only the download link for the driver, but also step by step instructions on how to add the driver as a library to my java project and is also the basis for the “Test Connection” method in my “DataConnect.java” class. There were other sources available for this technique, but I found these the most straightforward and user friendly.

My next issue was that I have never used SQL and Java together, so I didn’t know how to do SQL calls within the Java code. There were ample resources available, but to start with I went to the Java Docs homepage, specifically relating to the java.sql package. These guides are the ultimate resource, everything you could need is there; but personally, I found all of the information a bit overwhelming. I didn’t spend too long looking at the docs, but I did refer to them about specific methods if I came across something in my other research that I didn’t understand. For the bulk of my database interaction, I used a website I’m going to refer to as “programming notes” titled “Java Database (JDBC) Programming by Examples with MySQL“ (Hock-Chuan, n.d.). I found the notes here relevant, easy to use, and they were helpful in creating my own classes and adapting to suit my needs.

After realising that connecting to my database from my Cordova app wasn’t going to be as straightforward as I first though, I took to the internet to look for ideas and inspiration. The first website that really caught my eye was called ‘PhoneGap PHP MySQL Example’ (Sundaravel, 2019). Although I couldn’t actually get his code example to work on my machine it confirmed that what I wanted to do was possible, it acted as a proof of concept in my mind. After further searching and trying several other worked example, I came across another tutorial titled “Connection with the backend” (Apache Cordova, n.d.) which I finally got working, and it was at this point I was able to adapt the examples to suit my requirements. The tutorial was very concise and I found it easy to follow and it covered all of the examples that I needed to create my PHP code.

## Account of project work and its outcome

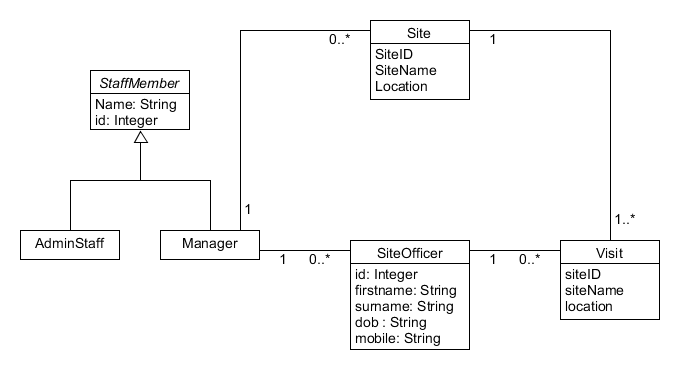
### 1.3.1 Stage 1 – Data Collection and Project Definition

#### 1.3.1.1 Initial Client Meeting and system

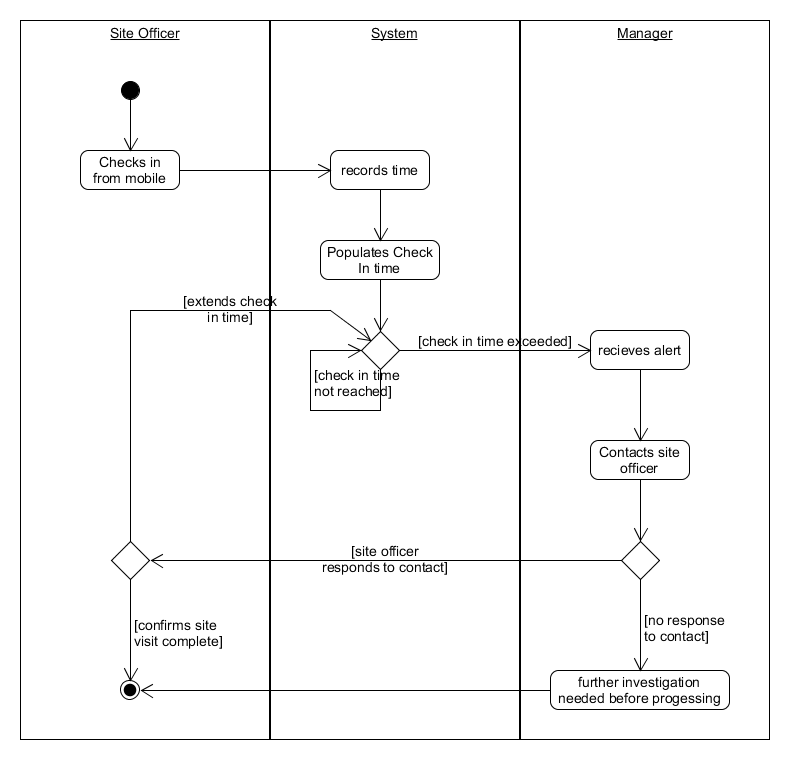
To start the project off I’ve had a meeting with Fiona Charlton on 07/03/19. It was my aim of this initial meeting to get some answers to some fundamental questions regarding the initial brief that was received in [Appendix 4.1](#_4.1_The_initial). The questions and her responses are in [Appendix 4.2](#_4.2_Initial_Meeting) .

Following this initial meeting I have managed to mock up some initial user interfaces (in Appendix 5) and I have planned a further meeting with Fiona on 24/03/2017 to get some feedback and talk her through the process of using the system. I still need some clarification on what the buttons along the bottom of the screen do (that I’ve labelled a-e) and if she needs this functionality included, and if its within the scope of this project. Once I’ve had the user interfaces confirmed I’ll be able to design them in Java using JSwing, then for the first iteration I’ll get the admin side of the system working.

Once the client is happy with the ideas so far, I’ll design the screens for adding and editing the site users and show the interactions between the different screens. At this moment, I imagine the classes needed and the associations between them will be:



After analysing the problem, my understanding is displayed in the following activity diagram:



At the end of the first TMA I had a list of questions that I wanted to ask my client. I have reproduced the list below with the response:

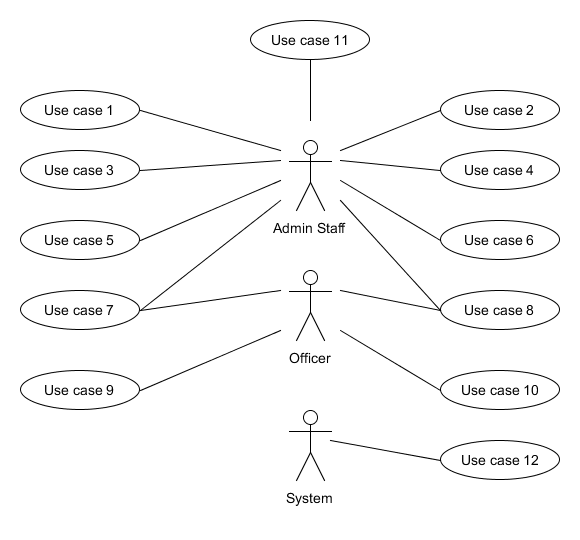
1. What is the role of the admin user? What do they do that the managers don’t and vice versa?  
   *There is no difference between the two users in regards to the function of the system, all staff will use the system, but admin staff wont be mentioned within the system. Site officers are admin staff that also perform site visits*
2. What happens in the event that the site user cant get a mobile signal, or use their phone for some reason.  
   *In this case, they tend to call or text their manager, or someone else in the office if they aren’t available. Even if a mobile signal cant be used for contact, they normally use a landline at whatever location they visit.*
3. What happens if a manager is not in the office when one of their team members go to site?  
   *If a manager is not available, due to annual leave etc, then another manager is appointed to that team in their absence. We’d quite like a way to move people teams in the system you’re developing to handle this if that’s at all possible.*
4. How would you expect to be able to manually edit the times of the site officers?  
   *We’d only need to edit the “actual” check in and out times, but I’d expect them to be available within the system that’s used on a desktop machine.*
5. If an estimated check out time is not specified, would it be helpful to use a default value?  
   *No, I’d like the officers to be forced to enter an estimated time please.*
6. What information do you store about the sites that the officers visit?  
   *We need to store their name, mobile number, car registration, next of kin name, and a contact number for the next of kin.*Is there any unique information you store about the users?  
   *Not at the moment, but we have the staff ID number that could be used as a unique identifier if that’s helpful?*
7. Is a site deleted once the officers have finished all their visits? Is it made inactive?  
   *The site details don’t need to be kept. We visit thousands of sites per year, so its not feasible to store all that data. We’re just using this system to view officers current visits, once they check back in, there no need to store the data about that visit.*

#### 1.3.1.2 The project so far

After reviewing all the information, and speaking with the client, I have developed to following use cases:

|  |  |  |
| --- | --- | --- |
| Reference | Name | Description |
| UC1 | Add a manager | Add a new manager to the system |
| UC2 | Edit a manager | Edit a currently stored manager from the system |
| UC3 | Delete a manager | Delete a currently stored manager from the system |
| UC4 | Add an officer | Add a new officer to the system |
| UC5 | Edit an officer | Edit a currently stored officer from the system |
| UC6 | Delete an officer | Delete a currently stored officer from the system |
| UC7 | Set estimated times | An officer sets the estimated visit times using the GUI |
| UC8 | View more details | Anyone can click “more details” on the GUI and see more details about the selected officer |
| UC9 | Check in from app | An officer uses the mobile app to check in when they get to site |
| UC10 | Check out from app | An officer uses the mobile app to check out when they get to site |
| UC11 | Manually edit actual times | If a user rings the office rather than use the app, the actual times need manually updated |
| UC12 | Alert the manager | If an Officer hasn’t checked in by the time the estimated check out time arrives, an alert will present on the screen |

From the use case, I have created a use case diagram denoting the three actors; Admin Staff, Officers and the System itself the show the responsibilities concerned:



### 1.3.2 Stage 2 - First Iteration

#### 1.3.2.1 The Protype Designs

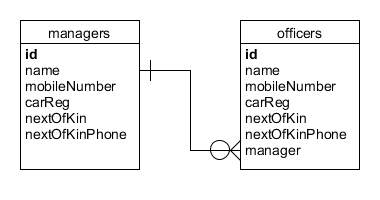
For the first iteration I have chosen to implement part of the GUI, specifically the part that allows new users to be added, edited or deleted; this will deal with UC1-UC6. I have initially developed the GUI as per [Appendix 4.3](#_4.3_Proposed_designs) and she has replied with the following comments:

* Could we get a button on the main screen, so admin staff can easily view all other details about the selected officer without having to go through the system looking for them
* The date field is irrelevant as we are only concerned with visits that are ongoing or happing on that day
* Could we have separate times on the page so we can see both the expected time on site and the actual time, that way we can see if someone has actually arrived on the site. This could be used in case someone has had an accident on the way to site. i.e. if they haven’t checked in around the time they were expecting, we can contact them to make sure everything is ok

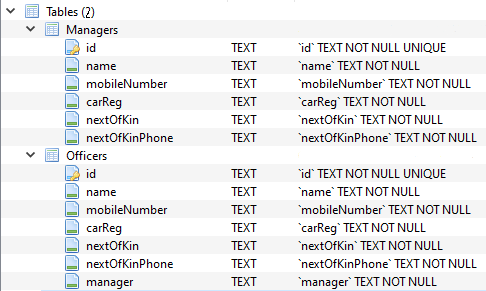
In response to the above points I have created the prototypes in [Appendix 4.4](#_Appendix_5.7_–) which include an amended front screen, and a new screen for the “more details” button. After showing these new designs to my client, she’s agreed to all the changes and has given the approval to develop them.

#### 1.3.2.2 SQLite Database

Since the first iteration is only concerned with the data concerning the site officers and the managers, my ERD is displayed below to reflect this:

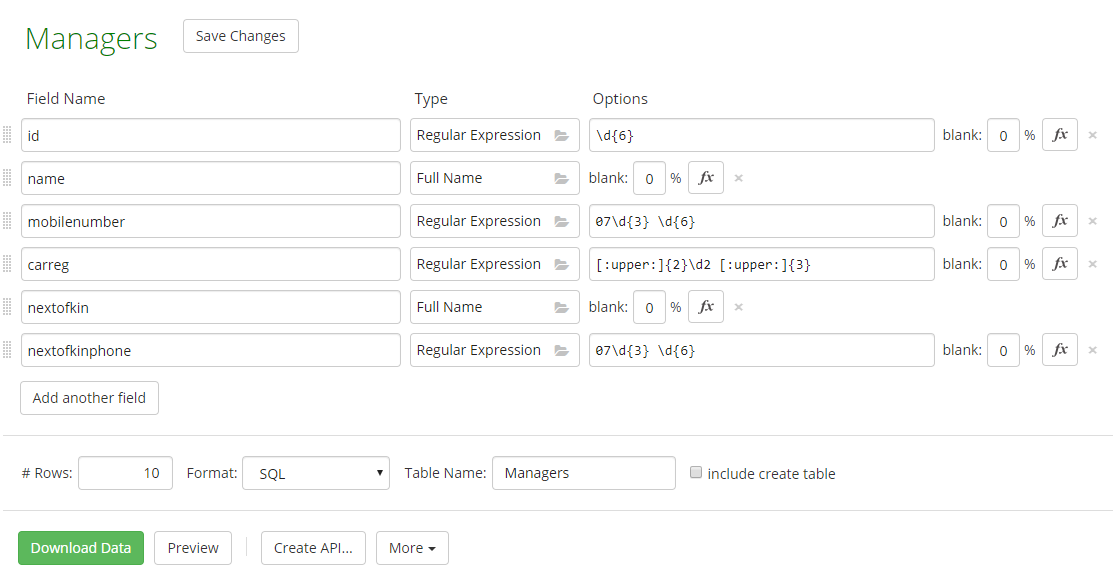


I have then used DB Browser for SQLite to create the database. This was done using the GUI, there is a create database button, from then it’s a simple as creating the tables using the buttons shown. My final table when created looks like this:



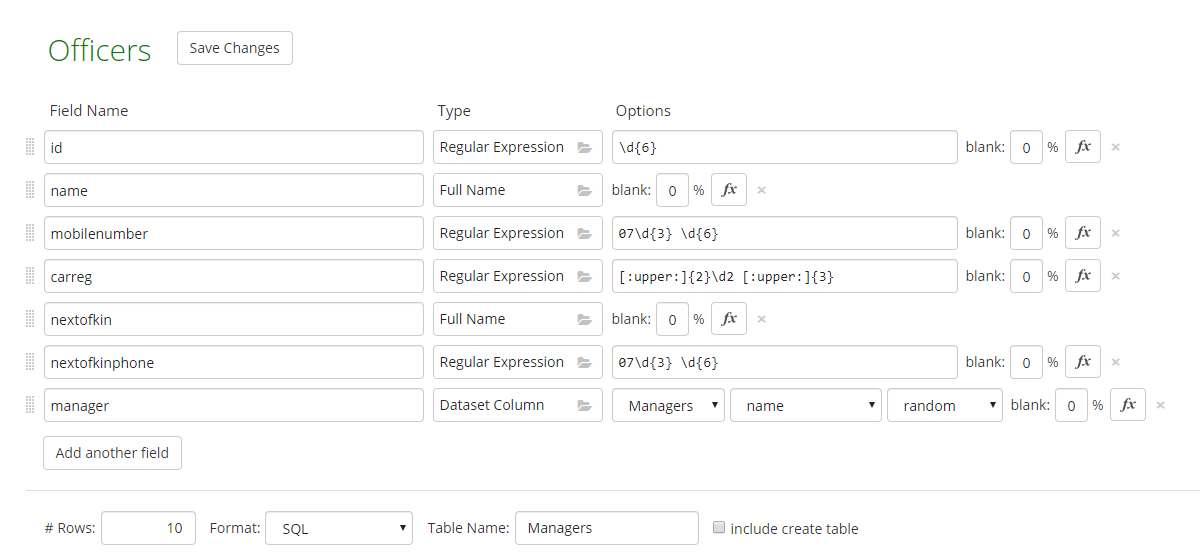
To populate the tables with test data I have used a website called Mockaroo. This website has allowed me to specify the types of data I want to create, and then provides the SQL for me to write to insert the data into the tables. The parameters I have used on the site to create the data are:

**Managers**



This gives me a 6 digit ID number (which matches our staff ID’s), a random name, a mobile number in the correct format, a car reg in the correct (post 2011) format, a random name for the next of kin and a phone number for them in the correct format.

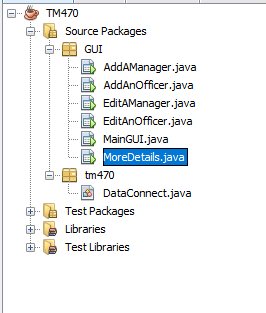
**Officers**



This gives me the same data as above, but with addition of a manager field, which randomly selects one of the previously created managers.

#### 1.3.2.3 Creating the GUI

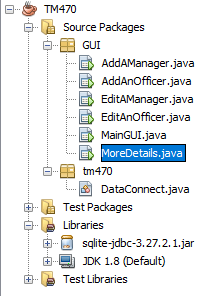
To create the GUI I have decided to use the JSwing elements of Java, as taught in M256. This will give me a drag-and-drop interface, which should allow me to fine tune the design to how its been agreed. I started by placing the elements on the page to recreate the main page and I’ll concentrate on the coding behind the objects later. I have created one main page, then an additional page to correspond to each of the buttons along the bottom of the page. All of these pages represent a Java Class, so that’s a total of 6 classes, one for each page. I have also created another class called ‘DataConnect.java’ which handles the connections to the database to make a total of 7 classes.



All the pages within the GUI package are available within [appendix](#_Appendix_5.10_–) 4.5. I have used the appendix to detail how the pages work together, and any other relevant notes are below the pictures.

#### 1.3.2.4 Problems so far

The only main problem I’ve faced up until this point was that I didn’t know how to connect my java pages to my SQLite database. Although my studies have taught me Java and SQL individually, there has never been any cross over between the two. After a bit of research on the internet, I found a tutorial (SQLiteTutorial.com, n.d.), and after downloading the SQLite JDBC driver I managed to add it to my project and this will allow me to connect my Java GUI with my SQLite database. My directory now looks like this:



#### 1.3.2.5 The code behind the pages

All of the classes I have created need code added to provide the required functionality. The below table lists the methods I will need are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Method** | **Activated by** | **Description** |
| MainGUI | btnTestConnectionMouseClicked() | Clicking on the ‘Test’ Connection’ button | Tests the connection to the database for testing purposes. To be removed in a later iteration |
| MainGUI | btnExitMouseClicked() | Clicking on the ‘Exit’ button | Displays a confirmation box asking if they want to close the system, if they click yes it closes, if they click no it returns to MainGUI |
| MainGUI | btnAddManagerMouseClicked() | Clicking on the ‘Add A Manager’ button | Displays the AddAManager screen |
| MainGUI | btnAddOfficerMouseClicked() | Clicking on the ‘Add An Officer’ button | Displays the EditAManager screen |
| MainGUI | btnEditManagerMouseClicked() | Clicking on the ‘Edit A Manager’ button | Displays the AddAnOfficer screen |
| MainGUI | btnEditOfficerMouseClicked() | Clicking on the ‘Edit An Officer Button’ button | Displays the EditAnOfficer screen |
| MainGUI | cmboOfficerMouseClicked() | Clicking on the ‘Officer’ dropdown menu | This will do an SQL call in the format “SELECT name FROM Officers” to populate the list of officers available |
| MainGUI | btnMoreDetailsMouseClicked() | Clicking on the ‘More details’ button | Displays the MoreDetails window |
| MainGUI | cmboOfficerActionPerformed() | Selecting a new officer from the officer drop down menu | Performs an SQL call in the format “SELECT \* FROM Officers WHERE name = ‘xxxx’”, where xxxx is the officer name selected in the list. |
| AddAManager | btnReturnMouseClicked() | Clicking on the ‘Return’ button | Returns to MainGUI |
| AddAManager | btnSaveMouseClicked() | Clicking on the ‘Save’ button | Performs an SQL call in the form “INSERT INTO Officers(id, name, mobileNumber, carReg, nextOfKin, nextOfKinPhone) values (?,?,?,?,?,?)", where the question marks refer to the values the user has entered on the screen |
| AddAnOfficer | btnReturnMouseClicked() | Clicking on the ‘Return’ button | Returns to MainGUI |
| AddAnOfficer | btnSaveMouseClicked() | Clicking on the ‘Save’ button | Performs an SQL call in the form “INSERT INTO Officers(id, name, mobileNumber, carReg, nextOfKin, nextOfKinPhone, manager) values (?,?,?,?,?,?,?)", where the question marks refer to the values the user has entered on the screen |
| EditAManager | btnReturnMouseClicked() | Clicking on the ‘Return’ button | Returns to MainGUI |
| EditAManager | btnDeleteMouseClicked() | Clicking on the ‘Delete’ button | Performs an SQL call in the form "DELETE FROM Managers WHERE id = \" + managerId + \" where the managerId is the manager’s ID that the user has selected in the form |
| EditAManager | btnUpdateMouseClicked() | Clicking on the ‘UPDATE button | Performs an SQL call in the form "UPDATE Managers SET name = xxxx. mobileNumber = xxxx, carReg = xxxx, nextOfKin = xxxx, nextOfKinPhone = xxxx WHERE id = \" + managerId + \" where the managerId is the manager’s ID that the user has selected in the form and xxxx are the details the user has entered on the form. |
| EditAnOfficer | btnReturnMouseClicked() | Clicking on the ‘Return’ button | Returns to MainGUI |
| EditAnOfficer | btnDeleteMouseClicked() | Clicking on the ‘Delete’ button | Performs an SQL call in the form "DELETE FROM Officers WHERE id = \" + id + \" where the id is the officers ID that the user has selected in the form |
| EditAnOfficer | btnUpdateMouseClicked() | Clicking on the ‘UPDATE button | Performs an SQL call in the form "UPDATE Officers SET name = xxxx. mobileNumber = xxxx, carReg = xxxx, nextOfKin = xxxx, nextOfKinPhone = xxxx, manager = xxxx WHERE id = \" + id + \" where the id is the officers ID that the user has selected in the form and xxxx are the details the user has entered on the form. |
| MoreDetails | btnReturnMouseClicked() | Clicking on the ‘Return’ button | Returns to MainGUI |

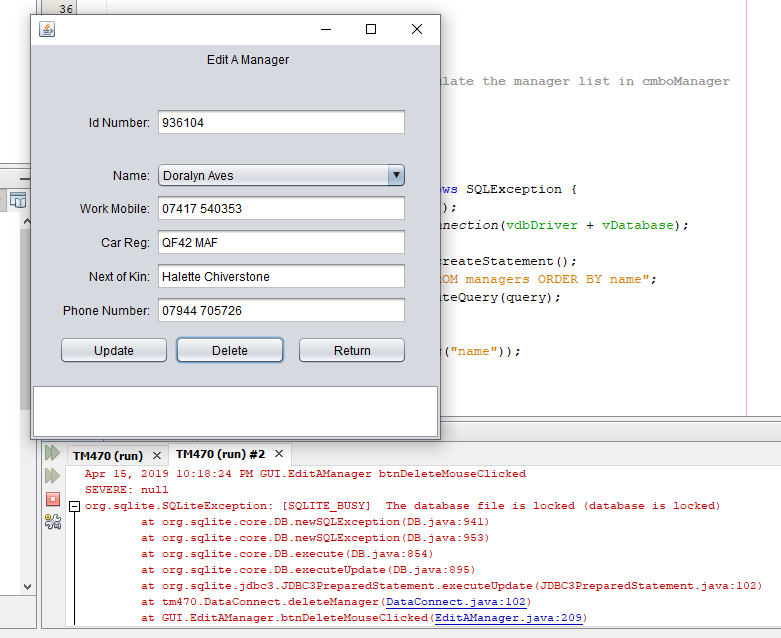
To help with the above methods, I will also create helper methods in the Data Connect class. These, and the full versions of my code can be found in [Appendix 4.6.](#_4.6_Java_Code)

#### 1.3.2.6 UAT – User acceptance testing

To test my GUI I will ask my client Fiona to run the following tests:

|  |  |  |  |
| --- | --- | --- | --- |
| **Reference** | **Name** | **Description** | **Outcome** |
| Test1 | testAddNewManager | To test the addition of a new manager, valid data entered by the user, expected success message | Success message generated, all fields cleared for next input |
| Test2 | testAddNewManager | To test the addition of a new manager, invalid data entered by the user, error message expected | Error message generated, as expected |
| Test3 | testAddNewOfficer | To test the addition of a new officer, valid data entered by the user, expected success message | Success message generated, all fields cleared for next input |
| Test4 | testAddNewOfficer | To test the addition of a new manager, invalid data entered by the user, error message expected | Error message generated, as expected |
| Test5 | testEditManager | To test deleting an existing manager, current manager selected, expected success message | Error message, database file has locked |
| Test6 | testEditManager | To test editing an existing manager, valid data entered by the user, expected success message | Not implemented yet due to error on previous method |
| Test7 | testEditManager | To test editing an existing manager, invalid data entered by the user, expected success message | Not implemented yet due to error on previous method |
| Test8 | testEditOfficer | Not implemented yet due to error on previous method | Not implemented yet due to error on previous method |
| Test9 | testEditOfficer | Not implemented yet due to error on previous method | Not implemented yet due to error on previous method |
| Test10 | testMoreDetails | After selecting a valid officer on MainGUI, all of their details should be available when the user clicks the ‘More Details’ button | Success. As expected, all the details of the currently selected officer are available when clicking on ‘More Details’ |

The tests above in Red failed, and I will be fixing this first as part of the next iteration. The error message showing on Test5 (with a random manager) is:



### 1.3.3 Stage 3 - Second Iteration

#### 1.3.3.1 Interview with the client

Before I start developing the mobile app for the officers to use on site, I arranged another meeting with Fiona to discuss the scope and usability of the project. Below are the questions I’ve asked with her responses:

Q. Can you clarify exactly what the difference is between an admin role in the office and a manager’s role. What can a manager do that an admin person can’t?

A. There is no difference, but the managers only should receive the alert.

Q. What would happen is the site officers listed manager was absent, or just away from their desk when an alert went off?

A. To avoid this, would we be able to get the screen to change to alert everyone that someone hasn’t checked in? Maybe we could install a tv in the office to display this screen only, so we could use it as a dashboard?

Q. Would the whole system not work better if that the case, designed as a dashboard? You could use it to display only the active visits, then you haven’t got to worry about the screen being cluttered with information that isn’t relevant for the purpose of this system?

A. Yes, that would be a much better idea, a dashboard only showing active visits.

Q. Would you need to interact with the new “dashboard” at all?

A. Yes, we would still need to be able to manually check in officers, as well as the normal admin functions like adding new officers etc.

Q. And what about the site information. Does this need to be retained?

A. It would be helpful if it could, although it isn’t really necessary. Maybe the site list could be controlled by the admin staff, so the officers don’t need to worry about it, they just select it when they go to site

Q. And lastly, how long does the normal site visit take? Do you need users to be able to check in for different times, or is there a minimum time that they need to contact the office?

A. Well, the policy at the moment is that they contact the office hourly when they are out on site. Maybe when they check in it could be set as 1 hour, then if they want to reset this they can just check in again to gain another hour.

From the discussion, I have noted the following points and will take them into consideration both when designed the app, but also in the third iteration when I finish the GUI:

* When a user checks in on site, they get one hour to either check out or check in again, restarting that hour.
* All sites can either be entered as new by the officers on site, or if returning to an existing site they can select that from a list.
* Sites will be deleted by office staff when officers hand them the file to say that inspections have finished at that site.
* The GUI will be redesigned as a dashboard, so I will be removing the check-out time fields and expected time fields and replacing them with a single “last checked in” box.
* The GUI will only show visits that are taking place at that time, and an alert will show up on screen if someone hasn’t checked in within the hour.
* Although site information can be deleted, a log of all check-ins and check-outs will be retained

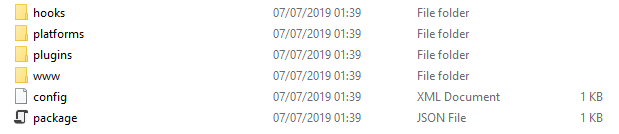
#### 1.3.3.2 Setting up my Cordova project

To develop my mobile phone app, I have used Apace Cordova. As I have previously used this in other subjects, I had it installed on my machine already, so I won’t go into the installation details again here. To create the project, I simply open a command prompt and type:

cordova create tm470app

which creates the project is the current working directory. I can then navigate to this file with cd tm470app

which puts me inside the project file. The default file structure should look like this when viewed with windows file explorer:

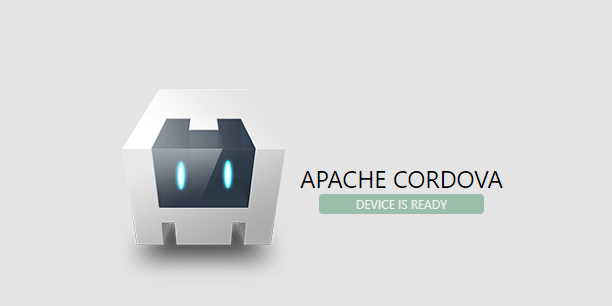


The www folder is containing the HTML, CSS and JavaScript files that I’ll use to develop the app. To run the file, I need to add ‘platforms’ that I wish it to run on, so I need to go back to the command line, navigate to the project directory and type:

cordova platform add android adds android as a platform – to run the app

cordova platform add browser adds the browser as a platform – used for testing

Running the app as it is shows



to confirm the app is working. Now I can start editing the www files, customising my app towards the agreed prototype, originally shown in [Appendix 4.7](#_4.7_Mobile_App) and now updated and agreed as per [Appendix 4.9](#_4.9_Proposed_Design).

#### 1.3.3.3 Writing the code for the app

I’ll begin with the HTML file, to get the content on the screen that I want to show. I have deleted all the autogenerated code to start with a blank canvas; from there I know the screen has roughly got to take on the following form:

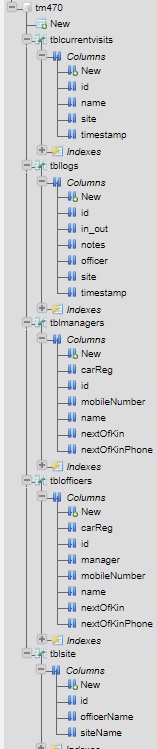
|  |
| --- |
| Logo |
| Select Username |
| Select Site |
| Enter Notes |
| Check In |
| Check Out |
| Status Update |

At this point it is worth noting that because I don’t have a mobile phone linked to a work account, I am unable to find the username by the account that’s logged into the device. To get around this I have decided to add a username dropdown box; the intention is that this will be removed in future iterations.

Once the HTML had been completed it was my intention to use the JavaScript to connect to my SQLite database that I have on my pc, but until this point I hadn’t thought about how to do this. After doing a google search for “connect to SQLite Database with Cordova” I found two plugins that would appear to provide the functionality I needed, “cordova-plugin-sqlite-2” and “cordova-sqlite-storage”. After following multiple tutorials on how to get these to work, as well as one or two less documented plugins, I realised that this wouldn’t work how I had previously imagined; all of the plugins required you to copy the database to the device and work with that, rather than use the data stored in a central location which is what I needed.

After a bit more research I found out that JavaScript, being a frontend language, can’t communicate with a backend service like a SQLite Database. From this point forward, I installed a program called XAMPP, which allowed the local host on my machine to act as a server. It is bundled with phpMyAdmin and MySQL which I can use to connect the frontend to the backend. As long as the mobile phone and my machine were on the same local network, they could communicate.

#### 1.3.3.4 Using XAMPP – phpMyAdmin and MySQL

The first task I had to do was to import my database into MySQL running on the local host. Luckily, the package I was originally using (DB Browser) had an export facility so it was as simple as exporting the data from one and importing it into MySQL. Once I had the original data, I also create tables for the current site visits, the active site lists, and another to hold all of the logs. My database structure was now as follows:

tblcurrentvisits will hold a list of all the officers that are on site right now. When they check in with the app their details will be added to this table, and when they check out they will be removed.

tbllogs will store all activity (every check in and check out) that takes place and be used as a reference of a visit taking place if needed.

tblmanagers (as previously described) will hold all the information for all of the team leaders.

tblofficers (as previouslt described) will hold all the information for all the site officers.

tblsite will hold a list of all of the active sites. They can be added to from the mobile app or deleted from the GUI

#### 1.3.3.5 Writing the PHP to Link the App with the MySQL Database

This part of the process took by far the longest; even though the PHP I’ve written is quite simplistic, I’ve never really used it before, so I had to pick everything up rom new. Luckily there is an abundance of resources available online that I was able to use a combination of to create the functionality I needed. Once XAMPP is running, it emulates a server on my local host. Within the installation folder there is a folder called “htdocs”; this acts as a sort of a file server, so any PHP files I place in there can be accessed from my app. The files I’ve created are:

|  |  |
| --- | --- |
| Filename | Function |
| db.php | This file creates a connection to the database stored under the variable $con. This file will be referenced at the start of all the others, so we can use the same connection |
| getNames.php | Used to retrieve the list of usernames from the officers table, activated when the user clicks the drop-down list in the app. Executes the SQL “select name from `tblofficers` ORDER BY name ASC” |
| getSites.php | Used to retrieve the list of available sites from the site table, activated when the user clicks the drop-down list in the app. Executes the SQL “select siteName from `tblsite` ORDER BY siteName ASC” |
| checkIn.php | This file has 3 functions:   1. If the site is new i.e. it has been typed by the user, add it to the tblsites 2. Add all the data to tbllogs to record the action 3. Add the info into tblcurrentvisits to note the officer is currently on site |
| checkout.php | This file has two fucntions:   1. Add all the data to tbllogs to record the action 2. Remove the corresponding entry in tblcurrentvisits to show the officer is not currently on site |

For the listing of all code, please see [Appendix 4.11](#_4.11_PHP_Code)

After all the PHP code had been written, I was able to use jQuery, which is a JavaScript framework, to run the PHP files running on the local host. Before putting this app into production, the HTML in index.php would need to be edited. Currently it’s pointing to the IP address of my desktop machine, a new server would have to be created running phpMyAdmin and MySQL, and that IP address of that server would need to replace my IP. This would allow anyone with an internet connection running the app to connect to the server. My completed code for the mobile app can be seen in [Appendix 4.8.](#_4.8_Code_for) For screenshots of the completed app, please refer to [Appendix 4.10](#_4.10_Screenshots_of).

#### 1.3.4 Next Steps

For the next iteration I need to:

1. Redesign my GUI to suit the new requirements, as agreed with Fiona and displayed in [Appendix 4.12](#_4.12_Protype_for).
2. Change the JDBC from SQLite to MySQL
3. Implement functionality to:
   1. Create a new entry in the GUI for every entry in tblcurrentvisits.
   2. Delete Site
   3. Produce an alert if a site officer misses a check in
   4. Refresh the page every x seconds (still to be agreed with Fiona)
4. Produce a user guide
5. Consider handling the transaction in PHP rather than Java?
6. Investigate technical considerations of having an external app communicating with an internal server (the MySQL server). How do we achieve this with the Firewall and DMZ?

### 1.3.4 Stage 4 – Third Iteration

#### 1.3.4.1 Rethinking the problem

The last time I worked on the GUI, some of the tests failed due to the database locking. I decided rather than try to fix the code that I had already written, it would be easier to refactor the code to use a single class for the connection, then to do the processing of the SQL and close the connection in each method. To achieve this, I’ve created the following method (see [Appendix 4.13.9](#_4.13.9_DBConnect.java)):

1. **public** Connection openConnection() {
2. **try** {
3. **final** String DB\_HOST = "jdbc:mysql://localhost:3306/tm470";
4. **final** String DB\_USER = "root";
5. **final** String DB\_PASSWD = "";
7. Connection conn = DriverManager.getConnection(
8. DB\_HOST, DB\_USER, DB\_PASSWD);
9. **return** conn;
11. } **catch** (SQLException e) {
12. System.out.println(e.getMessage());
13. }
15. **return** **null**;
16. }

Now to use this within a method in the GUI, I can write something like:

1. DBConnect conn =  **new** DBConnect();
2. Connection openConn = conn.openConnection();
4. <<process SQL statement here >>
6. openConn.close();

Using this method means that there won’t be an instance where one method runs while another method is accessing the SQL database, therefore avoid the database locking error I had encountered previously. Running the same test as [section 1.3.2.6](#_1.3.2.6_UAT_–) all now passed as expected.

#### 1.3.4.2 Writing the code for the GUI

Luckily, most of the code I had written for the desktop App v1 could be reused in this iteration. I’ve had to redesign the main page do that the rows of the database could be added dynamically, but all the designs for adding and editing users were reused, although I did have to amend the code to include the new DBConnect class. After this I’ve also create a class called SiteVisit which collects all of the data returned from the database and stores the information relating to an individual site visit, and this class is in turn used to add that information to the main interface.

I’ve removed the test connection button as used in the first iteration as this will not be used in the final product, it was there purely to aid my development. I’ve also added an “Officers Details” button which, when a row in the table is select and the button is clicked will return all the details held about the person on site; as was initially stated by Fiona “I was thinking it could be made available so if there was an accident, or maybe just someone hasn’t checked back in in time then we’d have the information to hand without having to search through files to find it.”.

#### 1.3.4.3 Problems with my approach

Up until this point, I had a pretty clear direction how I was going to proceed. I would create a form showing the site officers name, the site they were on, and the time they had checked into site. I would then query the database and for each row I would create an instance of this form and add it to my form, and name variables according to the row they came from, something like this:



But when I was running the code, I kept encountering errors. I did a lot of research online and asked a few colleagues and as it turns out, Java doesn’t support dynamically named variables. Since all of the variables are declared and compiled at runtime, the idea of dynamic naming isn’t available. At this point I didn’t know what to do, and I decided to cancel my whole java program and rewrite the system using HTML, CSS and JavaScript; I actually had quite a lot of it done when I came to my senses and realised that this functionality must be available somehow, even without the dynamic variable names. After a couple more days research, I found the jTables elements that is made for tabulated data, the type returned from my database. Now the code couldn’t have been easier, I simply wrote a method:

1. ArrayList<SiteVisit> list = siteVisits();
2. DefaultTableModel model = (DefaultTableModel) tblSiteVisits.getModel();
3. model.setRowCount(0); //to clear the current database
4. Object[] row = **new** Object[3]; //create a new row
5. **for** (**int** i = 0; i < list.size(); i++) {
6. row[0] = list.get(i).getName();
7. row[1] = list.get(i).getSite();
8. row[2] = list.get(i).getTimestamp();
9. model.addRow(row);

There’s a little bit of code on the end to show an alert, the full code can be viewed in [Appendix 4.13.1](#_4.13.1_MainGUI.java)

#### 1.3.4.4 Completing the GUI

After I had worked out how to get the data into the table, the rest was pretty straight forward; the only things still to complete was the timer so the table would refresh periodically, and the alert if someone hadn’t checked in on time.

For the timer, all I had to add were the following lines of code inhte the instance declaration:

1. timer = **new** Timer();
2. timer.schedule(**new** RemindTask(),
3. 0, //initial delay
4. 20 \* 1000);  //subsequent rate

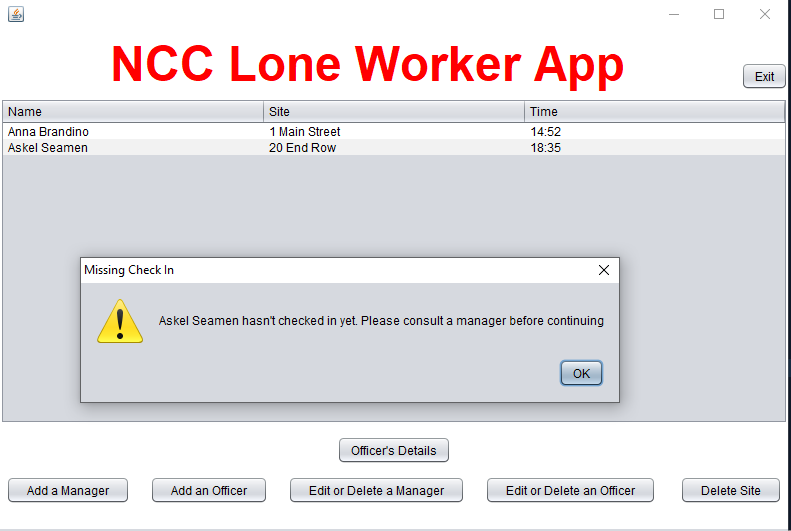
and every time an instance of the display is created, a new timer is created. In this example here I have set it to start 0 milliseconds from when the screen loads, and every 20 x 1000 milliseconds (20 seconds) after that. This time would need to be confirmed with Fiona before going live, but the subsequent rate is the value to change to suit requirements.

And lastly, the alert; I have used the timestamp variable from the database and created another variable *now* to hold the time that the function was called. All I had to do I add an hour to the time they checked into the site, and if that time was after *now* to display an alert. I have a few problem getting the logic correct here, but after a few trial and error attempts, it is now working.

A screen shot of the completed GUI running as normal:



And another showing the alert that is generated when someone hasn’t checked in on time:



#### 1.3.4.5 Putting the system into practice

Now that the system is complete, I’ve set out guidelines to adapt my system if it is to be put into production:

1. A new server will need to be created within the Northumberland County Council network situated within the DMZ. This server will need to:
   1. Allow access through the internal firewall to the SQL server and for suers of the GUI
   2. Allow access through the exterenal firewall for users of the Mobile App.
   3. Rules created in the firewall to allow communication between the app, GUI and SQL server
2. A new database will need to be created within the live environment.
3. In the DBConnect.java class, the variables to hold the database hose, username and password will need to be changed to reflect the live database.
4. The time that the database refreshes may need to be altered, this can be found .
5. Some additional development may be desired, please see the potential future iterations in section 2.1.

# 2 Review

## Review of the current stage of project work

At this point in the project, although I’m pleased with my progress and the product I have created, I’m a little bit disappointed in the direction the project has taken. Originally, I thought I had quite a good idea in my head of what the finished product would be, what functionality it would have, and how it would work. Since reaching this point my lack of experience has become apparent and I know that the things I had predicted were a little bit out of my reach. That being said, I am very pleased with how I’ve handled situations where things haven’t worked. The problem with not being able to connect to the database from my app using JavaScript was a major setback for me, it meant I had to find a workaround, something that I thought would be straightforward ending up involving more software and languages than I had anticipated. I still achieved my goal though, I did what I had to do to achieve the desired outcome, and the result to the user is the same as I had originally intended.

The main problem I think I’ve had so far with this project, a problem that I think will hurt my progress again in the next iteration, is the Java GUI. I don’t have much experience with designing GUI’s and I think I’m going to have a problem when it comes to the placement of items on the screen. I think I’m going to have to look for resources which will help me with positioning to make sure it works regardless of the screen resolution. In hindsight, again this is where my lack of experience shows, I should’ve developed a web-based html system. This would have limited the technologies I would have needed to use, and it is something I am way more comfortable with. Making this mistake though means I have learnt something else; your first idea isn’t always the right one, I wish I’d put more thought into the options available before just jumping in with a java-based system.

With everything that has happened though throughout this project, I think I’ll still be able to deliver a functionally sound piece of software at the end. I’m pleased with how I’ve been able to overcome the challenges along the way, and I look forward to seeing the completed result.

Items to be considered for future iterations outside the scope of this project:

* Mobile App authenticates with user account
* Allow co-ordinates to be logged when checking in and out for last known location
* Make it so that managers need to approve when someone hasn’t checked in
* Allow the GUI to be sorted by manager
* Allow the managers staff to be delegated to another in case of annual leave
* Mobile App to store request locally in the case of no signal
* Mobile App to allow user to specify the length of time they are expected to be on site for, and the alert not to be triggered until this time if they haven’t checked in.
* The ability to manually check in an officer from the GUI, or maybe even extend the time for one hour

## Review Project Management

My project management skills throughout this project still leave some room for improvement. Although I was able to list most of the tasks I needed to complete, the timeframes that I set for them were completely unrealistic. I also forgot to take into account my commitments from my other modules. Between TMA02 and TMA03 I submitted two EMA’s and studied and sat an exam, all of which I had scheduled myself a full workload on this assignment. Being unable to commit the time during this 5 week period meant that I was doing around 25 hours a weeks during the final few weeks, rather the initial 10 I had anticipated.

On the positive side, I have managed to complete and maintain a project diary. I found this resource a massive benefit when it has come to writing up this EMA. All of the resources I had used had been listed, all of the mistakes and progression I had made were detailed, and this is a habit I will be keeping up with when I (hopefully) start my master’s degree in November.

Overall, I have completed a project on time, and I have managed to meet all of the specification that I have agreed with Fiona throughout the course of this module. I

## Assess risks to project completion

Originally when I started this project I identified the flowing risks:

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Impact** | **Chance of Occurrence** | **Mitigation** |
| Finished product does not work as intended | M | M | I can only try as hard as I can to complete the project. If there is a part that doesn’t work I’ll document what it is, and why it doesn’t work, and try and find another method to achieve the same result. |
| Client rejects the finished product | H | L | Meet with the client at each step to make sure that the product meets their expectations |
| Users are unable to access through mobile | L | M | This would only happen if they were in an area with low internet signal. In this case, if they ring the office staff can update the system on their behalf |
| Project becomes unmanageable | H | L | I have defined the scope to be within my control. If anything unexpected comes up, I’ll speak to the client and come to a mutual arrangement so the project can be completed on time |
| Lack of co-operation from users | H | L | The client is trying to get approval from management to make it a disciplinary action if they don’t use the system as intended |
| Unable to complete project in the timeframe given | L | M | As mentioned earlier, I’ll speak with the client to see if we can reduce the functionality, which can then be included in a future iteration. |
| Change in hardware available | L | L | There are no plans to change the hardware used at Northumberland County Council. We have just recently deployed new equipment which should be suitable for at least the next 10 years |

In reference to the above table:

* I have delivered a finished working product as intended.
* Due to the contact that I’ve had with my client, there is zero percent chance that they will reject the finished product. She has agreed the design and functionality at every stage of the project, and I’ve done what was necessary to implement this.
* The users not being able to access the app due to no signal is still a low risk, but that is out of my control. It is my intention in future iterations to include a function in the app to advise them to call the office if this is the case.
* As far as the project becoming unmanageable, I have co-ordinated with Fiona at each stage to make sure that the product specification is both required and reasonable. I think some of the original features may have led to a bigger project than I can handle at this time in my career, but as it is now, it is under control.
* The lack of co-operation from users isn’t an option; Fiona has had it explained that it is part of their contract to keep contact with the office, and it has now been approved that they could face disciplinary action if they do not check in.
* I think due to the reduced functionality, I have more than enough time to complete my project. I don’t think it can be reduced any more without losing some of its purpose, but I don’t see why it won’t be completed as planned before the deadline.
* At Northumberland County Council, we have no intention of changing our hardware for at least the next 5 years. Even if it was changed, Cordova wold allow the app to run on an iOS or Windows phone, and the GUI can run on any hardware that supports a Java install. At work we use a Citrix environment where a windows device isn’t available, so regardless of the hardware used this environment can support the Java program.

## Review of personal development

In review of my personal development, I am quite proud of how I’ve handled this project. This will be the first development project I’ve taken from inception to completion. I’m particularly please with the simple PHP files I’ve created. I’ve always thought of PHP as the language I would never understand, and even though what I accomplished was simple, it was enough to show I can understand and use unknown languages to produce the functionality I require.

I have noted in previous sections on how my project management skills have left much to be desired, but with the final assignment (the EMA), I have had no distractions. I have now completed all of my other studies and there was no reason why I couldn’t use the time available to complete a satisfactory project.

If I had to start the whole project again, there are many things I would change (I would’ve developed the whole thing in HTML for a start), but I’m glad I went down the route I have. It’s been a learning experience from start to finish, and I’m glad to say I have completed my first project, from inception to completion.

**Word Count: 8989 words**

# 3 References

Apache Cordova (n.d.) *Connection with the Backend* [Online]. Available at <https://www.studytonight.com/apache-cordova/connection-with-backened> (Accessed 01/07/2019)

Hock-Chuan, C (n.d.). *Programming Notes* [Online]. Available at <http://www.ntu.edu.sg/home/ehchua/programming/java/jdbc_basic.html> (Accessed 05/04/2019)

Oracle Docs (n.d.) *Package java.sql* [Online]. Available at <https://docs.oracle.com/javase/8/docs/api/index.html?java/sql/package-summary.html> [Accessed 04/04/2019).

SQLite Tutorial (n.d.) *SQLite Java: Connect To The SQLite Database Using SQLite JDBC Drive*r [Online]. Available at <http://www.sqlitetutorial.net/sqlite-java/sqlite-jdbc-driver/> (Accessed 04/04/2019).

Sundaravel (2019) *PhoneGap PHP MySQL Example* [Online]. Available at <https://codesundar.com/phonegap-php-mysql-example/> (Accessed 28/06/2019)

# 4 Appendices

[4.1 The initial enquiry email from the client](#_4.1_The_initial)

[4.2 Initial Meeting with the Client](#_4.3_–_Initial)

[4.3 Proposed designs for the Java GUI](#_4.3_Proposed_designs)

[4.4 Agreed designs for the Java GUI](#_4.4_Agreed_designs)

[4.5 JSwing pages](#_Appendix_4.5_–)

[4.6 Java Code](#_4.6_–_Java)

[4.7 Mobile App Protype](#_4.7_Mobile_App)

[4.8 Code for the Mobile App](#_4.8_Code_for)

[4.9 Proposed Design of the Mobile App v2](#_4.9_Proposed_Design)

[4.10 Screenshots of the Mobile App](#_4.10_Screenshots_of)

[4.11 PHP Code](#_4.11_PHP_Code)

[4.12 Protype for the new GUI](#_4.12_Protype_for)

[4.13 Java Code – Desktop App v2](#_4.13_Java_Code)

## 4.1 The initial enquiry email from the client

|  |  |
| --- | --- |
| Northumberland County Council Mail | **Nicky Temperley <nicky.temperley@northumberland.gov.uk>** |

|  |
| --- |
| **FW: Lone Working Policy** 1 message |

|  |  |
| --- | --- |
| **Michelle Williams**<michelle.williams@northumberland.gov.uk> | 13 February 2019 at 12:11 |
| To: Nicky Temperley <nicky.temperley@gs.northumberland.gov.uk> | |
| |  | | --- | | Just checking with Joe Murphy/ Joanne Southern that anything has been done yet?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ From: Fiona Charleton <[fiona.charleton@northumberland.gov.uk](mailto:fiona.charleton@northumberland.gov.uk)> Sent: 15 November 2018 16:32 To: Michelle Williams; Harri Bellizzi Subject: Fwd: Lone Working Policy  Hi Michelle/Harri  Sorry I know this is quite a bad time for you! I have been speaking with John Froud today about looking at the options of building a lone worker system for our Planning Officers who frequently go out on site. The system we use currently does not work so we need to review how we monitor our lone workers movements so we know where they are, who they are meeting and highlight when they haven't returned as expected.  I have spoken with Julie Seaton at Newcastle City Council. She is a former member of NCC staff and was very complimentary of the Newcastle system when she moved over 3 years ago. She has sent me a screenshot and brief overview as to how their system works. I have forwarded this on to IT for a call to be raised as to whether we can build something similar for our Officers. John has suggested I send this to both of you to look at the possibility of developing this. He is also keen to maybe widen this to other Services if we can build something that would work. Newcastle have been using this for a few years now and it does seem to work for them so we would be foolish to not investigate it when we are struggling with our current process.  Thanks Fiona  ---------- Forwarded message --------- From: Fiona Charleton <[fiona.charleton@northumberland.gov.uk](mailto:fiona.charleton@northumberland.gov.uk)<mailto:[fiona.charleton@northumberland.gov.uk](mailto:fiona.charleton@northumberland.gov.uk)>> Date: Thu, 15 Nov 2018 at 16:21 Subject: Fwd: Lone Working Policy To: John Froud <[john.froud@northumberland.gov.uk](mailto:john.froud@northumberland.gov.uk)<mailto:[john.froud@northumberland.gov.uk](mailto:john.froud@northumberland.gov.uk)>>   Hi John  Further to our discussion. Please see the below email which I mentioned I received from Newcastle City Council which details the home built system that they have been using with success for their Planning Officers.  I will update you when I know more from the call I have raised with IT. I will also send it separately to Michelle and Harri for them to look at.  Thanks for your help.  Fiona  ---------- Forwarded message --------- From: Seaton, Julie <[julie.seaton@newcastle.gov.uk](mailto:julie.seaton@newcastle.gov.uk)<mailto:[julie.seaton@newcastle.gov.uk](mailto:julie.seaton@newcastle.gov.uk)>> Date: Tue, 13 Nov 2018 at 12:26 Subject: RE: Lone Working Policy To: Fiona Charleton <[fiona.charleton@northumberland.gov.uk](mailto:fiona.charleton@northumberland.gov.uk)<mailto:[fiona.charleton@northumberland.gov.uk](mailto:fiona.charleton@northumberland.gov.uk)>>   Hi Fiona  Things are going OK here  We have an IT sytem that all officers must use when going out on site. They need to say where they are going and what time they will be back. It flashes an alert if they are not back by the time they stated and a manager must be notified. We monitor usage and it is a disciplinary offence if it’s not used. There are the odd occasion where people may go home straight from site and then we get them to call a ‘buddy’ back in the office when they are leaving the site – but it must be at a reasonable time when we know there will still be someone in the office who could raise the alarm if they don’t call.  All our pool cars (which staff use) have trackers on them so we could trace a car if we needed to.  This is a screen shot of the site visit database. Simon (our IT guru) put it together so he could possibly do something similar again.  Julie    -- Fiona Charleton Senior Technical Planning Officer Planning Northumberland County Council (01670) 622687   -- Fiona Charleton Senior Technical Planning Officer Planning Northumberland County Council (01670) 622687 | | |

## 4.2 Initial Meeting with the Client

Q. What are you expecting from this system as a minimum requirement

A. I would like a system to record who’s out on site, when they got there, when their expected to leave, and receive an alert if they haven’t left by this time. I’d like to be able the add or delete staff as they change too, although that will be mainly be down to the admin staff.

Q. What information do you store about the officers

A. We store their first name, surname, a mobile number, and their date of birth in case we have two people with the same name.

Q. What do you do with this information, is it accessible/displayed at all from the system?

A. I was thinking it could be made available so if there was an accident, or maybe just someone hasn’t checked back in in time then we’d have the information to hand without having to search through files to find it.

Q. Do you need the users to log in for security, or are their Windows Credentials enough to verify their identity?

A. No, theres no need for them to log in really.

Q. Do you need managers to have a different level of access to the admin staff? If so, what is the difference?

A. No, managers and staff can just have the same access really. Maybe this could change in the future, but for now its fine.

Q. Rather than tracking pool cars, do all the staff have company issued mobile phones?

A. Yes, all staff have a company issued mobile phone and they all have a voice and data package.

Q. How would you like to see the alert to managers? By email, screen notification, text message etc.

A. Just an alert on the screen would be fine. If its sent by email or text, it could be missed if they aren’t checked regularly. If it’s on the screen it cant be avoided.

Q. Do you need to record historic information, or once the site officer has confirmed they’ve left the site can the information be discarded?

A. Yes, we would need to save the information for a minimum of one year.

Q. On the screenshot you’ve provided, would the user complete the site they are visiting, or is that selected from a list of sites?

A. I was thinking that they could select the site from a list, or add it themselves if it’s a new site if that’s possible?

Q. Do you need to import any previous information into the system, or will you be using this as a fresh start?

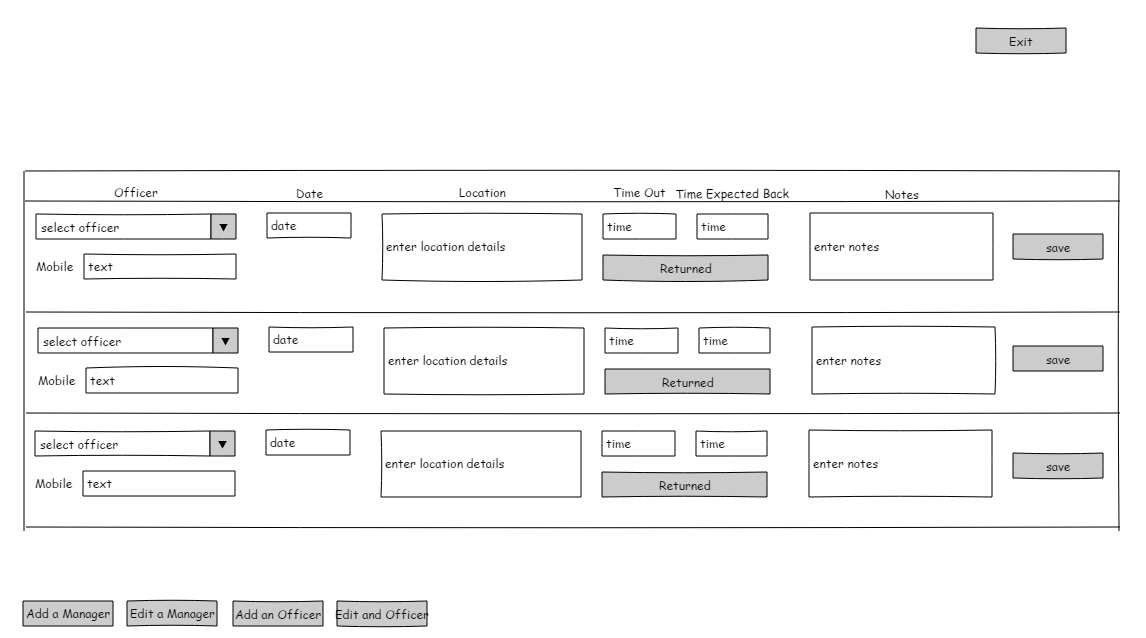
A. No, we’re going to use this system as new, no previous data will be imported into it, it’ll all be stored in its original state. If we need to refer back to it we’ll do it manually, there’s no need to link this new system to it.

Q. Do you a unique identifier for Officers and locations, or would a generic ID be enough?

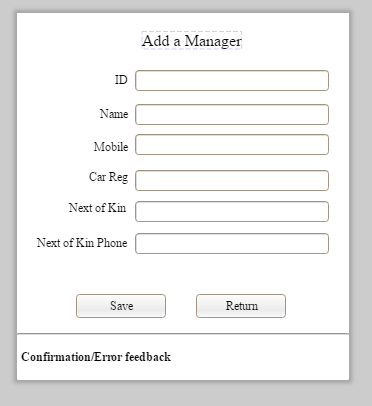
A. We use the staff payroll number as a unique ID for staff members, but the site ID isn’t relevant to anything we hold in our systems.

## 4.3 Proposed designs for the Java GUI

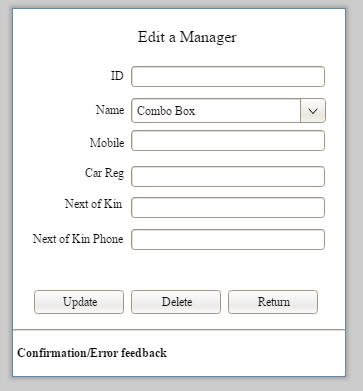
#### 4.3.1 Main User Interface



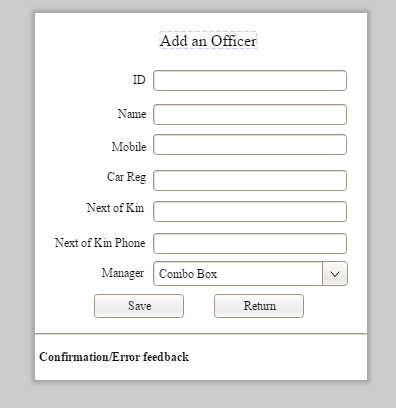
#### 4.3.2 Add a manager



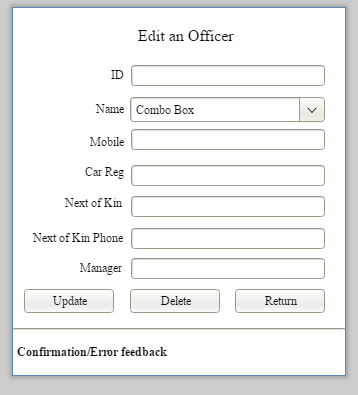
#### 4.3.3 Edit a Manager



#### 4.3.4 Add An Officer

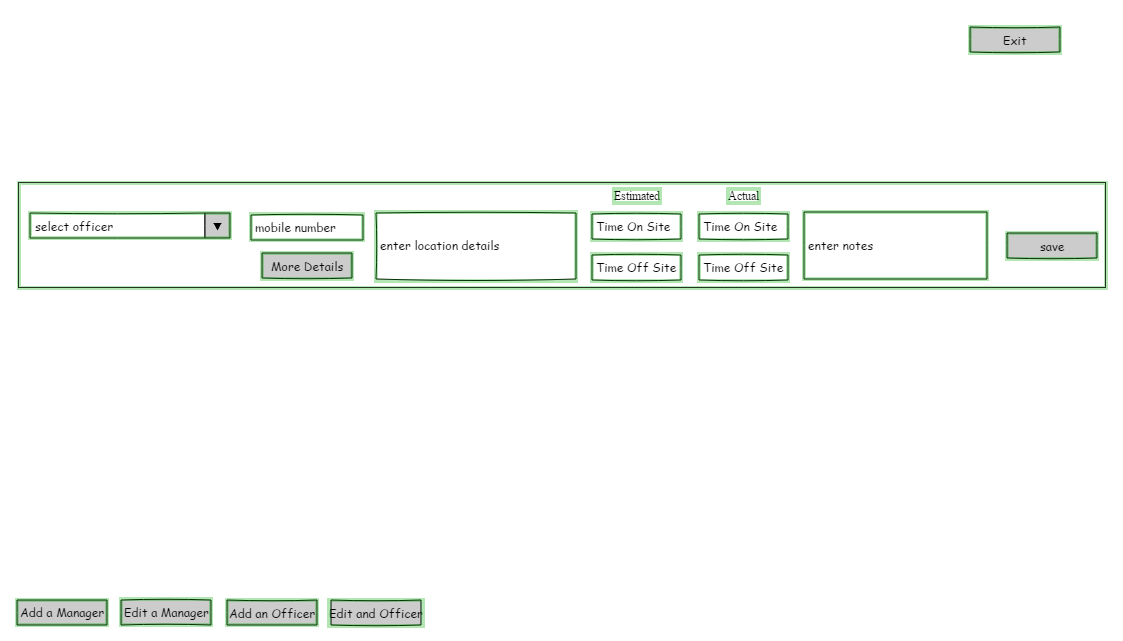


#### 4.3.5Edit An Officer

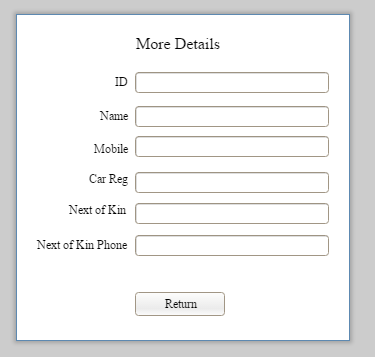


## 4.4 Agreed designs for the Java GUI

#### 4.4.1 Main User Interface



#### 4.4.2 More Details

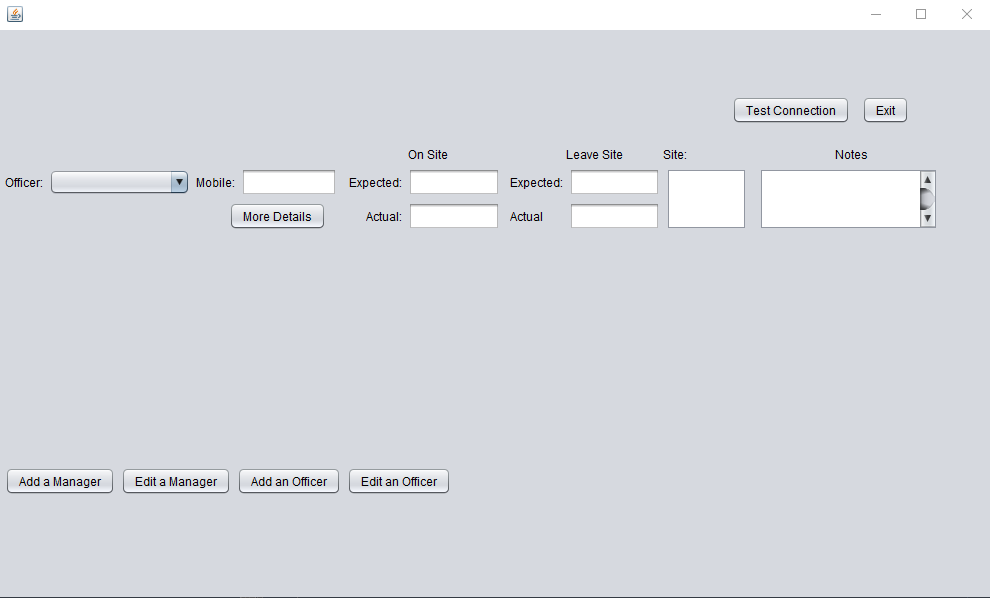


**All other pages for the GIU were agreed as proposed**

## 4.5 JSwing pages

#### 4.5.1 MainGUI,java

Exit the system



EditAManager.java

AddAManager.java

Only here for testing purposes

EditAnOfficer.java

AnnAnOfficer.java

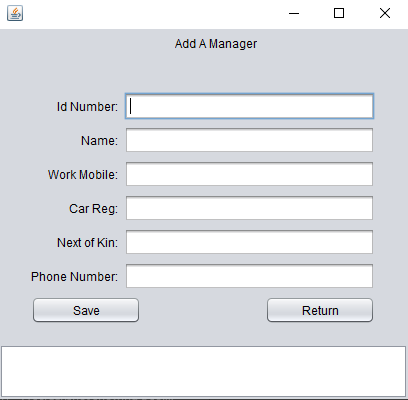
MoreDetails.java

For the purpose of this iteration, the users can:

* 1. Select an officer from the drop down box. This will populate the box with the officers name and the mobile field with their mobile number.
  2. Once an officer has been selected, the user can click on ‘More Details’ and see all the details held for that officer.
  3. Click ‘Add a Manager’ to go to the AddAManager.java page.
  4. Click ‘Edit a Manager’ to go to the EditAManager.java page.
  5. Click ‘Add an Officer’ to go to the AddAnOfficer.java page.
  6. Click ‘Edit an Officer’ to go to the EdditAnOfficer.java page.
  7. Click ‘Exit’ to close the system.

Please note, the ‘Test Connection’ button confirms connection to the database, and is only there for testing purposes. This will be remove in a later iteration.

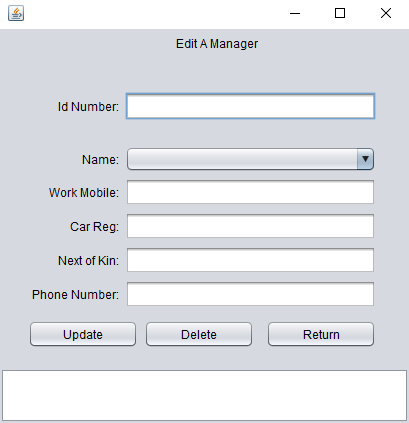
#### 4.5.2 AddAManager.java



MainGUI.java

The user can either enter all the required details here, or click return to go back to the main page. Once all the details have been entered and the user clicks save, either a confirmation message or an error message is displayed in the white bar at the bottom.

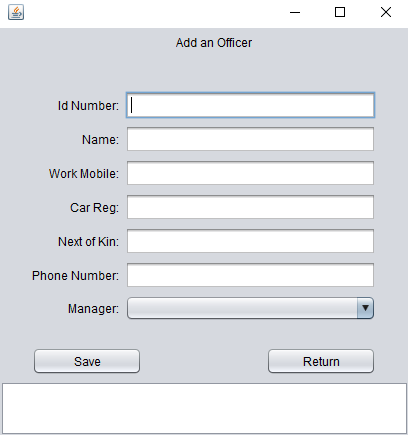
#### 4.5.3 EditAManager.java



MainGUI.java

On this page the user has to select the Manager they wish to amend from the drop-down list. Once selected they can either change any of the details and click update to save; click delete to remove the user from the database or return to the main screen. Once all the details have been entered and the user clicks save, either a confirmation message or an error message is displayed in the white bar at the bottom.

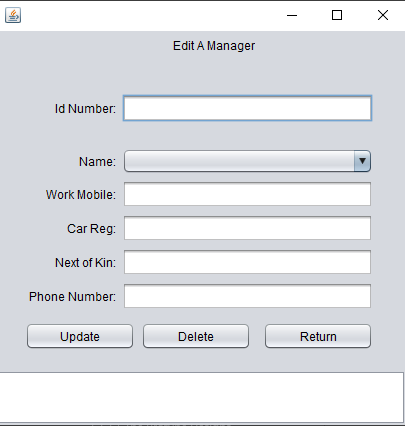
#### 4.5.4 AddAnOfficer.java



MainGUI.java

This form has the same functionality as AddAManager.java with the addition of the Manager drop down box; this needs to be selected when adding a new Officer. The manager drop-down provides a collection of all managers who are currently on the system. Once all the details have been entered and the user clicks save, either a confirmation message or an error message is displayed in the white bar at the bottom.

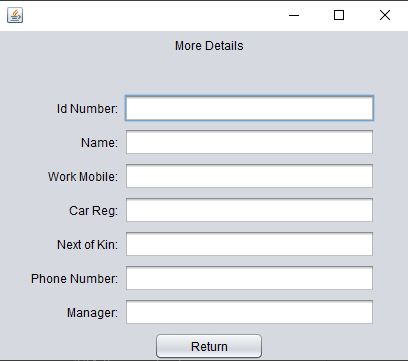
#### 4.5.5 EditAnOfficer.java



MainGUI.java

Again, this form has the same functionality as EditAManager.java with the addition of the Manager drop down box.

#### 4.5.6 More Details

  
  
This page has no functionality, the only option the user has is to click ‘Return’ and go back to the main page. The details cant be edited from this view either.

MainGUI.java

## 4.6 Java Code – Desktop App v1

#### 4.6.1 MainGUI.java

**I have removed all system generated code. For the full code please see the attached program in the zip directory**

|  |
| --- |
| package GUI;  import java.sql.Connection; import java.sql.DriverManager; import java.sql.PreparedStatement; import java.sql.ResultSet; import java.sql.SQLException; import java.sql.Statement; import java.util.List; import java.util.logging.Level; import java.util.logging.Logger; import javax.swing.JOptionPane; import tm470.DataConnect;  /\*\*  \*  \* @author zy395907  \*/ public class **MainGUI** extends **javax**.**swing**.**JFrame** {   private void **btnTestConnectionMouseClicked**(java.awt.event.MouseEvent evt) {   DataConnect a = new DataConnect();   a.testConnection();  }   private void **btnExitMouseClicked**(java.awt.event.MouseEvent evt) {   int dialogButton = JOptionPane.YES\_NO\_OPTION;   int dialogResult = JOptionPane.showConfirmDialog(this, "Are you sure?", "Close the system", dialogButton);   if (dialogResult == 0) {    System.exit(0);   } else {    System.out.println("No Option");   }  }   private void **btnAddManagerMouseClicked**(java.awt.event.MouseEvent evt) {   AddAManager aManager = new AddAManager();   aManager.setVisible(true);  }   private void **btnAddOfficerMouseClicked**(java.awt.event.MouseEvent evt) {   AddAnOfficer anOfficer = null;   try {    anOfficer = new AddAnOfficer();   } catch (SQLException ex) {    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);   }   anOfficer.setVisible(true);  }   private void **btnEditManagerMouseClicked**(java.awt.event.MouseEvent evt) {   EditAManager aManager = null;   try {    aManager = new EditAManager();   } catch (SQLException ex) {    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);   }   aManager.setVisible(true);  }   private void **cmboOfficerMouseClicked**(java.awt.event.MouseEvent evt) {   cmboOfficer.removeAllItems();   DataConnect dbc = new DataConnect();   List < String > officers = null;   try {    officers = dbc.getOfficers();   } catch (SQLException ex) {    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);   }   officers.forEach((officer) -> {    cmboOfficer.addItem(officer);   });  }   private void **btnMoreDetailsMouseClicked**(java.awt.event.MouseEvent evt) {   MoreDetails more = null;   try {    more = new MoreDetails();   } catch (SQLException ex) {    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);   }   String officerName = (String) cmboOfficer.getSelectedItem();   more.setVisible(true);   try {    more.getDetails(officerName);   } catch (SQLException ex) {    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);   }  }   private void **cmboOfficerPropertyChange**(java.beans.PropertyChangeEvent evt) {   System.out.println("Changed");  }   private void **cmboOfficerActionPerformed**(java.awt.event.ActionEvent evt) {   String vdbDriver = "jdbc:sqlite:";   String vDatabase = "D:/Program Files/SQLiteDatabaseBrowserPortable/Data/TM470.db";   Connection vConnection = null;   String name = (String) this.cmboOfficer.getSelectedItem();   if (name != null) {    try {     vConnection = DriverManager.getConnection(vdbDriver + vDatabase);     Statement stmt = vConnection.createStatement();     String query = "SELECT mobileNumber FROM Officers WHERE name = \"" + name + "\"";     ResultSet results = stmt.executeQuery(query);     txtMobile.setText(results.getString(1));    } catch (SQLException ex) {     Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);    }   }  } |

#### 4.6.2 AddAManager.java

**I have removed all system generated code. For the full code please see the attached program in the zip directory**

|  |
| --- |
| package GUI;  import java.sql.\*; import java.util.logging.Level; import java.util.logging.Logger;  /\*\*  \*  \* @author zy395907  \*/ public class **AddAManager** extends **javax**.**swing**.**JFrame** {       private void **btnReturnMouseClicked**(java.awt.event.MouseEvent evt) {                                               this.setVisible(false);     }                                        private void **btnSaveMouseClicked**(java.awt.event.MouseEvent evt) {                                               try {             String id;             String name;             String mobileNumber;             String carReg;             String nextOfKin;             String nextOfKinPhone;             String vdbDriver = "jdbc:sqlite:";             String vDatabase = "D:/Program Files/SQLiteDatabaseBrowserPortable/Data/TM470.db";             Connection vConnection;              id = txtId.getText();             name = txtName.getText();             mobileNumber = txtMobileNumber.getText();             carReg = txtCarReg.getText();             nextOfKin = txtNextOfKin.getText();             nextOfKinPhone = txtNextOfKinPhone.getText();             vConnection = DriverManager.getConnection(vdbDriver + vDatabase);             PreparedStatement st = vConnection.prepareStatement("insert into MANAGERS(id, name, mobileNumber, carReg, nextOfKin, nextOfKinPhone) values (?,?,?,?,?,?)");             st.setString(1, id);             st.setString(2, name);             st.setString(3, mobileNumber);             st.setString(4, carReg);             st.setString(5, nextOfKin);             st.setString(6, nextOfKinPhone);             int a = st.executeUpdate();             if (a > 0) {                 txtAddManagerOutcome.setText((String)name + " has been added, you may \n"                     + "now add another manager or close this screen by clicking Return");                 txtId.setText("");                 txtName.setText("");                 txtMobileNumber.setText("");                 txtCarReg.setText("");                 txtNextOfKin.setText("");                 txtNextOfKinPhone.setText("");             }         } catch (SQLException ex) {             Logger.getLogger(AddAManager.class.getName()).log(Level.SEVERE, null, ex);             txtAddManagerOutcome.setText("There has been an error \n"                     + "please check the data above and try again");         }      } |

#### 4.6.3 EditAManager.java

|  |
| --- |
| **I have removed all system generated code. For the full code please see the attached program in the zip directory**  package GUI;  import java.sql.Connection; import java.sql.DriverManager; import java.sql.PreparedStatement; import java.sql.SQLException; import java.util.List; import java.util.logging.Level; import java.util.logging.Logger; import tm470.DataConnect;  /\*\*  \*  \* @author zy395907  \*/ public class **EditAnOfficer** extends **javax**.**swing**.**JFrame** {      /\*\*      \* Creates new form AddAnOfficer      \* @throws java.sql.SQLException      \*/     public **EditAnOfficer**() throws SQLException {         initComponents();         cmboName.removeAllItems();         DataConnect dbc = new DataConnect();         List<String> managers = dbc.getManagers();         managers.forEach((manager) -> {             cmboName.addItem(manager);         });        }       private void **btnReturnMouseClicked**(java.awt.event.MouseEvent evt) {                                                this.setVisible(false);     } |

#### 4.6.4 AddAnOfficer.java

|  |
| --- |
| **I have removed all system generated code. For the full code please see the attached program in the zip directory**  package GUI;  import java.sql.Connection; import java.sql.DriverManager; import java.sql.PreparedStatement; import java.sql.SQLException; import java.util.List; import java.util.logging.Level; import java.util.logging.Logger; import tm470.DataConnect;  /\*\*  \*  \* @author zy395907  \*/ public class **AddAnOfficer** extends **javax**.**swing**.**JFrame** {      /\*\*      \* Creates new form AddAnOfficer      \* @throws java.sql.SQLException      \*/     public **AddAnOfficer**() throws SQLException {         initComponents();         cmboManager.removeAllItems();         DataConnect dbc = new DataConnect();         List<String> managers = dbc.getManagers();         managers.forEach((manager) -> {             cmboManager.addItem(manager);         });        }       private void **btnReturnMouseClicked**(java.awt.event.MouseEvent evt) {                                                this.setVisible(false);     }                                         private void **btnSaveMouseClicked**(java.awt.event.MouseEvent evt) {                                              try {             String id;             String name;             String mobileNumber;             String carReg;             String nextOfKin;             String nextOfKinPhone;             String manager;             String vdbDriver = "jdbc:sqlite:";             String vDatabase = "D:/Program Files/SQLiteDatabaseBrowserPortable/Data/TM470.db";             Connection vConnection = null;                                       id = txtId.getText();             name = txtName.getText();             mobileNumber = txtMobileNumber.getText();             carReg = txtCarReg.getText();             nextOfKin = txtNextOfKin.getText();             nextOfKinPhone = txtNextOfKinPhone.getText();             manager = (String)cmboManager.getSelectedItem();             Connection con = DriverManager.getConnection(vdbDriver + vDatabase);             PreparedStatement st = con.prepareStatement("insert into officers(id, name, mobileNumber, carReg, nextOfKin, nextOfKinPhone, manager) values (?,?,?,?,?,?,?)");             st.setString(1, id);             st.setString(2, name);             st.setString(3, mobileNumber);             st.setString(4, carReg);             st.setString(5, nextOfKin);             st.setString(6, nextOfKinPhone);             st.setString(7, manager);             int a = st.executeUpdate();             if (a>0)             {                 txtAddManagerOutcome.setText((String)name + " has been added, you may \n"                     + "now close this screen by clicking Return");                 txtId.setText("");                 txtName.setText("");                 txtMobileNumber.setText("");                 txtCarReg.setText("");                 txtNextOfKin.setText("");                 txtNextOfKinPhone.setText("");;                 cmboManager.setSelectedIndex(-1);                              }         } catch (SQLException ex) {             txtAddManagerOutcome.setText("There has been an error \n"                     + "please check the data above and try again");             Logger.getLogger(AddAnOfficer.class.getName()).log(Level.SEVERE, null, ex);         }          } |

#### 4.6.5 EditAnOfficer.java

|  |
| --- |
| **I have removed all system generated code. For the full code please see the attached program in the zip directory**  package GUI;  import java.sql.Connection; import java.sql.DriverManager; import java.sql.PreparedStatement; import java.sql.SQLException; import java.util.List; import java.util.logging.Level; import java.util.logging.Logger; import tm470.DataConnect;  /\*\*  \*  \* @author zy395907  \*/ public class **EditAnOfficer** extends **javax**.**swing**.**JFrame** {      /\*\*      \* Creates new form AddAnOfficer      \* @throws java.sql.SQLException      \*/     public **EditAnOfficer**() throws SQLException {         initComponents();         cmboName.removeAllItems();         DataConnect dbc = new DataConnect();         List<String> managers = dbc.getManagers();         managers.forEach((manager) -> {             cmboName.addItem(manager);         });        }       private void **btnReturnMouseClicked**(java.awt.event.MouseEvent evt) {                                                this.setVisible(false);     } |

#### 4.6.6 MoreDetails.java

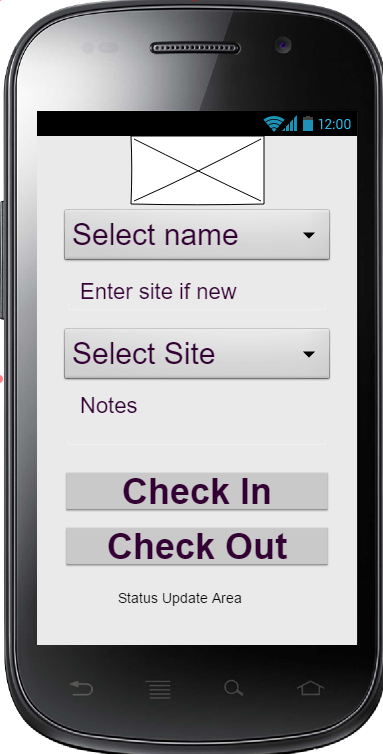
**I have removed all system generated code. For the full code please see the attached program in the zip directory**

|  |
| --- |
| package GUI;  import java.sql.Connection; import java.sql.DriverManager; import java.sql.PreparedStatement; import java.sql.ResultSet; import java.sql.SQLException; import java.sql.Statement; import java.util.List; import java.util.logging.Level; import java.util.logging.Logger; import tm470.DataConnect;  /\*\*  \*  \* @author zy395907  \*/ public class **MoreDetails** extends **javax**.**swing**.**JFrame** {      /\*\*      \* Creates new form AddAnOfficer      \* @param name      \* @throws java.sql.SQLException      \*/     public **MoreDetails**() throws SQLException {         initComponents();              }       private void **btnReturnMouseClicked**(java.awt.event.MouseEvent evt) {                                                this.setVisible(false);     }                                         private void **txtNameMouseExited**(java.awt.event.MouseEvent evt) {                                                  }                                          /\*\*      \* Get the details for the officer sselected in MainGUI      \* @param officerName      \* @throws SQLException       \*/     public void **getDetails**(String officerName) throws SQLException {         String vdbDriver = "jdbc:sqlite:";         String vDatabase = "D:/Program Files/SQLiteDatabaseBrowserPortable/Data/TM470.db";         Connection vConnection = null;                  txtName.setText(officerName);         vConnection = DriverManager.getConnection(vdbDriver + vDatabase);         Statement stmt = vConnection.createStatement();             String query = "SELECT \* FROM Officers WHERE name = \"" + officerName + "\"";             ResultSet results = stmt.executeQuery(query);              txtId.setText(results.getString(1));             txtMobileNumber.setText(results.getString(3));             txtCarReg.setText(results.getString(4));             txtNextOfKin.setText(results.getString(5));             txtNextOfKinPhone.setText(results.getString(6));             txtManager.setText(results.getString(7));                             } |

#### 4.6.7 DataConnect.java

|  |
| --- |
| package tm470;  import java.sql.\*; import java.util.\*;  public class **DataConnect** {     String vdbDriver = "jdbc:sqlite:";  String vDatabase = "D:/Program Files/SQLiteDatabaseBrowserPortable/Data/TM470.db";  Connection vConnection = null;                           /\*\*          \* method to test database connection which returns a boolean value          \*/  public boolean **testConnection**(){    //try connecting to database, if successful show success message and return true boolean value  try {    vConnection = (Connection)DriverManager.getConnection(vdbDriver + vDatabase);    System.out.println("Connection to database successful!");    return true;      //catch will show unsuccessful message and return false boolean value     } catch (Exception ex) {    System.err.println ("Connection unsuccessful\n" + ex.toString ());    return false;  }     }                                                         /\*\*      \* return the list of managers to populate the manager list in cmboManager      \* on AddAnOfficer.java      \*      \* @return List of managers      \* @throws java.sql.SQLException      \*/     public List<String> **getManagers**() throws SQLException {         List<String> list = new ArrayList();         vConnection = DriverManager.getConnection(vdbDriver + vDatabase);         try {             Statement stmt = vConnection.createStatement();             String query = "SELECT name FROM managers ORDER BY name";             ResultSet results = stmt.executeQuery(query);             list.add(null);             while (results.next()) {                 list.add(results.getString("name"));             }                     } catch (SQLException e) {             System.out.println("Exception = " + e);         }         return list;     }          public List<String> **getOfficers**() throws SQLException {         List<String> list = new ArrayList();         vConnection = DriverManager.getConnection(vdbDriver + vDatabase);         try {             Statement stmt = vConnection.createStatement();             String query = "SELECT name FROM officers ORDER BY name";             ResultSet results = stmt.executeQuery(query);             list.add(null);             while (results.next()) {                 list.add(results.getString("name"));             }             stmt.close();         } catch (SQLException e) {             System.out.println("Exception = " + e);         }         return list;     }              public ResultSet **getManagersDetails**(String name) throws SQLException {         List<String> list = new ArrayList();         vConnection = DriverManager.getConnection(vdbDriver + vDatabase);         try {             Statement stmt = vConnection.createStatement();             String query = "SELECT \* FROM managers WHERE name = \"" + name + "\" ORDER BY name";             ResultSet results = stmt.executeQuery(query);                          return results;                      } catch (SQLException e) {             System.out.println("Exception = " + e);         }         //System.out.println(list);         return null;     }               public void **deleteManager**(String managerId) throws SQLException{             vConnection = DriverManager.getConnection(vdbDriver + vDatabase);             PreparedStatement st = vConnection.prepareStatement("DELETE FROM Managers WHERE id = \" + managerId + \"");             st.executeUpdate();                   }    } |

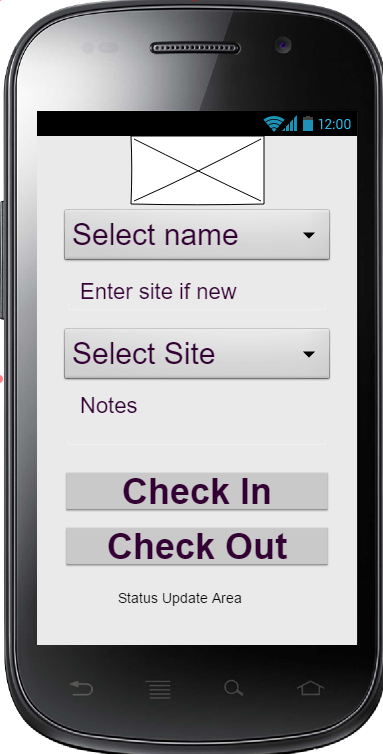
## 4.7 Mobile App Protype



## 4.8 Code for the Mobile App

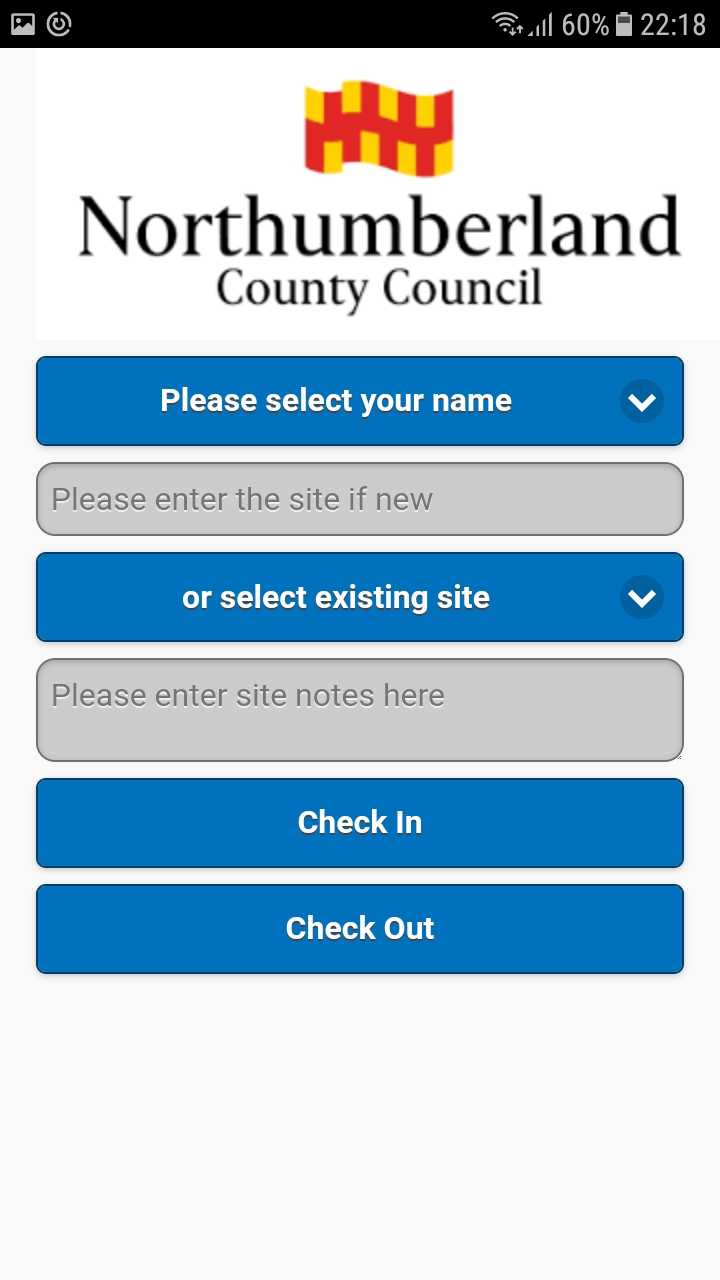
1. <html>
2. <head>
3. <meta charset="utf-8" />
4. <meta name="viewport" content="width=device-width, initial-scale=1">
5. <link rel="stylesheet" href="css/index.css" />
6. <link rel="stylesheet" href="css/themes/app.css" />
7. <link rel="stylesheet" href="css/themes/jquery.mobile.icons.min.css" />
8. <link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile.structure-1.4.5.min.css" />
9. <script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
10. <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
11. <script type="text/javascript"> //function to insert officer names from db into dropdown list
12. $(document).ready(function () {
13. var url = "http://192.168.1.247/getNames.php"; //ip of local host over ethernet
14. $.getJSON(url, function (result) {
15. $.each(result, function (i, field) {
16. var name = field.name;
17. $("#name").append("<option value='" + name + "'>" + name + "</option>");
18. });
19. });
20. });
21. </script>
22. <script type="text/javascript"> //function to insert existing sites from db into dropdown list
23. $(document).ready(function () {
24. var url = "http://192.168.1.247/getSites.php"; //ip of local host over ethernet
25. $.getJSON(url, function (result) {
26. $.each(result, function (i, field) {
27. var siteName = field.siteName;
28. $("#existingSite").append("<option value='" + siteName + "'>" + siteName + "</option>");
29. });
30. });
31. });
32. </script>
33. <script type="text/javascript"> //function to check in
34. $(document).ready(function ()
35. {
36. $("#checkIn").click(function () {
37. var action = "";
38. var notes = "";
39. if ($("#site").val() === "" && $("#existingSite option:selected").val()!=="Please enter the site if new") {
40. var site = $("#existingSite option:selected").val();
41. action = "not new";
42. } else {
43. var site = $("#site").val();
44. action = "new";
45. }
46. notes = $("#notes").val();
47. var name = $("#name option:selected").val();
48. var url = "http://192.168.1.247/checkIn.php";
49. var dataString = "site=" + site + "&notes=" + notes + "&name=" + name + "&action=" + action;
50. if ($.trim(site).length > 0 & $(name) !== "Please select your name")
51. {
52. $.ajax({
53. type: "POST",
54. url: url,
55. data: dataString,
56. crossDomain: true,
57. cache: false,
58. success: function (data) {
59. if (data === "Success")
60. {
61. $("#status").html('Checked in Successfully');
62. $("#site").val("");
63. $("#notes").val("");
64. } else if (data === "Error")
65. {
66. alert("error");
67. }
68. }
69. });
70. } else {
71. alert("Please make sure you have selected your name, entered a new or existing site, and entered any notes if required");
72. }
73. });
74. });
75. </script>
76. <script type="text/javascript"> //function to check out
77. $(document).ready(function ()
78. {
79. $("#checkOut").click(function () {
80. var site = $("#existingSite option:selected").val();
81. var notes = $("#notes").val();
82. var name = $("#name option:selected").val();
83. var url = "http://192.168.1.247/checkOut.php";
84. var dataString = "site=" + site + "&notes=" + notes + "&name=" + name;
85. if ($(site) !== "existingSite" & $(name) !== "Please select your name")
86. {
87. $.ajax({
88. type: "POST",
89. url: url,
90. data: dataString,
91. crossDomain: true,
92. cache: false,
93. success: function (data) {
94. if (data === "Success")
95. {
96. $("#status").html('Checked out Successfully');
97. $("#site").val("");
98. $("#notes").val("");
99. $("#name").val("Please select your name");
100. $("#existingSite").val("Please select your name");
101. } else if (data === "Error")
102. {
103. alert("error");
104. }
105. }
106. });
107. }
108. return false;
109. });
110. });
111. </script>
113. <title>test</title>
114. </head>
115. <body>
116. <div class="app">
117. <img src="data:image/png;base64," alt="NCC Logo">
118. <select id="name">
119. <option value="name">Please select your name</option>
120. </select>
121. <input type="text" placeholder="Please enter the site if new" id="site">
122. <select id="existingSite">
123. <option value="name" >or select existing site</option>
124. </select>
125. <textarea id="notes" placeholder="Please enter site notes here"></textarea>
126. <button id="checkIn">Check In</button>
127. <button id="checkOut">Check Out</button>
128. <div id="status"></div>
129. </div>
130. </body>
131. </html>

## 4.9 Proposed Design of the Mobile App v2

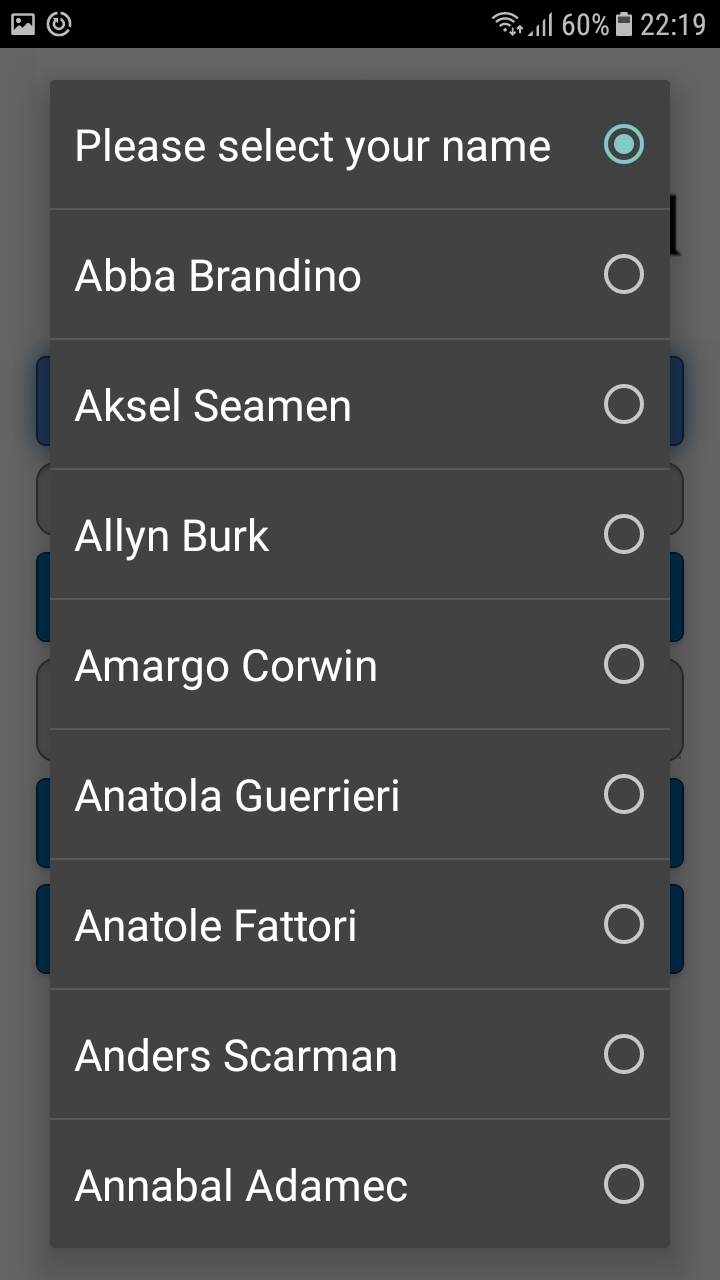


## 4.10 Screenshots of the Mobile App

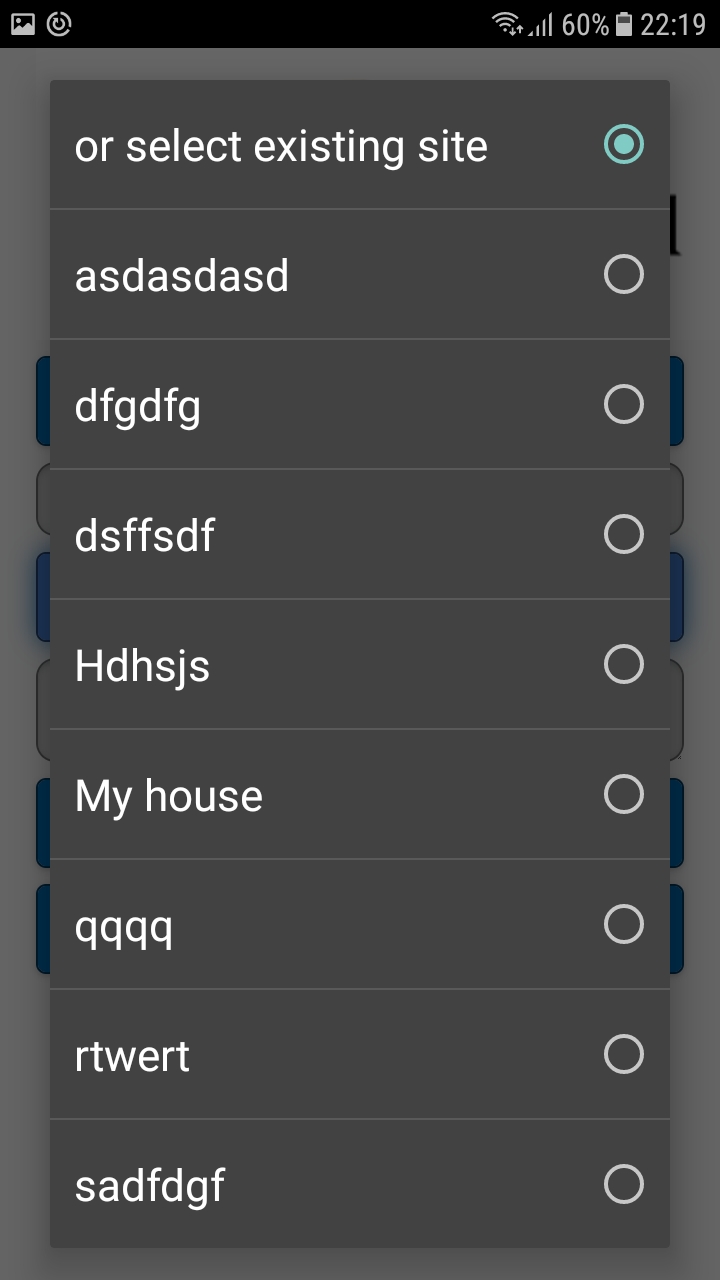
#### 4.10.1 Main Screen



#### 4.10.2 Menu Showing List of Names



#### 4.10.3 Menu Showing Site List



## 4.11 PHP Code

#### 4.11.1 db.php

1. <?php
2. header("Access-Control-Allow-Origin: \*");
3. $con = mysqli\_connect("localhost","root","","tm470") or die ("could not connect database");
4. ?>

#### 4.11.2 getNames.php

1. <?php
2. include "db.php";
3. $data=array();
4. $q=mysqli\_query($con,"select name from `tblofficers` ORDER BY name ASC");
5. while ($row=mysqli\_fetch\_object($q)){
6. $data[]=$row;
7. }
8. echo json\_encode($data);
9. ?>

#### 4.11.3 getSites.php

1. <?php
2. include "db.php";
3. $data=array();
4. $q=mysqli\_query($con,"select siteName from `tblsite` ORDER BY siteName ASC");
5. while ($row=mysqli\_fetch\_object($q)){
6. $data[]=$row;
7. }
8. echo json\_encode($data);
9. ?>

#### 4.11.4 checkIn.php

1. <?php
2. include "db.php";
4. $site=$\_POST['site'];
5. $notes=$\_POST['notes'];
6. $name=$\_POST['name'];
7. $action=$\_POST['action'];
8. $event="in";

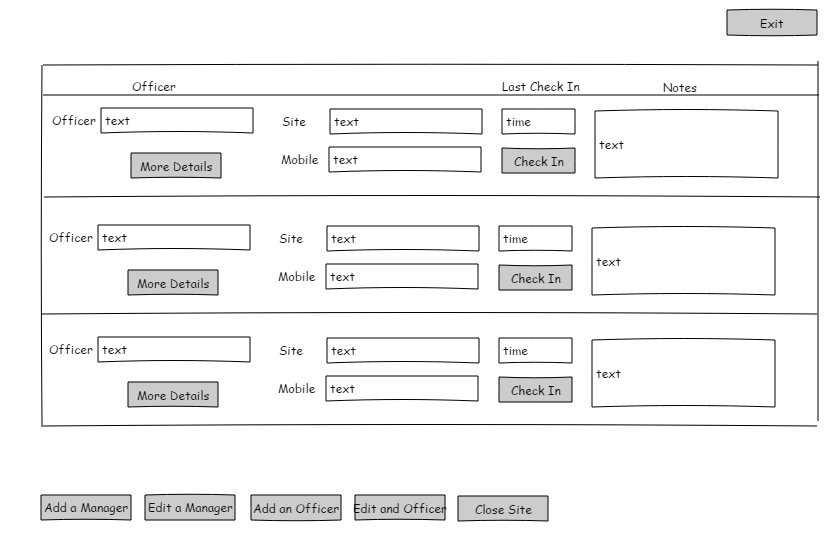
11. $sql = "INSERT INTO tbllogs (site, notes, officer, in\_out)
12. VALUES ('$site', '$notes', '$name', '$event')";
14. $sql2 = "INSERT INTO tblsite (siteName, officerName)
15. VALUES ('$site', '$name')";
17. $sql3 = "INSERT INTO tblcurrentvisits (site, name)
18. VALUES ('$site', '$name')";
20. if ($action === "new"){
21. $con->query($sql2);
22. }
24. if ($con->query($sql) === TRUE) {
25. ($con->query($sql3));
26. echo "Success";
27. } else {
28. echo "Error";
29. }
31. $con->close();
32. ?>

#### 4.11.5 checkOut.php

1. <?php
2. include "db.php";
4. $site=$\_POST['site'];
5. $notes=$\_POST['notes'];
6. $name=$\_POST['name'];
7. $event="out";

10. $sql = "INSERT INTO tbllogs (site, notes, officer, in\_out)
11. VALUES ('$site', '$notes', '$name', '$event')";
13. $sql2 = "DELETE FROM tblcurrentvisits
14. WHERE name='$name'";
16. if (($con->query($sql) === TRUE) && ($con->query($sql2) === TRUE)) {
17. echo "Success";
18. } else {
19. echo "Error";
20. }
22. $con->close();
23. ?>

## 4.12 Protype for the new GUI



## 4.13 Java Code – Desktop App v2

#### 4.13.1 MainGUI.java

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** GUI;
8. **import** desktop.app.v2.DBConnect;
9. **import** java.awt.Toolkit;
10. **import** java.sql.Connection;
11. **import** java.sql.ResultSet;
12. **import** java.sql.SQLException;
13. **import** java.sql.Statement;
14. **import** java.text.ParseException;
15. **import** java.text.SimpleDateFormat;
16. **import** java.util.ArrayList;
17. **import** java.util.Date;
18. **import** java.util.Timer;
19. **import** java.util.TimerTask;
20. **import** java.util.logging.Level;
21. **import** java.util.logging.Logger;
22. **import** javax.swing.JFrame;
23. **import** javax.swing.JOptionPane;
24. **import** javax.swing.table.DefaultTableModel;
26. /\*\*
27. \*
28. \* @author zy395907
29. \*/
30. **public** **class** MainGUI **extends** javax.swing.JFrame {
32. Toolkit toolkit;
33. Timer timer;
35. /\*\*
36. \* Creates new form MainGUI
37. \*/
38. **public** MainGUI() {
39. initComponents();
40. setExtendedState(getExtendedState() | JFrame.MAXIMIZED\_BOTH); //sets the screen to maximise
42. timer = **new** Timer();
43. timer.schedule(**new** RemindTask(),
44. 0, //initial delay
45. 20 \* 1000);  //subsequent rate
46. }
48. **class** RemindTask **extends** TimerTask {
50. @Override
51. **public** **void** run() {
52. **try** {
53. loadTable();
54. } **catch** (ParseException ex) {
55. Logger.getLogger(MainGUI.**class**.getName()).log(Level.SEVERE, **null**, ex);
56. }
57. }
58. }
60. /\*\*
61. \* This method is called from within the constructor to initialize the form.
62. \* WARNING: Do NOT modify this code. The content of this method is always
63. \* regenerated by the Form Editor.
64. \*/
65. @SuppressWarnings("unchecked")
66. // <editor-fold defaultstate="collapsed" desc="Generated Code">
67. **private** **void** initComponents() {
69. jPanel1 = **new** javax.swing.JPanel();
70. jLabel1 = **new** javax.swing.JLabel();
71. jButton6 = **new** javax.swing.JButton();
72. jPanel2 = **new** javax.swing.JPanel();
73. jButton1 = **new** javax.swing.JButton();
74. jButton2 = **new** javax.swing.JButton();
75. jButton3 = **new** javax.swing.JButton();
76. jButton4 = **new** javax.swing.JButton();
77. jButton5 = **new** javax.swing.JButton();
78. jPanel3 = **new** javax.swing.JPanel();
79. jScrollPane2 = **new** javax.swing.JScrollPane();
80. tblSiteVisits = **new** javax.swing.JTable();
81. jButton7 = **new** javax.swing.JButton();
83. setDefaultCloseOperation(javax.swing.WindowConstants.DO\_NOTHING\_ON\_CLOSE);
84. setBackground(**new** java.awt.Color(255, 255, 255));
85. setCursor(**new** java.awt.Cursor(java.awt.Cursor.DEFAULT\_CURSOR));
87. jPanel1.setBackground(**new** java.awt.Color(255, 255, 255));
89. jLabel1.setBackground(**new** java.awt.Color(255, 255, 255));
90. jLabel1.setFont(**new** java.awt.Font("Arial", 1, 48)); // NOI18N
91. jLabel1.setForeground(**new** java.awt.Color(255, 0, 0));
92. jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
93. jLabel1.setText("NCC Lone Worker App");
95. jButton6.setText("Exit");
96. jButton6.addMouseListener(**new** java.awt.event.MouseAdapter() {
97. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
98. jButton6MouseClicked(evt);
99. }
100. });
102. javax.swing.GroupLayout jPanel1Layout = **new** javax.swing.GroupLayout(jPanel1);
103. jPanel1.setLayout(jPanel1Layout);
104. jPanel1Layout.setHorizontalGroup(
105. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
106. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
107. .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
108. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
109. .addComponent(jButton6))
110. );
111. jPanel1Layout.setVerticalGroup(
112. jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
113. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
114. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
115. .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
116. .addComponent(jLabel1)
117. .addComponent(jButton6))
118. .addContainerGap())
119. );
121. jPanel2.setBackground(**new** java.awt.Color(255, 255, 255));
123. jButton1.setText("Add a Manager");
124. jButton1.addMouseListener(**new** java.awt.event.MouseAdapter() {
125. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
126. jButton1MouseClicked(evt);
127. }
128. });
130. jButton2.setText("Add an Officer");
131. jButton2.addMouseListener(**new** java.awt.event.MouseAdapter() {
132. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
133. jButton2MouseClicked(evt);
134. }
135. });
137. jButton3.setText("Edit or Delete a Manager");
138. jButton3.addMouseListener(**new** java.awt.event.MouseAdapter() {
139. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
140. jButton3MouseClicked(evt);
141. }
142. });
144. jButton4.setText("Edit or Delete an Officer");
145. jButton4.addMouseListener(**new** java.awt.event.MouseAdapter() {
146. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
147. jButton4MouseClicked(evt);
148. }
149. });
151. jButton5.setText("Delete Site");
152. jButton5.addMouseListener(**new** java.awt.event.MouseAdapter() {
153. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
154. jButton5MouseClicked(evt);
155. }
156. });
158. javax.swing.GroupLayout jPanel2Layout = **new** javax.swing.GroupLayout(jPanel2);
159. jPanel2.setLayout(jPanel2Layout);
160. jPanel2Layout.setHorizontalGroup(
161. jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
162. .addGroup(jPanel2Layout.createSequentialGroup()
163. .addContainerGap()
164. .addComponent(jButton1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
165. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
166. .addComponent(jButton2, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
167. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
168. .addComponent(jButton3, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
169. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
170. .addComponent(jButton4, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
171. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
172. .addComponent(jButton5, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
173. .addContainerGap())
174. );
175. jPanel2Layout.setVerticalGroup(
176. jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
177. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()
178. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
179. .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
180. .addComponent(jButton1)
181. .addComponent(jButton2)
182. .addComponent(jButton3)
183. .addComponent(jButton4)
184. .addComponent(jButton5))
185. .addGap(50, 50, 50))
186. );
188. jPanel3.setBackground(**new** java.awt.Color(255, 255, 255));
190. jScrollPane2.setBackground(**new** java.awt.Color(255, 255, 255));
192. tblSiteVisits.setModel(**new** javax.swing.table.DefaultTableModel(
193. **new** Object [][] {
195. },
196. **new** String [] {
197. "Name", "Site", "Time"
198. }
199. ) {
200. **boolean**[] canEdit = **new** **boolean** [] {
201. **false**, **false**, **false**
202. };
204. **public** **boolean** isCellEditable(**int** rowIndex, **int** columnIndex) {
205. **return** canEdit [columnIndex];
206. }
207. });
208. tblSiteVisits.setGridColor(**new** java.awt.Color(0, 0, 0));
209. tblSiteVisits.getTableHeader().setReorderingAllowed(**false**);
210. jScrollPane2.setViewportView(tblSiteVisits);
211. **if** (tblSiteVisits.getColumnModel().getColumnCount() > 0) {
212. tblSiteVisits.getColumnModel().getColumn(0).setResizable(**false**);
213. tblSiteVisits.getColumnModel().getColumn(1).setResizable(**false**);
214. tblSiteVisits.getColumnModel().getColumn(2).setResizable(**false**);
215. }
217. jButton7.setText("Officer's Details");
218. jButton7.addMouseListener(**new** java.awt.event.MouseAdapter() {
219. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
220. jButton7MouseClicked(evt);
221. }
222. });
224. javax.swing.GroupLayout jPanel3Layout = **new** javax.swing.GroupLayout(jPanel3);
225. jPanel3.setLayout(jPanel3Layout);
226. jPanel3Layout.setHorizontalGroup(
227. jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
228. .addComponent(jScrollPane2)
229. .addGroup(jPanel3Layout.createSequentialGroup()
230. .addContainerGap(337, Short.MAX\_VALUE)
231. .addComponent(jButton7)
232. .addContainerGap(337, Short.MAX\_VALUE))
233. );
234. jPanel3Layout.setVerticalGroup(
235. jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
236. .addGroup(jPanel3Layout.createSequentialGroup()
237. .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 326, javax.swing.GroupLayout.PREFERRED\_SIZE)
238. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 12, Short.MAX\_VALUE)
239. .addComponent(jButton7)
240. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))
241. );
243. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
244. getContentPane().setLayout(layout);
245. layout.setHorizontalGroup(
246. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
247. .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
248. .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
249. .addComponent(jPanel3, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
250. );
251. layout.setVerticalGroup(
252. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
253. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
254. .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
255. .addGap(0, 0, 0)
256. .addComponent(jPanel3, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
257. .addGap(0, 0, 0)
258. .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 59, javax.swing.GroupLayout.PREFERRED\_SIZE)
259. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))
260. );
262. pack();
263. }// </editor-fold>
265. **private** **void** jButton1MouseClicked(java.awt.event.MouseEvent evt) {
266. AddAManager aManager = **new** AddAManager();
267. aManager.setVisible(**true**);
268. }
270. **private** **void** jButton3MouseClicked(java.awt.event.MouseEvent evt) {
271. **try** {
272. EditAManager aManager = **new** EditAManager();
273. aManager.setVisible(**true**);
274. } **catch** (SQLException ex) {
275. Logger.getLogger(MainGUI.**class**.getName()).log(Level.SEVERE, **null**, ex);
276. }
277. }
279. **private** **void** jButton2MouseClicked(java.awt.event.MouseEvent evt) {
280. **try** {
281. AddAnOfficer anOfficer = **new** AddAnOfficer();
282. anOfficer.setVisible(**true**);
283. } **catch** (SQLException ex) {
284. Logger.getLogger(MainGUI.**class**.getName()).log(Level.SEVERE, **null**, ex);
285. }
286. }
288. **private** **void** jButton4MouseClicked(java.awt.event.MouseEvent evt) {
289. **try** {
290. EditAnOfficer anOfficer = **new** EditAnOfficer();
291. anOfficer.setVisible(**true**);
292. } **catch** (SQLException ex) {
293. Logger.getLogger(MainGUI.**class**.getName()).log(Level.SEVERE, **null**, ex);
294. }
295. }
297. **private** **void** jButton6MouseClicked(java.awt.event.MouseEvent evt) {
298. **if** (JOptionPane.showConfirmDialog(**null** ,"Are you sure you want to exit?", "Close this page", JOptionPane.YES\_NO\_OPTION)==0) {
299. System.exit(0);
300. }
301. }
303. **private** **void** jButton5MouseClicked(java.awt.event.MouseEvent evt) {
304. DeleteSite aSite = **new** DeleteSite();
305. aSite.setVisible(**true**);
306. }
308. **private** **void** jButton7MouseClicked(java.awt.event.MouseEvent evt) {
309. **int** row = tblSiteVisits.getSelectedRow();
310. String name = (String)tblSiteVisits.getValueAt(row, 0);
311. MoreDetails more;
312. **try** {
313. more = **new** MoreDetails(name);
314. more.setVisible(**true**);
315. } **catch** (SQLException ex) {
316. Logger.getLogger(MainGUI.**class**.getName()).log(Level.SEVERE, **null**, ex);
317. }
318. }
320. **public** ResultSet getOfficersDetails(String name) **throws** SQLException {
321. DBConnect conn = **new** DBConnect();
322. Connection openConn = conn.openConnection();
323. **try** {
324. Statement stmt = openConn.createStatement();
325. String query = "SELECT \* FROM tblofficers WHERE name = \"" + name + "\" ORDER BY name";
326. ResultSet results = stmt.executeQuery(query);
327. System.out.println(results);
328. **return** results;
330. } **catch** (SQLException e) {
331. System.out.println("Exception = " + e);
332. }
333. //System.out.println(list);
334. **return** **null**;
335. }
337. /\*\*
338. \*
339. \* @return
340. \*/
341. **public** ArrayList<SiteVisit> siteVisits() {
342. ArrayList<SiteVisit> siteVisits = **new** ArrayList<>();
343. DBConnect conn = **new** DBConnect();
344. Connection openConn = conn.openConnection();
345. **try** {
346. Statement stmt = openConn.createStatement();
347. String query = "SELECT \* FROM tblcurrentvisits ORDER BY timestamp ASC";
348. ResultSet results = stmt.executeQuery(query);
349. SiteVisit siteVisit;
350. **while** (results.next()) {
351. siteVisit = **new** SiteVisit(results.getString("name"), results.getString("site"), results.getString("timestamp"));
352. siteVisits.add(siteVisit);
353. }
354. } **catch** (SQLException e) {
355. System.out.println("Exception = " + e);
356. }
357. **return** siteVisits;
358. }
360. **public** **void** alertManager() {
362. }
364. /\*\*
365. \* @param args the command line arguments
366. \*/
367. **public** **static** **void** main(String args[]) {
368. /\* Set the Nimbus look and feel \*/
369. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
370. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
371. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
372. \*/
373. **try** {
374. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
375. **if** ("Nimbus".equals(info.getName())) {
376. javax.swing.UIManager.setLookAndFeel(info.getClassName());
377. **break**;
378. }
379. }
380. } **catch** (ClassNotFoundException ex) {
381. java.util.logging.Logger.getLogger(MainGUI.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
382. } **catch** (InstantiationException ex) {
383. java.util.logging.Logger.getLogger(MainGUI.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
384. } **catch** (IllegalAccessException ex) {
385. java.util.logging.Logger.getLogger(MainGUI.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
386. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
387. java.util.logging.Logger.getLogger(MainGUI.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
388. }
389. //</editor-fold>
391. /\* Create and display the form \*/
392. java.awt.EventQueue.invokeLater(**new** Runnable() {
393. @Override
394. **public** **void** run() {
395. **new** MainGUI().setVisible(**true**);
396. }
397. });
398. }
400. **public** **void** loadTable() **throws** ParseException {
401. ArrayList<SiteVisit> list = siteVisits();
402. DefaultTableModel model = (DefaultTableModel) tblSiteVisits.getModel();
403. model.setRowCount(0);
404. Object[] row = **new** Object[3];
405. **for** (**int** i = 0; i < list.size(); i++) {
406. row[0] = list.get(i).getName();
407. row[1] = list.get(i).getSite();
408. row[2] = list.get(i).getTimestamp();
409. model.addRow(row);
410. Date date1=**new** SimpleDateFormat("hh:mm").parse((String) row[2]);
411. Date date2=**new** Date();
412. date1.setDate(date2.getDate());
413. date1.setMonth(date2.getMonth());
414. date1.setYear(date2.getYear());
415. Date remindTime = (Date) date1.clone();
416. remindTime.setHours(remindTime.getHours()+1);
417. System.out.println("date1: "+date1);
418. System.out.println("date2: " +date2);
419. System.out.println("remindTIme: " +remindTime);
421. //I think it's showing all alerts here as adding an hour to the test time takes it past midnight, but does incrememnt the day.
422. //in production this wont be an issue, but I'll need to testagain tomorrow
424. //date1 is the time they've checked in
425. //date2 is now
426. //remindTime is date1 + 1 hour
427. **if** (date2.after(remindTime)) {
428. //configure alert here
429. JOptionPane.showMessageDialog(**null**,
430. row[0] + " hasn't checked in yet. Please consult a manager before continuing",
431. "Missing Check In",
432. JOptionPane.WARNING\_MESSAGE);
433. System.out.println("Reminder for " + row[0]);
434. }
436. }
437. }
439. // Variables declaration - do not modify
440. **private** javax.swing.JButton jButton1;
441. **private** javax.swing.JButton jButton2;
442. **private** javax.swing.JButton jButton3;
443. **private** javax.swing.JButton jButton4;
444. **private** javax.swing.JButton jButton5;
445. **private** javax.swing.JButton jButton6;
446. **private** javax.swing.JButton jButton7;
447. **private** javax.swing.JLabel jLabel1;
448. **private** javax.swing.JPanel jPanel1;
449. **private** javax.swing.JPanel jPanel2;
450. **private** javax.swing.JPanel jPanel3;
451. **private** javax.swing.JScrollPane jScrollPane2;
452. **private** javax.swing.JTable tblSiteVisits;
453. // End of variables declaration
454. }

#### 4.13.2 AddAManager.java

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** GUI;
8. **import** desktop.app.v2.DBConnect;
9. **import** java.sql.Connection;
10. **import** java.sql.PreparedStatement;
11. **import** java.sql.SQLException;
12. **import** java.util.logging.Level;
13. **import** java.util.logging.Logger;
15. /\*\*
16. \*
17. \* @author zy395907
18. \*/
19. **public** **class** AddAManager **extends** javax.swing.JFrame {
21. /\*\*
22. \* Creates new form AddAnOfficer
23. \*/
24. **public** AddAManager() {
25. initComponents();
26. }
28. /\*\*
29. \* This method is called from within the constructor to initialize the form.
30. \* WARNING: Do NOT modify this code. The content of this method is always
31. \* regenerated by the Form Editor.
32. \*/
33. @SuppressWarnings("unchecked")
34. // <editor-fold defaultstate="collapsed" desc="Generated Code">
35. **private** **void** initComponents() {
37. jLabel1 = **new** javax.swing.JLabel();
38. jLabel2 = **new** javax.swing.JLabel();
39. txtId = **new** javax.swing.JTextField();
40. jLabel4 = **new** javax.swing.JLabel();
41. jLabel5 = **new** javax.swing.JLabel();
42. txtName = **new** javax.swing.JTextField();
43. txtMobileNumber = **new** javax.swing.JTextField();
44. btnSave = **new** javax.swing.JButton();
45. btnReturn = **new** javax.swing.JButton();
46. jLabel6 = **new** javax.swing.JLabel();
47. jLabel7 = **new** javax.swing.JLabel();
48. jLabel8 = **new** javax.swing.JLabel();
49. txtCarReg = **new** javax.swing.JTextField();
50. txtNextOfKin = **new** javax.swing.JTextField();
51. txtNextOfKinPhone = **new** javax.swing.JTextField();
52. jScrollPane1 = **new** javax.swing.JScrollPane();
53. txtAddManagerOutcome = **new** javax.swing.JTextArea();
55. setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE\_ON\_CLOSE);
57. jLabel1.setText("Id Number:");
59. jLabel2.setText("Add A Manager");
61. jLabel4.setText("Name:");
63. jLabel5.setText("Work Mobile:");
65. btnSave.setText("Save");
66. btnSave.addMouseListener(**new** java.awt.event.MouseAdapter() {
67. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
68. btnSaveMouseClicked(evt);
69. }
70. });
72. btnReturn.setText("Return");
73. btnReturn.addMouseListener(**new** java.awt.event.MouseAdapter() {
74. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
75. btnReturnMouseClicked(evt);
76. }
77. });
79. jLabel6.setText("Car Reg:");
81. jLabel7.setText("Next of Kin:");
83. jLabel8.setText("Phone Number:");
85. jScrollPane1.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);
86. jScrollPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);
88. txtAddManagerOutcome.setEditable(**false**);
89. txtAddManagerOutcome.setColumns(20);
90. txtAddManagerOutcome.setRows(5);
91. jScrollPane1.setViewportView(txtAddManagerOutcome);
93. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
94. getContentPane().setLayout(layout);
95. layout.setHorizontalGroup(
96. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
97. .addGroup(layout.createSequentialGroup()
98. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
99. .addGroup(layout.createSequentialGroup()
100. .addGap(176, 176, 176)
101. .addComponent(jLabel2))
102. .addGroup(layout.createSequentialGroup()
103. .addGap(58, 58, 58)
104. .addComponent(jLabel1)
105. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
106. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, 251, javax.swing.GroupLayout.PREFERRED\_SIZE))
107. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
108. .addGap(28, 28, 28)
109. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
110. .addComponent(btnSave, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)
111. .addGroup(layout.createSequentialGroup()
112. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
113. .addComponent(jLabel4)
114. .addComponent(jLabel5)
115. .addComponent(jLabel6)
116. .addComponent(jLabel7)
117. .addComponent(jLabel8))
118. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
119. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)
120. .addComponent(txtName, javax.swing.GroupLayout.DEFAULT\_SIZE, 251, Short.MAX\_VALUE)
121. .addComponent(txtMobileNumber)
122. .addComponent(txtCarReg)
123. .addComponent(txtNextOfKin)
124. .addComponent(txtNextOfKinPhone)))
125. .addComponent(btnReturn, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE))))
126. .addContainerGap(33, Short.MAX\_VALUE))
127. .addComponent(jScrollPane1)
128. );
129. layout.setVerticalGroup(
130. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
131. .addGroup(layout.createSequentialGroup()
132. .addContainerGap()
133. .addComponent(jLabel2)
134. .addGap(41, 41, 41)
135. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
136. .addComponent(jLabel1)
137. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
138. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
139. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
140. .addComponent(jLabel4)
141. .addComponent(txtName, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
142. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
143. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
144. .addComponent(jLabel5)
145. .addComponent(txtMobileNumber, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
146. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
147. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
148. .addComponent(jLabel6)
149. .addComponent(txtCarReg, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
150. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
151. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
152. .addComponent(jLabel7)
153. .addComponent(txtNextOfKin, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
154. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
155. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
156. .addComponent(txtNextOfKinPhone, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
157. .addComponent(jLabel8))
158. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
159. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
160. .addComponent(btnSave)
161. .addComponent(btnReturn))
162. .addGap(20, 20, 20)
163. .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 55, javax.swing.GroupLayout.PREFERRED\_SIZE))
164. );
166. pack();
167. }// </editor-fold>
169. **private** **void** btnReturnMouseClicked(java.awt.event.MouseEvent evt) {
170. **this**.setVisible(**false**);
171. }
173. **private** **void** btnSaveMouseClicked(java.awt.event.MouseEvent evt) {
174. String id;
175. String name;
176. String mobileNumber;
177. String carReg;
178. String nextOfKin;
179. String nextOfKinPhone;

182. id = txtId.getText();
183. name = txtName.getText();
184. mobileNumber = txtMobileNumber.getText();
185. carReg = txtCarReg.getText();
186. nextOfKin = txtNextOfKin.getText();
187. nextOfKinPhone = txtNextOfKinPhone.getText();
188. **if** (!id.equals("") && !name.equals("") && !mobileNumber.equals("") && !carReg.equals("") && !nextOfKin.equals("") && !nextOfKinPhone.equals("")){
189. **try** {
190. DBConnect conn =  **new** DBConnect();
191. Connection openConn = conn.openConnection();
192. PreparedStatement st = openConn.prepareStatement("insert into tblmanagers(id, name, mobileNumber, carReg, nextOfKin, nextOfKinPhone) values (?,?,?,?,?,?)");
193. st.setString(1, id);
194. st.setString(2, name);
195. st.setString(3, mobileNumber);
196. st.setString(4, carReg);
197. st.setString(5, nextOfKin);
198. st.setString(6, nextOfKinPhone);
199. **int** a = st.executeUpdate();
200. **if** (a > 0) {
201. txtAddManagerOutcome.setText((String)name + " has been added, you may now\n"
202. + "add another manager or close this screen by clicking Return");
203. txtId.setText("");
204. txtName.setText("");
205. txtMobileNumber.setText("");
206. txtCarReg.setText("");
207. txtNextOfKin.setText("");
208. txtNextOfKinPhone.setText("");
209. openConn.close();
210. }
211. } **catch** (SQLException ex) {
212. Logger.getLogger(AddAManager.**class**.getName()).log(Level.SEVERE, **null**, ex);
213. txtAddManagerOutcome.setText("There has been an error. \n"
214. + "Please check all of the data and \n"
215. + "try again");
216. }
217. } **else** {
218. txtAddManagerOutcome.setText("There has been an error. \n"
219. + "Please make sure you have completed all fields and \n"
220. + "try again");
221. }

224. }
226. /\*\*
227. \* @param args the command line arguments
228. \*/
229. **public** **static** **void** main(String args[]) {
230. /\* Set the Nimbus look and feel \*/
231. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
232. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
233. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
234. \*/
235. **try** {
236. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
237. **if** ("Nimbus".equals(info.getName())) {
238. javax.swing.UIManager.setLookAndFeel(info.getClassName());
239. **break**;
240. }
241. }
242. } **catch** (ClassNotFoundException ex) {
243. java.util.logging.Logger.getLogger(AddAManager.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
244. } **catch** (InstantiationException ex) {
245. java.util.logging.Logger.getLogger(AddAManager.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
246. } **catch** (IllegalAccessException ex) {
247. java.util.logging.Logger.getLogger(AddAManager.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
248. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
249. java.util.logging.Logger.getLogger(AddAManager.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
250. }
251. //</editor-fold>
252. //</editor-fold>
253. //</editor-fold>
254. //</editor-fold>
256. /\* Create and display the form \*/
257. java.awt.EventQueue.invokeLater(**new** Runnable() {
258. **public** **void** run() {
259. **new** AddAManager().setVisible(**true**);
260. }
261. });
262. }

265. // Variables declaration - do not modify
266. **private** javax.swing.JButton btnReturn;
267. **private** javax.swing.JButton btnSave;
268. **private** javax.swing.JLabel jLabel1;
269. **private** javax.swing.JLabel jLabel2;
270. **private** javax.swing.JLabel jLabel4;
271. **private** javax.swing.JLabel jLabel5;
272. **private** javax.swing.JLabel jLabel6;
273. **private** javax.swing.JLabel jLabel7;
274. **private** javax.swing.JLabel jLabel8;
275. **private** javax.swing.JScrollPane jScrollPane1;
276. **private** javax.swing.JTextArea txtAddManagerOutcome;
277. **private** javax.swing.JTextField txtCarReg;
278. **private** javax.swing.JTextField txtId;
279. **private** javax.swing.JTextField txtMobileNumber;
280. **private** javax.swing.JTextField txtName;
281. **private** javax.swing.JTextField txtNextOfKin;
282. **private** javax.swing.JTextField txtNextOfKinPhone;
283. // End of variables declaration
284. }

#### 4.13.3 AddAnOfficer.java

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** GUI;
8. **import** desktop.app.v2.DBConnect;
9. **import** java.sql.Connection;
10. **import** java.sql.PreparedStatement;
11. **import** java.sql.ResultSet;
12. **import** java.sql.SQLException;
13. **import** java.sql.Statement;
14. **import** java.util.ArrayList;
15. **import** java.util.List;
16. **import** java.util.logging.Level;
17. **import** java.util.logging.Logger;
19. /\*\*
20. \*
21. \* @author zy395907
22. \*/
23. **public** **class** AddAnOfficer **extends** javax.swing.JFrame {
25. /\*\*
26. \* Creates new form AddAnOfficer
27. \* @throws java.sql.SQLException
28. \*/
29. **public** AddAnOfficer() **throws** SQLException {
30. initComponents();
31. cmboManager.removeAllItems();
32. List<String> managers = **this**.getManagers();
33. managers.forEach((manager) -> {
34. cmboManager.addItem(manager);
35. });
37. }
39. /\*\*
40. \* This method is called from within the constructor to initialize the form.
41. \* WARNING: Do NOT modify this code. The content of this method is always
42. \* regenerated by the Form Editor.
43. \*/
44. @SuppressWarnings("unchecked")
45. // <editor-fold defaultstate="collapsed" desc="Generated Code">
46. **private** **void** initComponents() {
48. jLabel1 = **new** javax.swing.JLabel();
49. jLabel2 = **new** javax.swing.JLabel();
50. txtId = **new** javax.swing.JTextField();
51. jLabel4 = **new** javax.swing.JLabel();
52. jLabel5 = **new** javax.swing.JLabel();
53. txtName = **new** javax.swing.JTextField();
54. txtMobileNumber = **new** javax.swing.JTextField();
55. btnSave = **new** javax.swing.JButton();
56. btnReturn = **new** javax.swing.JButton();
57. jLabel6 = **new** javax.swing.JLabel();
58. jLabel7 = **new** javax.swing.JLabel();
59. jLabel8 = **new** javax.swing.JLabel();
60. txtCarReg = **new** javax.swing.JTextField();
61. txtNextOfKin = **new** javax.swing.JTextField();
62. txtNextOfKinPhone = **new** javax.swing.JTextField();
63. jScrollPane1 = **new** javax.swing.JScrollPane();
64. txtAddManagerOutcome = **new** javax.swing.JTextArea();
65. cmboManager = **new** javax.swing.JComboBox<>();
66. jLabel9 = **new** javax.swing.JLabel();
68. setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE\_ON\_CLOSE);
70. jLabel1.setText("Id Number:");
72. jLabel2.setText("Add an Officer");
74. jLabel4.setText("Name:");
76. jLabel5.setText("Work Mobile:");
78. btnSave.setText("Save");
79. btnSave.addMouseListener(**new** java.awt.event.MouseAdapter() {
80. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
81. btnSaveMouseClicked(evt);
82. }
83. });
85. btnReturn.setText("Return");
86. btnReturn.addMouseListener(**new** java.awt.event.MouseAdapter() {
87. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
88. btnReturnMouseClicked(evt);
89. }
90. });
92. jLabel6.setText("Car Reg:");
94. jLabel7.setText("Next of Kin:");
96. jLabel8.setText("Phone Number:");
98. jScrollPane1.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);
99. jScrollPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);
101. txtAddManagerOutcome.setEditable(**false**);
102. txtAddManagerOutcome.setColumns(20);
103. txtAddManagerOutcome.setRows(5);
104. jScrollPane1.setViewportView(txtAddManagerOutcome);
106. cmboManager.setModel(**new** javax.swing.DefaultComboBoxModel<>(**new** String[] { "Item 1", "Item 2", "Item 3", "Item 4" }));
108. jLabel9.setText("Manager:");
110. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
111. getContentPane().setLayout(layout);
112. layout.setHorizontalGroup(
113. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
114. .addGroup(layout.createSequentialGroup()
115. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
116. .addGroup(layout.createSequentialGroup()
117. .addGap(176, 176, 176)
118. .addComponent(jLabel2))
119. .addGroup(layout.createSequentialGroup()
120. .addGap(58, 58, 58)
121. .addComponent(jLabel1)
122. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
123. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, 251, javax.swing.GroupLayout.PREFERRED\_SIZE))
124. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
125. .addGap(28, 28, 28)
126. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
127. .addComponent(btnSave, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)
128. .addComponent(btnReturn, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)
129. .addGroup(layout.createSequentialGroup()
130. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
131. .addComponent(jLabel4)
132. .addComponent(jLabel5)
133. .addComponent(jLabel6)
134. .addComponent(jLabel7)
135. .addComponent(jLabel8)
136. .addComponent(jLabel9))
137. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
138. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)
139. .addComponent(cmboManager, 0, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
140. .addComponent(txtName, javax.swing.GroupLayout.DEFAULT\_SIZE, 251, Short.MAX\_VALUE)
141. .addComponent(txtMobileNumber)
142. .addComponent(txtCarReg)
143. .addComponent(txtNextOfKin)
144. .addComponent(txtNextOfKinPhone))))))
145. .addContainerGap(33, Short.MAX\_VALUE))
146. .addComponent(jScrollPane1)
147. );
148. layout.setVerticalGroup(
149. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
150. .addGroup(layout.createSequentialGroup()
151. .addContainerGap()
152. .addComponent(jLabel2)
153. .addGap(41, 41, 41)
154. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
155. .addComponent(jLabel1)
156. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
157. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
158. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
159. .addComponent(jLabel4)
160. .addComponent(txtName, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
161. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
162. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
163. .addComponent(jLabel5)
164. .addComponent(txtMobileNumber, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
165. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
166. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
167. .addComponent(jLabel6)
168. .addComponent(txtCarReg, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
169. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
170. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
171. .addComponent(jLabel7)
172. .addComponent(txtNextOfKin, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
173. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
174. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
175. .addComponent(txtNextOfKinPhone, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
176. .addComponent(jLabel8))
177. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
178. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
179. .addComponent(cmboManager, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
180. .addComponent(jLabel9))
181. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 26, Short.MAX\_VALUE)
182. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
183. .addComponent(btnSave)
184. .addComponent(btnReturn))
185. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
186. .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 55, javax.swing.GroupLayout.PREFERRED\_SIZE))
187. );
189. pack();
190. }// </editor-fold>
192. **private** **void** btnReturnMouseClicked(java.awt.event.MouseEvent evt) {
193. **this**.setVisible(**false**);
194. }
196. **private** **void** btnSaveMouseClicked(java.awt.event.MouseEvent evt) {
197. **try** {
198. String id;
199. String name;
200. String mobileNumber;
201. String carReg;
202. String nextOfKin;
203. String nextOfKinPhone;
204. String manager;

207. id = txtId.getText();
208. name = txtName.getText();
209. mobileNumber = txtMobileNumber.getText();
210. carReg = txtCarReg.getText();
211. nextOfKin = txtNextOfKin.getText();
212. nextOfKinPhone = txtNextOfKinPhone.getText();
213. manager = (String)cmboManager.getSelectedItem();
214. DBConnect conn =  **new** DBConnect();
215. Connection openConn = conn.openConnection();
216. PreparedStatement st = openConn.prepareStatement("insert into tblofficers(id, name, mobileNumber, carReg, nextOfKin, nextOfKinPhone, manager) values (?,?,?,?,?,?,?)");
217. st.setString(1, id);
218. st.setString(2, name);
219. st.setString(3, mobileNumber);
220. st.setString(4, carReg);
221. st.setString(5, nextOfKin);
222. st.setString(6, nextOfKinPhone);
223. st.setString(7, manager);
224. **int** a = st.executeUpdate();
225. **if** (a>0)
226. {
227. txtAddManagerOutcome.setText((String)name + " has been added, you may \n"
228. + "now close this screen by clicking Return");
229. txtId.setText("");
230. txtName.setText("");
231. txtMobileNumber.setText("");
232. txtCarReg.setText("");
233. txtNextOfKin.setText("");
234. txtNextOfKinPhone.setText("");
235. cmboManager.setSelectedIndex(-1);
236. openConn.close();
237. }
238. } **catch** (SQLException ex) {
239. txtAddManagerOutcome.setText("There has been an error. \n"
240. + "Please make sure you have completed all fields and \n"
241. + "try again");
242. Logger.getLogger(AddAnOfficer.**class**.getName()).log(Level.SEVERE, **null**, ex);
243. }
245. }
247. **private** List<String> getManagers() **throws** SQLException {
248. List<String> list = **new** ArrayList();
249. DBConnect conn =  **new** DBConnect();
250. Connection openConn = conn.openConnection();
251. **try** {
252. Statement stmt = openConn.createStatement();
253. String query = "SELECT name FROM tblmanagers ORDER BY name";
254. ResultSet results = stmt.executeQuery(query);
255. list.add(**null**);
256. **while** (results.next()) {
257. list.add(results.getString("name"));
258. }
259. openConn.close();
260. } **catch** (SQLException e) {
261. System.out.println("Exception = " + e);
262. }
263. **return** list;
264. }

267. /\*\*
268. \* @param args the command line arguments
269. \*/
270. **public** **static** **void** main(String args[]) {
271. /\* Set the Nimbus look and feel \*/
272. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
273. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
274. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
275. \*/
276. **try** {
277. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
278. **if** ("Nimbus".equals(info.getName())) {
279. javax.swing.UIManager.setLookAndFeel(info.getClassName());
280. **break**;
281. }
282. }
283. } **catch** (ClassNotFoundException ex) {
284. java.util.logging.Logger.getLogger(AddAnOfficer.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
285. } **catch** (InstantiationException ex) {
286. java.util.logging.Logger.getLogger(AddAnOfficer.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
287. } **catch** (IllegalAccessException ex) {
288. java.util.logging.Logger.getLogger(AddAnOfficer.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
289. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
290. java.util.logging.Logger.getLogger(AddAnOfficer.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
291. }
292. //</editor-fold>
293. //</editor-fold>
294. //</editor-fold>
295. //</editor-fold>
296. //</editor-fold>
297. //</editor-fold>
298. //</editor-fold>
299. //</editor-fold>
301. /\* Create and display the form \*/
302. java.awt.EventQueue.invokeLater(**new** Runnable() {
303. @Override
304. **public** **void** run() {
305. **try** {
306. **new** AddAnOfficer().setVisible(**true**);
307. } **catch** (SQLException ex) {
308. Logger.getLogger(AddAnOfficer.**class**.getName()).log(Level.SEVERE, **null**, ex);
309. }
310. }
311. });
312. }
314. // Variables declaration - do not modify
315. **private** javax.swing.JButton btnReturn;
316. **private** javax.swing.JButton btnSave;
317. **private** javax.swing.JComboBox<String> cmboManager;
318. **private** javax.swing.JLabel jLabel1;
319. **private** javax.swing.JLabel jLabel2;
320. **private** javax.swing.JLabel jLabel4;
321. **private** javax.swing.JLabel jLabel5;
322. **private** javax.swing.JLabel jLabel6;
323. **private** javax.swing.JLabel jLabel7;
324. **private** javax.swing.JLabel jLabel8;
325. **private** javax.swing.JLabel jLabel9;
326. **private** javax.swing.JScrollPane jScrollPane1;
327. **private** javax.swing.JTextArea txtAddManagerOutcome;
328. **private** javax.swing.JTextField txtCarReg;
329. **private** javax.swing.JTextField txtId;
330. **private** javax.swing.JTextField txtMobileNumber;
331. **private** javax.swing.JTextField txtName;
332. **private** javax.swing.JTextField txtNextOfKin;
333. **private** javax.swing.JTextField txtNextOfKinPhone;
334. // End of variables declaration
335. }

#### 4.13.4 EditAManager.java

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** GUI;
8. **import** desktop.app.v2.DBConnect;
9. **import** java.sql.\*;
10. **import** java.util.ArrayList;
11. **import** java.util.List;
12. **import** java.util.logging.Level;
13. **import** java.util.logging.Logger;
14. **import** javax.swing.JOptionPane;
16. /\*\*
17. \*
18. \* @author zy395907
19. \*/
20. **public** **class** EditAManager **extends** javax.swing.JFrame {
22. /\*\*
23. \* Creates new form AddAnOfficer
24. \*
25. \* @throws java.sql.SQLException
26. \*/
27. **public** EditAManager() **throws** SQLException {
28. initComponents();
30. }
32. /\*\*
33. \* This method is called from within the constructor to initialize the form.
34. \* WARNING: Do NOT modify this code. The content of this method is always
35. \* regenerated by the Form Editor.
36. \*/
37. @SuppressWarnings("unchecked")
38. // <editor-fold defaultstate="collapsed" desc="Generated Code">
39. **private** **void** initComponents() {
41. jLabel1 = **new** javax.swing.JLabel();
42. jLabel2 = **new** javax.swing.JLabel();
43. txtId = **new** javax.swing.JTextField();
44. jLabel4 = **new** javax.swing.JLabel();
45. jLabel5 = **new** javax.swing.JLabel();
46. txtMobileNumber = **new** javax.swing.JTextField();
47. btnReturn = **new** javax.swing.JButton();
48. jLabel6 = **new** javax.swing.JLabel();
49. jLabel7 = **new** javax.swing.JLabel();
50. jLabel8 = **new** javax.swing.JLabel();
51. txtCarReg = **new** javax.swing.JTextField();
52. txtNextOfKin = **new** javax.swing.JTextField();
53. txtNextOfKinPhone = **new** javax.swing.JTextField();
54. jScrollPane1 = **new** javax.swing.JScrollPane();
55. txtEditManagerOutcome = **new** javax.swing.JTextArea();
56. cmboManagerName = **new** javax.swing.JComboBox<>();
57. btnDelete = **new** javax.swing.JButton();
58. btnUpdate = **new** javax.swing.JButton();
59. btnLoad = **new** javax.swing.JButton();
61. setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE\_ON\_CLOSE);
63. jLabel1.setText("Id Number:");
65. jLabel2.setText("Edit A Manager");
67. txtId.setEditable(**false**);
69. jLabel4.setText("Name:");
71. jLabel5.setText("Work Mobile:");
73. btnReturn.setText("Return");
74. btnReturn.addMouseListener(**new** java.awt.event.MouseAdapter() {
75. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
76. btnReturnMouseClicked(evt);
77. }
78. });
80. jLabel6.setText("Car Reg:");
82. jLabel7.setText("Next of Kin:");
84. jLabel8.setText("Phone Number:");
86. jScrollPane1.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);
87. jScrollPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);
89. txtEditManagerOutcome.setEditable(**false**);
90. txtEditManagerOutcome.setColumns(20);
91. txtEditManagerOutcome.setRows(5);
92. jScrollPane1.setViewportView(txtEditManagerOutcome);
94. cmboManagerName.addMouseListener(**new** java.awt.event.MouseAdapter() {
95. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
96. cmboManagerNameMouseClicked(evt);
97. }
98. });
100. btnDelete.setText("Delete");
101. btnDelete.addMouseListener(**new** java.awt.event.MouseAdapter() {
102. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
103. btnDeleteMouseClicked(evt);
104. }
105. });
107. btnUpdate.setText("Update");
108. btnUpdate.addMouseListener(**new** java.awt.event.MouseAdapter() {
109. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
110. btnUpdateMouseClicked(evt);
111. }
112. });
114. btnLoad.setText("Load");
115. btnLoad.addMouseListener(**new** java.awt.event.MouseAdapter() {
116. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
117. btnLoadMouseClicked(evt);
118. }
119. });
121. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
122. getContentPane().setLayout(layout);
123. layout.setHorizontalGroup(
124. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
125. .addGroup(layout.createSequentialGroup()
126. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
127. .addGroup(layout.createSequentialGroup()
128. .addGap(58, 58, 58)
129. .addComponent(jLabel1)
130. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
131. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, 251, javax.swing.GroupLayout.PREFERRED\_SIZE))
132. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
133. .addGap(21, 21, 21)
134. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
135. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
136. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
137. .addComponent(jLabel4)
138. .addComponent(jLabel5)
139. .addComponent(jLabel6)
140. .addComponent(jLabel7)
141. .addComponent(jLabel8))
142. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
143. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)
144. .addComponent(txtMobileNumber, javax.swing.GroupLayout.DEFAULT\_SIZE, 251, Short.MAX\_VALUE)
145. .addComponent(txtCarReg)
146. .addComponent(txtNextOfKin)
147. .addComponent(txtNextOfKinPhone)
148. .addComponent(cmboManagerName, 0, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)))
149. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
150. .addComponent(btnUpdate, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)
151. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
152. .addComponent(btnDelete, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)
153. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
154. .addComponent(btnReturn, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE))))
155. .addGroup(layout.createSequentialGroup()
156. .addGap(176, 176, 176)
157. .addComponent(jLabel2)))
158. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))
159. .addComponent(jScrollPane1)
160. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
161. .addGap(0, 172, Short.MAX\_VALUE)
162. .addComponent(btnLoad)
163. .addContainerGap(173, Short.MAX\_VALUE))
164. );
165. layout.setVerticalGroup(
166. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
167. .addGroup(layout.createSequentialGroup()
168. .addContainerGap()
169. .addComponent(jLabel2)
170. .addGap(41, 41, 41)
171. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
172. .addComponent(jLabel1)
173. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
174. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
175. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
176. .addComponent(jLabel4)
177. .addComponent(cmboManagerName, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
178. .addGap(3, 3, 3)
179. .addComponent(btnLoad, javax.swing.GroupLayout.PREFERRED\_SIZE, 17, javax.swing.GroupLayout.PREFERRED\_SIZE)
180. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
181. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
182. .addComponent(jLabel5)
183. .addComponent(txtMobileNumber, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
184. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
185. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
186. .addComponent(jLabel6)
187. .addComponent(txtCarReg, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
188. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
189. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
190. .addComponent(jLabel7)
191. .addComponent(txtNextOfKin, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
192. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
193. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
194. .addComponent(txtNextOfKinPhone, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
195. .addComponent(jLabel8))
196. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 12, Short.MAX\_VALUE)
197. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
198. .addComponent(btnReturn)
199. .addComponent(btnDelete)
200. .addComponent(btnUpdate))
201. .addGap(20, 20, 20)
202. .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 55, javax.swing.GroupLayout.PREFERRED\_SIZE))
203. );
205. pack();
206. }// </editor-fold>
208. /\*\*
209. \* Return to the main menu when clicking return
210. \* @param evt
211. \*/
212. **private** **void** btnReturnMouseClicked(java.awt.event.MouseEvent evt) {
213. **this**.setVisible(**false**);
214. }
216. /\*\*
217. \* Delete the selected manager and display an appropriate outcome message, before finally setting all the values to blank
218. \* @param evt
219. \*/
220. **private** **void** btnDeleteMouseClicked(java.awt.event.MouseEvent evt) {
221. **if** (cmboManagerName.getSelectedItem() == **null**) {
222. txtEditManagerOutcome.setText("Please select a manager before trying to delete");
223. } **else** {
224. String id = txtId.getText();
225. String name = (String)cmboManagerName.getSelectedItem();
226. **try** {
227. **this**.deleteManager(id);
228. **if** (JOptionPane.showConfirmDialog(**null**, "Are you sure you want to delete \"" + name + "\"?", "Delete", JOptionPane.YES\_NO\_OPTION) == 0) {
229. txtEditManagerOutcome.setText((String)name + " has been deleted, you may now\n"
230. + "edit/delete another manager or close this screen by clicking Return");
231. txtId.setText("");
232. cmboManagerName.setSelectedIndex(-1);
233. txtMobileNumber.setText("");
234. txtCarReg.setText("");
235. txtNextOfKin.setText("");
236. txtNextOfKinPhone.setText("");
237. }
238. } **catch** (SQLException ex) {
239. Logger.getLogger(EditAManager.**class**.getName()).log(Level.SEVERE, **null**, ex);
240. }
241. }
242. }
244. /\*\*
245. \* Updates the selected manager with the details provided
246. \* @param evt
247. \*/
248. **private** **void** btnUpdateMouseClicked(java.awt.event.MouseEvent evt) {
249. **if** (cmboManagerName.getSelectedItem() == **null**) {
250. txtEditManagerOutcome.setText("Please select a manager before trying to update");
251. } **else** {
252. String id = txtId.getText();
253. String name = (String)cmboManagerName.getSelectedItem();
254. **try** {
255. **this**.updateManager(id);
256. txtEditManagerOutcome.setText((String)name + " has been updated, you may now\n"
257. + "edit/delete another manager or close this screen by clicking Return");
258. txtId.setText("");
259. cmboManagerName.setSelectedIndex(-1);
260. txtMobileNumber.setText("");
261. txtCarReg.setText("");
262. txtNextOfKin.setText("");
263. txtNextOfKinPhone.setText("");
264. } **catch** (SQLException ex) {
265. Logger.getLogger(EditAManager.**class**.getName()).log(Level.SEVERE, **null**, ex);
266. }
267. }
268. }
270. /\*\*
271. \* Display the list of managers when clicking on the manager drop down menu
272. \* @param evt
273. \*/
274. **private** **void** cmboManagerNameMouseClicked(java.awt.event.MouseEvent evt) {
275. cmboManagerName.removeAllItems();
276. List<String> managers = **null**;
277. **try** {
278. managers = **this**.getManagers();
279. } **catch** (SQLException ex) {
280. Logger.getLogger(EditAManager.**class**.getName()).log(Level.SEVERE, **null**, ex);
281. }
282. managers.forEach((manager) -> {
283. cmboManagerName.addItem(manager);
284. });
285. }
287. /\*\*
288. \* Load the current details for the selected manager
289. \* @param evt
290. \*/
291. **private** **void** btnLoadMouseClicked(java.awt.event.MouseEvent evt) {
292. **if** (cmboManagerName.getSelectedItem() == **null**) {
293. txtEditManagerOutcome.setText("Please select a manager before clicking load");
294. } **else** {
295. String name = (String)cmboManagerName.getSelectedItem();
297. **try** {
298. ResultSet results = getManagersDetails(name);
299. **while** (results.next()){
300. txtId.setText(results.getString(1));
301. txtMobileNumber.setText(results.getString(3));
302. txtCarReg.setText(results.getString(4));
303. txtNextOfKin.setText(results.getString(5));
304. txtNextOfKinPhone.setText(results.getString(6));
305. }
306. } **catch** (SQLException ex) {
307. Logger.getLogger(EditAManager.**class**.getName()).log(Level.SEVERE, **null**, ex);
308. }
309. }
310. }
312. /\*\*
313. \* Helper method for btnLoadMouseClicked
314. \* @param name
315. \* @return ResultSet of selected manager details
316. \* @throws SQLException
317. \*/
318. **public** ResultSet getManagersDetails(String name) **throws** SQLException {
319. DBConnect conn = **new** DBConnect();
320. Connection openConn = conn.openConnection();
321. **try** {
322. Statement stmt = openConn.createStatement();
323. String query = "SELECT \* FROM tblmanagers WHERE name = \"" + name + "\" ORDER BY name";
324. ResultSet results = stmt.executeQuery(query);
325. System.out.println(results);
326. **return** results;
328. } **catch** (SQLException e) {
329. System.out.println("Exception = " + e);
330. }
331. //System.out.println(list);
332. **return** **null**;
333. }
335. /\*\*
336. \* Helper method for cmboManagerNameMouseClicked
337. \* @return A list of all managers in tblmanagers
338. \* @throws SQLException
339. \*/
340. **private** List<String> getManagers() **throws** SQLException {
341. List<String> list = **new** ArrayList();
342. DBConnect conn = **new** DBConnect();
343. Connection openConn = conn.openConnection();
344. **try** {
345. Statement stmt = openConn.createStatement();
346. String query = "SELECT name FROM tblmanagers ORDER BY name";
347. ResultSet results = stmt.executeQuery(query);
348. list.add(**null**);
349. **while** (results.next()) {
350. list.add(results.getString("name"));
351. }
353. } **catch** (SQLException e) {
354. System.out.println("Exception = " + e);
355. }
357. **return** list;
358. }
360. /\*\*
361. \* Helper method for btnDeleteMouseClicked, deletes the managers with the specified id.
362. \* @param managerId
363. \* @throws SQLException
364. \*/
365. **public** **void** deleteManager(String managerId) **throws** SQLException {
366. DBConnect conn = **new** DBConnect();
367. Connection openConn = conn.openConnection();
368. PreparedStatement st = openConn.prepareStatement("DELETE FROM tblmanagers WHERE id = \"" + managerId + "\"");
369. st.executeUpdate();
371. }
373. /\*\*
374. \* Helper method for btnUpdateMouseClicked, updates all details for the specified manager
375. \* @param managerId
376. \* @throws SQLException
377. \*/
378. **public** **void** updateManager (String managerId) **throws** SQLException {
379. String name = (String)cmboManagerName.getSelectedItem();
380. String id = txtId.getText();
381. String mobileNumber = txtMobileNumber.getText();
382. String carReg = txtCarReg.getText();
383. String nextOfKin = txtNextOfKin.getText();
384. String nextOfKinPhone = txtNextOfKinPhone.getText();
386. DBConnect conn = **new** DBConnect();
387. Connection openConn = conn.openConnection();
388. PreparedStatement st = openConn.prepareStatement("UPDATE tblmanagers SET name = \"" + name +
389. "\", mobileNumber = \"" + mobileNumber + "\", carReg = \"" + carReg +
390. "\", nextOfKin = \"" + nextOfKin + "\", nextOfKinPhone = \"" + nextOfKinPhone +
391. "\" WHERE id = \"" + id + "\" ");
392. st.executeUpdate();
393. }
395. /\*\*
396. \* @param args the command line arguments
397. \*/
398. **public** **static** **void** main(String args[]) {
399. /\* Set the Nimbus look and feel \*/
400. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
401. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
402. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
403. \*/
404. **try** {
405. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
406. **if** ("Nimbus".equals(info.getName())) {
407. javax.swing.UIManager.setLookAndFeel(info.getClassName());
408. **break**;
409. }
410. }
411. } **catch** (ClassNotFoundException ex) {
412. java.util.logging.Logger.getLogger(EditAManager.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
413. } **catch** (InstantiationException ex) {
414. java.util.logging.Logger.getLogger(EditAManager.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
415. } **catch** (IllegalAccessException ex) {
416. java.util.logging.Logger.getLogger(EditAManager.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
417. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
418. java.util.logging.Logger.getLogger(EditAManager.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
419. }
420. //</editor-fold>
421. //</editor-fold>
422. //</editor-fold>
423. //</editor-fold>
424. //</editor-fold>
425. //</editor-fold>
426. //</editor-fold>
427. //</editor-fold>
429. /\* Create and display the form \*/
430. java.awt.EventQueue.invokeLater(() -> {
431. **try** {
432. **new** EditAManager().setVisible(**true**);
433. } **catch** (SQLException ex) {
434. Logger.getLogger(EditAManager.**class**.getName()).log(Level.SEVERE, **null**, ex);
435. }
436. });
437. }
439. // Variables declaration - do not modify
440. **private** javax.swing.JButton btnDelete;
441. **private** javax.swing.JButton btnLoad;
442. **private** javax.swing.JButton btnReturn;
443. **private** javax.swing.JButton btnUpdate;
444. **private** javax.swing.JComboBox<String> cmboManagerName;
445. **private** javax.swing.JLabel jLabel1;
446. **private** javax.swing.JLabel jLabel2;
447. **private** javax.swing.JLabel jLabel4;
448. **private** javax.swing.JLabel jLabel5;
449. **private** javax.swing.JLabel jLabel6;
450. **private** javax.swing.JLabel jLabel7;
451. **private** javax.swing.JLabel jLabel8;
452. **private** javax.swing.JScrollPane jScrollPane1;
453. **private** javax.swing.JTextField txtCarReg;
454. **private** javax.swing.JTextArea txtEditManagerOutcome;
455. **private** javax.swing.JTextField txtId;
456. **private** javax.swing.JTextField txtMobileNumber;
457. **private** javax.swing.JTextField txtNextOfKin;
458. **private** javax.swing.JTextField txtNextOfKinPhone;
459. // End of variables declaration
460. }

#### 4.13.5 EditAnOfficer.java

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** GUI;
8. **import** desktop.app.v2.DBConnect;
9. **import** java.sql.\*;
10. **import** java.util.ArrayList;
11. **import** java.util.List;
12. **import** java.util.logging.Level;
13. **import** java.util.logging.Logger;
14. **import** javax.swing.JOptionPane;
16. /\*\*
17. \*
18. \* @author zy395907
19. \*/
20. **public** **class** EditAnOfficer **extends** javax.swing.JFrame {
22. /\*\*
23. \* Creates new form EditAnOfficer
24. \*
25. \* @throws java.sql.SQLException
26. \*/
27. **public** EditAnOfficer() **throws** SQLException {
28. initComponents();
30. }
32. /\*\*
33. \* This method is called from within the constructor to initialize the form.
34. \* WARNING: Do NOT modify this code. The content of this method is always
35. \* regenerated by the Form Editor.
36. \*/
37. @SuppressWarnings("unchecked")
38. // <editor-fold defaultstate="collapsed" desc="Generated Code">
39. **private** **void** initComponents() {
41. jLabel1 = **new** javax.swing.JLabel();
42. jLabel2 = **new** javax.swing.JLabel();
43. txtId = **new** javax.swing.JTextField();
44. jLabel4 = **new** javax.swing.JLabel();
45. jLabel5 = **new** javax.swing.JLabel();
46. txtMobileNumber = **new** javax.swing.JTextField();
47. btnReturn = **new** javax.swing.JButton();
48. jLabel6 = **new** javax.swing.JLabel();
49. jLabel7 = **new** javax.swing.JLabel();
50. jLabel8 = **new** javax.swing.JLabel();
51. txtCarReg = **new** javax.swing.JTextField();
52. txtNextOfKin = **new** javax.swing.JTextField();
53. txtNextOfKinPhone = **new** javax.swing.JTextField();
54. jScrollPane1 = **new** javax.swing.JScrollPane();
55. txtEditOfficerOutcome = **new** javax.swing.JTextArea();
56. cmboOfficerName = **new** javax.swing.JComboBox<>();
57. btnDelete = **new** javax.swing.JButton();
58. btnUpdate = **new** javax.swing.JButton();
59. btnLoad = **new** javax.swing.JButton();
60. cmboManagerName = **new** javax.swing.JComboBox<>();
61. jLabel3 = **new** javax.swing.JLabel();
63. setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE\_ON\_CLOSE);
65. jLabel1.setText("Id Number:");
67. jLabel2.setText("Edit An Officer");
69. txtId.setEditable(**false**);
71. jLabel4.setText("Name:");
73. jLabel5.setText("Work Mobile:");
75. btnReturn.setText("Return");
76. btnReturn.addMouseListener(**new** java.awt.event.MouseAdapter() {
77. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
78. btnReturnMouseClicked(evt);
79. }
80. });
82. jLabel6.setText("Car Reg:");
84. jLabel7.setText("Next of Kin:");
86. jLabel8.setText("Phone Number:");
88. jScrollPane1.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);
89. jScrollPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);
91. txtEditOfficerOutcome.setEditable(**false**);
92. txtEditOfficerOutcome.setColumns(20);
93. txtEditOfficerOutcome.setRows(5);
94. jScrollPane1.setViewportView(txtEditOfficerOutcome);
96. cmboOfficerName.addMouseListener(**new** java.awt.event.MouseAdapter() {
97. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
98. cmboOfficerNameMouseClicked(evt);
99. }
100. });
102. btnDelete.setText("Delete");
103. btnDelete.addMouseListener(**new** java.awt.event.MouseAdapter() {
104. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
105. btnDeleteMouseClicked(evt);
106. }
107. });
109. btnUpdate.setText("Update");
110. btnUpdate.addMouseListener(**new** java.awt.event.MouseAdapter() {
111. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
112. btnUpdateMouseClicked(evt);
113. }
114. });
116. btnLoad.setText("Load");
117. btnLoad.addMouseListener(**new** java.awt.event.MouseAdapter() {
118. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
119. btnLoadMouseClicked(evt);
120. }
121. });
123. jLabel3.setText("Manager:");
125. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
126. getContentPane().setLayout(layout);
127. layout.setHorizontalGroup(
128. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
129. .addGroup(layout.createSequentialGroup()
130. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
131. .addGroup(layout.createSequentialGroup()
132. .addGap(58, 58, 58)
133. .addComponent(jLabel1)
134. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
135. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, 251, javax.swing.GroupLayout.PREFERRED\_SIZE))
136. .addGroup(layout.createSequentialGroup()
137. .addGap(176, 176, 176)
138. .addComponent(jLabel2))
139. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
140. .addGap(38, 38, 38)
141. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
142. .addComponent(jLabel4)
143. .addComponent(jLabel5)
144. .addComponent(jLabel6)
145. .addComponent(jLabel7)
146. .addComponent(jLabel8)
147. .addComponent(jLabel3))
148. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
149. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)
150. .addComponent(cmboManagerName, 0, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
151. .addComponent(txtMobileNumber, javax.swing.GroupLayout.DEFAULT\_SIZE, 251, Short.MAX\_VALUE)
152. .addComponent(txtCarReg)
153. .addComponent(txtNextOfKin)
154. .addComponent(txtNextOfKinPhone)
155. .addComponent(cmboOfficerName, 0, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))))
156. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))
157. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
158. .addGap(0, 172, Short.MAX\_VALUE)
159. .addComponent(btnLoad)
160. .addContainerGap(173, Short.MAX\_VALUE))
161. .addGroup(layout.createSequentialGroup()
162. .addGap(21, 21, 21)
163. .addComponent(btnUpdate, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)
164. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
165. .addComponent(btnDelete, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)
166. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
167. .addComponent(btnReturn, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)
168. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))
169. .addComponent(jScrollPane1)
170. );
171. layout.setVerticalGroup(
172. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
173. .addGroup(layout.createSequentialGroup()
174. .addContainerGap()
175. .addComponent(jLabel2)
176. .addGap(41, 41, 41)
177. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
178. .addComponent(jLabel1)
179. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
180. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
181. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
182. .addComponent(jLabel4)
183. .addComponent(cmboOfficerName, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
184. .addGap(3, 3, 3)
185. .addComponent(btnLoad, javax.swing.GroupLayout.PREFERRED\_SIZE, 17, javax.swing.GroupLayout.PREFERRED\_SIZE)
186. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
187. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
188. .addComponent(jLabel5)
189. .addComponent(txtMobileNumber, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
190. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
191. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
192. .addComponent(jLabel6)
193. .addComponent(txtCarReg, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
194. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
195. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
196. .addComponent(jLabel7)
197. .addComponent(txtNextOfKin, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
198. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
199. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
200. .addComponent(txtNextOfKinPhone, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
201. .addComponent(jLabel8))
202. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
203. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
204. .addComponent(cmboManagerName, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
205. .addComponent(jLabel3))
206. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 39, Short.MAX\_VALUE)
207. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
208. .addComponent(btnReturn)
209. .addComponent(btnDelete)
210. .addComponent(btnUpdate))
211. .addGap(20, 20, 20)
212. .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 55, javax.swing.GroupLayout.PREFERRED\_SIZE)
213. .addGap(0, 0, 0))
214. );
216. pack();
217. }// </editor-fold>
219. /\*\*
220. \* Return to the main menu when clicking return
221. \* @param evt
222. \*/
223. **private** **void** btnReturnMouseClicked(java.awt.event.MouseEvent evt) {
224. **this**.setVisible(**false**);
225. }
227. /\*\*
228. \* Delete the selected officer and display an appropriate outcome message, before finally setting all the values to blank
229. \* @param evt
230. \*/
231. **private** **void** btnDeleteMouseClicked(java.awt.event.MouseEvent evt) {
232. **if** (cmboOfficerName.getSelectedItem() == **null**) {
233. txtEditOfficerOutcome.setText("Please select an officer before trying to delete");
234. } **else** {
235. String id = txtId.getText();
236. String name = (String)cmboOfficerName.getSelectedItem();
237. **try** {
238. **if** (JOptionPane.showConfirmDialog(**null**, "Are you sure you want to delete \"" + name + "\"?", "Delete", JOptionPane.YES\_NO\_OPTION) == 0) {
239. **this**.deleteOfficer(id);
240. txtEditOfficerOutcome.setText((String)name + " has been deleted, you may now\n"
241. + "edit/delete another officer or close this screen by clicking Return");
242. txtId.setText("");
243. cmboOfficerName.setSelectedIndex(-1);
244. txtMobileNumber.setText("");
245. txtCarReg.setText("");
246. txtNextOfKin.setText("");
247. txtNextOfKinPhone.setText("");
248. cmboManagerName.setSelectedIndex(-1);
249. }
250. } **catch** (SQLException ex) {
251. Logger.getLogger(EditAnOfficer.**class**.getName()).log(Level.SEVERE, **null**, ex);
252. }
253. }
254. }
256. /\*\*
257. \* Updates the selected officer with the details provided
258. \* @param evt
259. \*/
260. **private** **void** btnUpdateMouseClicked(java.awt.event.MouseEvent evt) {
261. **if** (cmboOfficerName.getSelectedItem() == **null**) {
262. txtEditOfficerOutcome.setText("Please select an officer before trying to update");
263. } **else** {
264. String id = txtId.getText();
265. String name = (String)cmboOfficerName.getSelectedItem();
266. **try** {
267. **this**.updateOfficer(id);
268. txtEditOfficerOutcome.setText((String)name + " has been updated, you may now\n"
269. + "edit/delete another officer or close this screen by clicking Return");
270. txtId.setText("");
271. cmboOfficerName.setSelectedIndex(-1);
272. txtMobileNumber.setText("");
273. txtCarReg.setText("");
274. txtNextOfKin.setText("");
275. txtNextOfKinPhone.setText("");
276. cmboManagerName.setSelectedIndex(-1);
277. } **catch** (SQLException ex) {
278. Logger.getLogger(EditAnOfficer.**class**.getName()).log(Level.SEVERE, **null**, ex);
279. }
280. }
281. }
283. /\*\*
284. \* Display the list of officers when clicking on the officer drop down menu
285. \* @param evt
286. \*/
287. **private** **void** cmboOfficerNameMouseClicked(java.awt.event.MouseEvent evt) {
288. cmboOfficerName.removeAllItems();
289. List<String> officers = **null**;
290. **try** {
291. officers = **this**.getOfficers();
292. } **catch** (SQLException ex) {
293. Logger.getLogger(EditAnOfficer.**class**.getName()).log(Level.SEVERE, **null**, ex);
294. }
295. officers.forEach((officer) -> {
296. cmboOfficerName.addItem(officer);
297. });
298. }
300. /\*\*
301. \* Load the current details for the selected officer
302. \* @param evt
303. \*/
304. **private** **void** btnLoadMouseClicked(java.awt.event.MouseEvent evt) {
305. **if** (cmboOfficerName.getSelectedItem() == **null**) {
306. txtEditOfficerOutcome.setText("Please select an officer before clicking load");
307. } **else** {
308. String name = (String)cmboOfficerName.getSelectedItem();
310. **try** {
311. ResultSet results = getOfficersDetails(name);
312. List<String> managers = **null**;
313. **while** (results.next()){
314. txtId.setText(results.getString(1));
315. txtMobileNumber.setText(results.getString(3));
316. txtCarReg.setText(results.getString(4));
317. txtNextOfKin.setText(results.getString(5));
318. txtNextOfKinPhone.setText(results.getString(6));
319. **try** {
320. managers = **this**.getManagers();
321. } **catch** (SQLException ex) {
322. Logger.getLogger(EditAManager.**class**.getName()).log(Level.SEVERE, **null**, ex);
323. }
324. managers.forEach((manager) -> {
325. cmboManagerName.addItem(manager);
326. });
327. System.out.println(results.getString(7));
328. cmboManagerName.setSelectedItem(results.getString(7));
329. }
330. } **catch** (SQLException ex) {
331. Logger.getLogger(EditAnOfficer.**class**.getName()).log(Level.SEVERE, **null**, ex);
332. }
333. }
334. }
336. /\*\*
337. \* Helper method for btnLoadMouseClicked
338. \* @param name
339. \* @return ResultSet of selected officer details
340. \* @throws SQLException
341. \*/
342. **public** ResultSet getOfficersDetails(String name) **throws** SQLException {
343. DBConnect conn = **new** DBConnect();
344. Connection openConn = conn.openConnection();
345. **try** {
346. Statement stmt = openConn.createStatement();
347. String query = "SELECT \* FROM tblofficers WHERE name = \"" + name + "\" ORDER BY name";
348. ResultSet results = stmt.executeQuery(query);
349. System.out.println(results);
350. **return** results;
352. } **catch** (SQLException e) {
353. System.out.println("Exception = " + e);
354. }
355. //System.out.println(list);
356. **return** **null**;
357. }
359. /\*\*
360. \* Helper method for cmboOfficerNameMouseClicked
361. \* @return A list of all officers in tblofficers
362. \* @throws SQLException
363. \*/
364. **private** **static** List<String> getOfficers() **throws** SQLException {
365. List<String> list = **new** ArrayList();
366. DBConnect conn = **new** DBConnect();
367. Connection openConn = conn.openConnection();
368. **try** {
369. Statement stmt = openConn.createStatement();
370. String query = "SELECT name FROM tblofficers ORDER BY name";
371. ResultSet results = stmt.executeQuery(query);
372. list.add(**null**);
373. **while** (results.next()) {
374. list.add(results.getString("name"));
375. }
377. } **catch** (SQLException e) {
378. System.out.println("Exception = " + e);
379. }
381. **return** list;
382. }
384. /\*\*
385. \* Helper method for btnDeleteMouseClicked, deletes the officers with the specified id.
386. \* @param officerId
387. \* @throws SQLException
388. \*/
389. **public** **void** deleteOfficer(String officerId) **throws** SQLException {
390. DBConnect conn = **new** DBConnect();
391. Connection openConn = conn.openConnection();
392. PreparedStatement st = openConn.prepareStatement("DELETE FROM tblofficers WHERE id = \"" + officerId + "\"");
393. st.executeUpdate();
395. }
397. /\*\*
398. \* Helper method for btnUpdateMouseClicked, updates all details for the specified officer
399. \* @param officerId
400. \* @throws SQLException
401. \*/
402. **public** **void** updateOfficer (String officerId) **throws** SQLException {
403. String name = (String)cmboOfficerName.getSelectedItem();
404. String id = txtId.getText();
405. String mobileNumber = txtMobileNumber.getText();
406. String carReg = txtCarReg.getText();
407. String nextOfKin = txtNextOfKin.getText();
408. String nextOfKinPhone = txtNextOfKinPhone.getText();
409. String managerName = (String)cmboManagerName.getSelectedItem();
411. DBConnect conn = **new** DBConnect();
412. Connection openConn = conn.openConnection();
413. PreparedStatement st = openConn.prepareStatement("UPDATE tblofficers SET name = \"" + name +
414. "\", mobileNumber = \"" + mobileNumber + "\", carReg = \"" + carReg +
415. "\", nextOfKin = \"" + nextOfKin + "\", nextOfKinPhone = \"" + nextOfKinPhone +
416. "\", manager = \"" + managerName + "\" WHERE id = \"" + id + "\" ");
417. st.executeUpdate();
418. }
420. /\*\*
421. \* Helper method for cmboManagerNameMouseClicked
422. \* @return A list of all managers in tblmanagers
423. \* @throws SQLException
424. \*/
425. **private** List<String> getManagers() **throws** SQLException {
426. List<String> list = **new** ArrayList();
427. DBConnect conn = **new** DBConnect();
428. Connection openConn = conn.openConnection();
429. **try** {
430. Statement stmt = openConn.createStatement();
431. String query = "SELECT name FROM tblmanagers ORDER BY name";
432. ResultSet results = stmt.executeQuery(query);
433. list.add(**null**);
434. **while** (results.next()) {
435. list.add(results.getString("name"));
436. }
438. } **catch** (SQLException e) {
439. System.out.println("Exception = " + e);
440. }
442. **return** list;
443. }
445. /\*\*
446. \* @param args the command line arguments
447. \*/
448. **public** **static** **void** main(String args[]) {
449. **try** {
450. getOfficers();
451. } **catch** (SQLException ex) {
452. Logger.getLogger(EditAnOfficer.**class**.getName()).log(Level.SEVERE, **null**, ex);
453. }
454. /\* Set the Nimbus look and feel \*/
455. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
456. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
457. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
458. \*/
459. **try** {
460. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
461. **if** ("Nimbus".equals(info.getName())) {
462. javax.swing.UIManager.setLookAndFeel(info.getClassName());
463. **break**;
464. }
465. }
466. } **catch** (ClassNotFoundException ex) {
467. java.util.logging.Logger.getLogger(EditAnOfficer.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
468. } **catch** (InstantiationException ex) {
469. java.util.logging.Logger.getLogger(EditAnOfficer.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
470. } **catch** (IllegalAccessException ex) {
471. java.util.logging.Logger.getLogger(EditAnOfficer.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
472. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
473. java.util.logging.Logger.getLogger(EditAnOfficer.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
474. }
475. //</editor-fold>
476. //</editor-fold>
477. //</editor-fold>
478. //</editor-fold>
479. //</editor-fold>
480. //</editor-fold>
481. //</editor-fold>
482. //</editor-fold>
483. //</editor-fold>
484. //</editor-fold>
485. //</editor-fold>
486. //</editor-fold>
487. //</editor-fold>
488. //</editor-fold>
489. //</editor-fold>
490. //</editor-fold>
491. //</editor-fold>
492. //</editor-fold>
493. //</editor-fold>
494. //</editor-fold>
495. //</editor-fold>
496. //</editor-fold>
497. //</editor-fold>
498. //</editor-fold>
499. //</editor-fold>
500. //</editor-fold>
501. //</editor-fold>
502. //</editor-fold>
503. //</editor-fold>
504. //</editor-fold>
505. //</editor-fold>
506. //</editor-fold>
508. /\* Create and display the form \*/
509. java.awt.EventQueue.invokeLater(() -> {
510. **try** {
511. **new** EditAnOfficer().setVisible(**true**);
512. } **catch** (SQLException ex) {
513. Logger.getLogger(EditAnOfficer.**class**.getName()).log(Level.SEVERE, **null**, ex);
514. }
515. });
516. }
518. // Variables declaration - do not modify
519. **private** javax.swing.JButton btnDelete;
520. **private** javax.swing.JButton btnLoad;
521. **private** javax.swing.JButton btnReturn;
522. **private** javax.swing.JButton btnUpdate;
523. **private** javax.swing.JComboBox<String> cmboManagerName;
524. **private** javax.swing.JComboBox<String> cmboOfficerName;
525. **private** javax.swing.JLabel jLabel1;
526. **private** javax.swing.JLabel jLabel2;
527. **private** javax.swing.JLabel jLabel3;
528. **private** javax.swing.JLabel jLabel4;
529. **private** javax.swing.JLabel jLabel5;
530. **private** javax.swing.JLabel jLabel6;
531. **private** javax.swing.JLabel jLabel7;
532. **private** javax.swing.JLabel jLabel8;
533. **private** javax.swing.JScrollPane jScrollPane1;
534. **private** javax.swing.JTextField txtCarReg;
535. **private** javax.swing.JTextArea txtEditOfficerOutcome;
536. **private** javax.swing.JTextField txtId;
537. **private** javax.swing.JTextField txtMobileNumber;
538. **private** javax.swing.JTextField txtNextOfKin;
539. **private** javax.swing.JTextField txtNextOfKinPhone;
540. // End of variables declaration
541. }

#### 4.13.6 MoreDetails.java

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** GUI;
8. **import** desktop.app.v2.DBConnect;
9. **import** java.sql.Connection;
10. **import** java.sql.ResultSet;
11. **import** java.sql.SQLException;
12. **import** java.sql.Statement;
13. **import** java.util.logging.Level;
14. **import** java.util.logging.Logger;
16. /\*\*
17. \*
18. \* @author zy395907
19. \*/
20. **public** **class** MoreDetails **extends** javax.swing.JFrame {
22. /\*\*
23. \* Creates new form AddAnOfficer
24. \*
25. \* @param name
26. \* @throws java.sql.SQLException
27. \*/
28. **public** MoreDetails(String name) **throws** SQLException {
29. initComponents();
30. **try** {
31. ResultSet results = getOfficersDetails(name);
32. **while** (results.next()) {
33. txtId.setText(results.getString(1));
34. txtName.setText(results.getString(2));
35. txtMobileNumber.setText(results.getString(3));
36. txtCarReg.setText(results.getString(4));
37. txtNextOfKin.setText(results.getString(5));
38. txtNextOfKinPhone.setText(results.getString(6));
39. txtManager.setText(results.getString(7));
40. }
41. } **catch** (SQLException ex) {
42. Logger.getLogger(EditAnOfficer.**class**.getName()).log(Level.SEVERE, **null**, ex);
43. }
44. }
46. /\*\*
47. \* This method is called from within the constructor to initialize the form.
48. \* WARNING: Do NOT modify this code. The content of this method is always
49. \* regenerated by the Form Editor.
50. \*/
51. @SuppressWarnings("unchecked")
52. // <editor-fold defaultstate="collapsed" desc="Generated Code">
53. **private** **void** initComponents() {
55. jLabel1 = **new** javax.swing.JLabel();
56. jLabel2 = **new** javax.swing.JLabel();
57. txtId = **new** javax.swing.JTextField();
58. jLabel4 = **new** javax.swing.JLabel();
59. jLabel5 = **new** javax.swing.JLabel();
60. txtName = **new** javax.swing.JTextField();
61. txtMobileNumber = **new** javax.swing.JTextField();
62. btnReturn = **new** javax.swing.JButton();
63. jLabel6 = **new** javax.swing.JLabel();
64. jLabel7 = **new** javax.swing.JLabel();
65. jLabel8 = **new** javax.swing.JLabel();
66. txtCarReg = **new** javax.swing.JTextField();
67. txtNextOfKin = **new** javax.swing.JTextField();
68. txtNextOfKinPhone = **new** javax.swing.JTextField();
69. jLabel9 = **new** javax.swing.JLabel();
70. txtManager = **new** javax.swing.JTextField();
72. setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE\_ON\_CLOSE);
74. jLabel1.setText("Id Number:");
76. jLabel2.setText("Add an Officer");
78. jLabel4.setText("Name:");
80. jLabel5.setText("Work Mobile:");
82. btnReturn.setText("Return");
83. btnReturn.addMouseListener(**new** java.awt.event.MouseAdapter() {
84. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
85. btnReturnMouseClicked(evt);
86. }
87. });
89. jLabel6.setText("Car Reg:");
91. jLabel7.setText("Next of Kin:");
93. jLabel8.setText("Phone Number:");
95. jLabel9.setText("Manager:");
97. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
98. getContentPane().setLayout(layout);
99. layout.setHorizontalGroup(
100. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
101. .addGroup(layout.createSequentialGroup()
102. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
103. .addGroup(layout.createSequentialGroup()
104. .addGap(176, 176, 176)
105. .addComponent(jLabel2))
106. .addGroup(layout.createSequentialGroup()
107. .addGap(58, 58, 58)
108. .addComponent(jLabel1)
109. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
110. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, 251, javax.swing.GroupLayout.PREFERRED\_SIZE))
111. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
112. .addContainerGap()
113. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
114. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
115. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
116. .addComponent(jLabel4)
117. .addComponent(jLabel5)
118. .addComponent(jLabel6)
119. .addComponent(jLabel7)
120. .addComponent(jLabel8)
121. .addComponent(jLabel9))
122. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
123. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)
124. .addComponent(txtName, javax.swing.GroupLayout.DEFAULT\_SIZE, 251, Short.MAX\_VALUE)
125. .addComponent(txtMobileNumber)
126. .addComponent(txtCarReg)
127. .addComponent(txtNextOfKin)
128. .addComponent(txtNextOfKinPhone)
129. .addComponent(txtManager)))
130. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
131. .addComponent(btnReturn, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)
132. .addGap(108, 108, 108)))))
133. .addContainerGap(33, Short.MAX\_VALUE))
134. );
135. layout.setVerticalGroup(
136. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
137. .addGroup(layout.createSequentialGroup()
138. .addContainerGap()
139. .addComponent(jLabel2)
140. .addGap(41, 41, 41)
141. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
142. .addComponent(jLabel1)
143. .addComponent(txtId, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
144. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
145. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
146. .addComponent(jLabel4)
147. .addComponent(txtName, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
148. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
149. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
150. .addComponent(jLabel5)
151. .addComponent(txtMobileNumber, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
152. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
153. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
154. .addComponent(jLabel6)
155. .addComponent(txtCarReg, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
156. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
157. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
158. .addComponent(jLabel7)
159. .addComponent(txtNextOfKin, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
160. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
161. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
162. .addComponent(txtNextOfKinPhone, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
163. .addComponent(jLabel8))
164. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
165. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
166. .addComponent(jLabel9)
167. .addComponent(txtManager, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
168. .addGap(18, 18, 18)
169. .addComponent(btnReturn)
170. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))
171. );
173. pack();
174. }// </editor-fold>
176. **private** **void** btnReturnMouseClicked(java.awt.event.MouseEvent evt) {
177. **this**.setVisible(**false**);
178. }
180. /\*\*
181. \* Load the current details for the selected officer
182. \*
183. \* @param name
184. \* @param evt
185. \* @return
186. \* @throws java.sql.SQLException
187. \*/
189. **public** ResultSet getOfficersDetails(String name) **throws** SQLException {
190. DBConnect conn = **new** DBConnect();
191. Connection openConn = conn.openConnection();
192. **try** {
193. Statement stmt = openConn.createStatement();
194. String query = "SELECT \* FROM tblofficers WHERE name = \"" + name + "\" ORDER BY name";
195. ResultSet results = stmt.executeQuery(query);
196. **return** results;
197. } **catch** (SQLException e) {
198. System.out.println("Exception = " + e);
199. }
200. **return** **null**;
201. }
203. /\*\*
204. \* @param args the command line arguments
205. \*/
206. **public** **static** **void** main(String args[]) {
207. /\* Set the Nimbus look and feel \*/
208. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
209. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
210. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
211. \*/
212. **try** {
213. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
214. **if** ("Nimbus".equals(info.getName())) {
215. javax.swing.UIManager.setLookAndFeel(info.getClassName());
216. **break**;
217. }
218. }
219. } **catch** (ClassNotFoundException ex) {
220. java.util.logging.Logger.getLogger(MoreDetails.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
221. } **catch** (InstantiationException ex) {
222. java.util.logging.Logger.getLogger(MoreDetails.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
223. } **catch** (IllegalAccessException ex) {
224. java.util.logging.Logger.getLogger(MoreDetails.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
225. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
226. java.util.logging.Logger.getLogger(MoreDetails.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
227. }
228. //</editor-fold>
229. //</editor-fold>
230. //</editor-fold>
231. //</editor-fold>
232. //</editor-fold>
233. //</editor-fold>
234. //</editor-fold>
235. //</editor-fold>
236. //</editor-fold>
237. //</editor-fold>
238. //</editor-fold>
239. //</editor-fold>
240. //</editor-fold>
241. //</editor-fold>
242. //</editor-fold>
243. //</editor-fold>
245. /\* Create and display the form \*/
246. java.awt.EventQueue.invokeLater(**new** Runnable() {
247. @Override
248. **public** **void** run() {
250. }
251. });
252. }
254. // Variables declaration - do not modify
255. **private** javax.swing.JButton btnReturn;
256. **private** javax.swing.JLabel jLabel1;
257. **private** javax.swing.JLabel jLabel2;
258. **private** javax.swing.JLabel jLabel4;
259. **private** javax.swing.JLabel jLabel5;
260. **private** javax.swing.JLabel jLabel6;
261. **private** javax.swing.JLabel jLabel7;
262. **private** javax.swing.JLabel jLabel8;
263. **private** javax.swing.JLabel jLabel9;
264. **private** javax.swing.JTextField txtCarReg;
265. **private** javax.swing.JTextField txtId;
266. **private** javax.swing.JTextField txtManager;
267. **private** javax.swing.JTextField txtMobileNumber;
268. **private** javax.swing.JTextField txtName;
269. **private** javax.swing.JTextField txtNextOfKin;
270. **private** javax.swing.JTextField txtNextOfKinPhone;
271. // End of variables declaration
272. }

#### 4.13.7 DeleteSite.java

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** GUI;
8. **import** desktop.app.v2.DBConnect;
9. **import** java.sql.Connection;
10. **import** java.sql.PreparedStatement;
11. **import** java.sql.ResultSet;
12. **import** java.sql.SQLException;
13. **import** java.sql.Statement;
14. **import** java.util.ArrayList;
15. **import** java.util.List;
16. **import** java.util.logging.Level;
17. **import** java.util.logging.Logger;
18. **import** javax.swing.JOptionPane;
20. /\*\*
21. \*
22. \* @author zy395907
23. \*/
24. **public** **class** DeleteSite **extends** javax.swing.JFrame {
26. /\*\*
27. \* Creates new form DeleteSite
28. \*/
29. **public** DeleteSite() {
30. initComponents();
31. }
33. /\*\*
34. \* This method is called from within the constructor to initialize the form.
35. \* WARNING: Do NOT modify this code. The content of this method is always
36. \* regenerated by the Form Editor.
37. \*/
38. @SuppressWarnings("unchecked")
39. // <editor-fold defaultstate="collapsed" desc="Generated Code">
40. **private** **void** initComponents() {
42. jButton1 = **new** javax.swing.JButton();
43. jButton2 = **new** javax.swing.JButton();
44. cmboSites = **new** javax.swing.JComboBox<>();
45. jLabel1 = **new** javax.swing.JLabel();
46. jLabel2 = **new** javax.swing.JLabel();
47. jScrollPane1 = **new** javax.swing.JScrollPane();
48. txtOutcome = **new** javax.swing.JTextArea();
50. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
52. jButton1.setText("Delete");
53. jButton1.addMouseListener(**new** java.awt.event.MouseAdapter() {
54. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
55. jButton1MouseClicked(evt);
56. }
57. });
59. jButton2.setText("Return");
60. jButton2.addMouseListener(**new** java.awt.event.MouseAdapter() {
61. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
62. jButton2MouseClicked(evt);
63. }
64. });
66. cmboSites.addMouseListener(**new** java.awt.event.MouseAdapter() {
67. **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {
68. cmboSitesMouseClicked(evt);
69. }
70. });
72. jLabel1.setText("Delete Site");
74. jLabel2.setText("Site:");
76. txtOutcome.setEditable(**false**);
77. txtOutcome.setColumns(20);
78. txtOutcome.setRows(2);
79. txtOutcome.setAutoscrolls(**false**);
80. jScrollPane1.setViewportView(txtOutcome);
82. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());
83. getContentPane().setLayout(layout);
84. layout.setHorizontalGroup(
85. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
86. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
87. .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
88. .addComponent(jLabel1)
89. .addGap(181, 181, 181))
90. .addGroup(layout.createSequentialGroup()
91. .addGap(77, 77, 77)
92. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, **false**)
93. .addGroup(layout.createSequentialGroup()
94. .addComponent(jLabel2)
95. .addGap(28, 28, 28)
96. .addComponent(cmboSites, javax.swing.GroupLayout.PREFERRED\_SIZE, 197, javax.swing.GroupLayout.PREFERRED\_SIZE))
97. .addGroup(layout.createSequentialGroup()
98. .addComponent(jButton1)
99. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
100. .addComponent(jButton2)))
101. .addContainerGap(76, Short.MAX\_VALUE))
102. .addComponent(jScrollPane1)
103. );
104. layout.setVerticalGroup(
105. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
106. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
107. .addGap(29, 29, 29)
108. .addComponent(jLabel1)
109. .addGap(54, 54, 54)
110. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
111. .addComponent(cmboSites, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
112. .addComponent(jLabel2))
113. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 45, Short.MAX\_VALUE)
114. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
115. .addComponent(jButton1)
116. .addComponent(jButton2))
117. .addGap(18, 18, 18)
118. .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 56, javax.swing.GroupLayout.PREFERRED\_SIZE))
119. );
121. pack();
122. }// </editor-fold>
124. **private** **void** jButton2MouseClicked(java.awt.event.MouseEvent evt) {
125. **this**.setVisible(**false**);
126. }
128. **private** **void** cmboSitesMouseClicked(java.awt.event.MouseEvent evt) {
129. cmboSites.removeAllItems();
130. List<String> sites = **null**;
131. **try** {
132. sites = **this**.getSites();
133. } **catch** (SQLException ex) {
134. Logger.getLogger(DeleteSite.**class**.getName()).log(Level.SEVERE, **null**, ex);
135. }
136. sites.forEach((site) -> {
137. cmboSites.addItem(site);
138. });
139. }
141. **private** **void** jButton1MouseClicked(java.awt.event.MouseEvent evt) {
142. String siteName = (String) cmboSites.getSelectedItem();
143. **if** (cmboSites.getSelectedItem() == **null**) {
144. txtOutcome.setText("Please select a manager before trying to update");
145. } **else** {
146. **try** {
147. **if** (JOptionPane.showConfirmDialog(**null**, "Are you sure you want to delete \"" + siteName + "\"?", "Delete", JOptionPane.YES\_NO\_OPTION) == 0) {
148. **this**.deleteSite(siteName);
149. txtOutcome.setText((String) siteName + " has been deleted, you may now\n"
150. + "delete another site or close this screen by clicking Return");
151. }
152. } **catch** (SQLException ex) {
153. Logger.getLogger(DeleteSite.**class**.getName()).log(Level.SEVERE, **null**, ex);
154. }
155. }
156. }
158. **private** **void** deleteSite(String siteName) **throws** SQLException {
159. DBConnect conn = **new** DBConnect();
160. Connection openConn = conn.openConnection();
161. PreparedStatement st = openConn.prepareStatement("DELETE FROM tblsite WHERE siteName = \"" + siteName + "\"");
162. st.executeUpdate();
163. }
165. **private** List<String> getSites() **throws** SQLException {
166. List<String> list = **new** ArrayList();
167. DBConnect conn = **new** DBConnect();
168. Connection openConn = conn.openConnection();
169. **try** {
170. Statement stmt = openConn.createStatement();
171. String query = "SELECT siteName FROM tblsite";
172. ResultSet results = stmt.executeQuery(query);
173. list.add(**null**);
174. **while** (results.next()) {
175. list.add(results.getString("siteName"));
176. }
178. } **catch** (SQLException e) {
179. System.out.println("Exception = " + e);
180. }
181. **return** list;
182. }
184. /\*\*
185. \* @param args the command line arguments
186. \*/
187. **public** **static** **void** main(String args[]) {
188. /\* Set the Nimbus look and feel \*/
189. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
190. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
191. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
192. \*/
193. **try** {
194. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
195. **if** ("Nimbus".equals(info.getName())) {
196. javax.swing.UIManager.setLookAndFeel(info.getClassName());
197. **break**;
198. }
199. }
200. } **catch** (ClassNotFoundException ex) {
201. java.util.logging.Logger.getLogger(DeleteSite.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
202. } **catch** (InstantiationException ex) {
203. java.util.logging.Logger.getLogger(DeleteSite.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
204. } **catch** (IllegalAccessException ex) {
205. java.util.logging.Logger.getLogger(DeleteSite.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
206. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
207. java.util.logging.Logger.getLogger(DeleteSite.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
208. }
209. //</editor-fold>
211. /\* Create and display the form \*/
212. java.awt.EventQueue.invokeLater(**new** Runnable() {
213. **public** **void** run() {
214. **new** DeleteSite().setVisible(**true**);
215. }
216. });
217. }
219. // Variables declaration - do not modify
220. **private** javax.swing.JComboBox<String> cmboSites;
221. **private** javax.swing.JButton jButton1;
222. **private** javax.swing.JButton jButton2;
223. **private** javax.swing.JLabel jLabel1;
224. **private** javax.swing.JLabel jLabel2;
225. **private** javax.swing.JScrollPane jScrollPane1;
226. **private** javax.swing.JTextArea txtOutcome;
227. // End of variables declaration
228. }

#### 4.13.8 SiteVisit.java

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** GUI;
8. /\*\*
9. \*
10. \* @author zy395907
11. \*/
12. **class** SiteVisit {
13. **private** String name, site, timestamp;
15. **public** SiteVisit(String name, String site, String timestamp){
16. **this**.name = name;
17. **this**.site = site;
18. **this**.timestamp = timestamp;
19. }
21. **public** String getName() {
22. **return** name;
23. }
25. **public** String getSite() {
26. **return** site;
27. }
29. **public** String getTimestamp() {
30. **return** timestamp.substring(11, 16);
31. }
32. }

#### 4.13.9 DBConnect.java

1. /\*
2. \* To change this license header, choose License Headers in Project Properties.
3. \* To change this template file, choose Tools | Templates
4. \* and open the template in the editor.
5. \*/
6. **package** desktop.app.v2;
8. **import** java.sql.\*;
10. /\*\*
11. \*
12. \* @author zy395907
13. \*/
14. **public** **class** DBConnect {
16. **public** Connection openConnection() {
17. **try** {
18. **final** String DB\_HOST = "jdbc:mysql://localhost:3306/tm470";
19. **final** String DB\_USER = "root";
20. **final** String DB\_PASSWD = "";
22. Connection conn = DriverManager.getConnection(
23. DB\_HOST, DB\_USER, DB\_PASSWD);
24. **return** conn;
26. } **catch** (SQLException e) {
27. System.out.println(e.getMessage());
28. }
30. **return** **null**;
31. }
33. }