BIRDS EYE SECURITY SURVEILLANCE

# BIRDS EYE SECURITY SURVEILLANCE

System Specification Document

Ryan Trickett
Gr12 Reddam House Bedfordview
Exam Number: 201112020858

# TABLE OF CONTENTS

# SUMMARY

Birds Eye Security Surveillance aims to make users feel safe 24/7 and allow them to be at ease when they are away from their home or family. Birds Eye Security Surveillance will achieve this by having drones that are able to do high quality surveillance, each drone will be designed for each surveillance they must perform, such as night vision for night flights or infrared for emergency situations. Birds Eye Security Surveillance will also be faster than any other security company as we will not be delayed by traffic or human error, thus making Birds Eye Security Surveillance more efficient and giving clients better piece of mind.

Crime in South Africa and all around the world is growing and in order to stop it we need to implement new solutions that use advances in technology. Birds Eye Security Surveillance will help to fight crime and keep people safe. We always hope to make every household feel safe and allow that they can live their lives while we ensure the security of it.

A person will never have to feel uneasy when leaving a child at home as they will be notified of anything that is unusual and immediately contact the local police department in order to ensure total safety of a household when a person is away from their house.

When going on vacation many people worry about their house and its possessions and therefore don't enjoy their holiday or don't go as often as they would like to. Birds Eye Security Surveillance makes it easy for ongoing surveillance while a person is away and thus ensuring they have peace of mind about their house and its possessions.

We are here to ensure peace of mind and that every person is safe all the time. Birds Eye Security Surveillance follows the slogan, *Live Your Life, While We Secure It.*

# PROGRAM FUNCTIONS SPECIFICATIONS

This section includes the interface and program functionality

## Login Application

### Login Page

- Displayed in front of the user is two text editors in which the user must enter an **email**, if they are a client, or a **username**, if they are a member of staff, and their **password**.
- There is a button for a user to toggle between making their password visible or invisible (shown as "*”). The button switches between “show” and “hide” depending on the visibility of the password.
- A button at the bottom of the page allows the user to **login**, if their password and username/email is correct, or if the user presses **“enter”** within the password text editor the same action will be achieved.
- There is a button that will allow the user to proceed to the **register page** if they have not created and account.

### Add New User

- Users are displayed a page with text edits and a dropdown in which they must **fill in information** such as:
    - Name and Surname
    - Email
    - Password
    - Address Line One
    - Suburb
- If a user’s suburb is not in the suburb dropdown (the drone service is not available in their area) a user may click a button in order to **recommend a new suburb** and be added to a **mailing list** in case the suburb is invested in.
- A user can then **register** after filling in all the information by clicking the register button. This will create a new user and the user will be able to **access the client application**.
- If a user has **already register**, they can click on the already registered button in order to return to the login page.

*Only clients can be added through the login application

# Client Application

## Drone Services

- Displayed in front of a user is a section where they can sign up for a drone services, in order to select a drone service a client must select the service they require from a dropdown and then select the dates that they would like the service to be provided.
- Alongside the services is a breakdown of how much a client will be paying and a "Confirm" button which they must click in order to purchase their service. (Clients may not change or cancel any services request).
- There is a rich edit that states every service that a client is currently signed up for.
- A user can select a service they wish to request, and request it if their service doesn't overlap with another of the same service and a drone is available

## Response Logs

- A grid of all the flights that have been scheduled, in flight or completed for a client are shown, where a user may click on a record in order to look at all the information on the specific flight.
- The grid can be sorted by date and/or service
- A user can use this to see the incident report, which was formulated by a staff member, for a specific flight.
- A user can click a button in order to view the footage if a flight has been completed

## Reports

- Users can choose either a suburb report or service cost report and a report once one is selected a user can click a button to show the report
- If a report is being shown a user can click a button to send the report to the via email

## Account Details

- Users can see all the details about their account, such as email, first name, surname, etc.
- By clicking "Change Password" button a user can change their password by confirming their current password and the entering a new one that is to the password requirements.
    - An email will then be sent to a user stating that their password was changed along with the time of the change

# Staff Application

## Dashboard

- The first screen is used to show all the data for the drones and suburbs
    - Drones: information on how many of each type of drone is deployed and how busy the drones are.
    - Suburbs: information on and how many drones are active

## Analyse Surveillance

- A staff member is presented by a grid with every surveillance flight assigned to them that has not been completed.
    - Once a staff member clicks on a specific surveillance flight a new form will come up where the user must look at the surveillance footage and write a short description of their analysis of the surveillance.
    - A user can then click “Complete” in order to complete their analysis of the surveillance, which is then sent to clients that were connected to that surveillance flight. A time and date also will be saved in order to track performance of employees.

## Drone Management

- Add Drone:
    - Choose drone type
    - Enter drone’s serial number or generate via the application
    - Choose drones suburb
    - SQL inserts drone into the table, as active
- Change Drone Status
    - Select drone from drones
    - Change details of the drone through inputs and dropdowns
    - Drone details are updated in the table through SQL

## Suburb Management

- Add New Suburb
    - Enter the name of the suburb and postal code or choose from recommended suburbs
    - SQL inserts the suburb into the database
    - If the suburb was part of the recommended suburbs, then an email notification will be sent to any user that was interested in the specific suburb
- Change Suburb Name
    - Select a suburb
    - Enter new name of a suburb
    - Name updates in database table

## Staff Management

- Add Staff Member
    - Enter staff members details (Name, Surname, Username, Email, etc)
    - SQL inserts the staff member
- Deactivate/Reactivate Staff Member
    - Select an employee that is shown
    - Staff member will be updated as inactive/active

## Flight Logs

- All the details of each flight
    - Flight status be filtered by:
        - All (default)
        - Completed
        - In Flight
    - It can be sorted by:
        - Suburb
        - Drone Type
        - Latest to Earliest Response
        - Earliest to Latest Response
        - Assigned Staffs Name
    - Date can be chosen to filter results

## Reports

- Drone Reports
    - List all drones
    - List drones that are active
    - List drones that are being repaired or being serviced
- Suburb Reports
    - List all suburbs
    - List all suburbs and the amount of drone flights per drone type
- Recommended Suburb Reports
    - List all recommended suburbs
    - List all recommended suburbs with a specific amount of interest

## Account Details

- Users can see all the details about their account, such as email, first name, surname, etc.
- By clicking “Change Password” button a user can change their password by confirming their current password and the entering a new one that is to the password requirements.
    - An email will then be sent to a user stating that their password was changed along with the time of the change

# Footage Form

This form allows a client to view surveillance footage which has been assigned to a specific flight. A user can view the image in quite a large screen in order to see all the details of the footage. At the bottom of the form there is a close button in order to go back to the main application.

# FORMS OF HELP

## Web Help

Whenever a user needs help, they can click the help button that is on every page. This will take a user outside of the app to a custom website where they can effortlessly find their issue and be helped. Whenever a user is on a specific page the website will load a specific page of the website in order to help with faster help.

## Integrated Help

All necessary components will use descriptive information in order to show their function. Whenever a user is required to enter information a label will describe what the user is required to enter, and to avoid mistakes when certain data types are meant to be used. Components such as dropdowns or spin edits will help prevent incorrect data input.

# SPECIFICATIONS OF DATA STORAGE

## Database Structure

Birds Eye Security Surveillance uses an Microsoft Access database to store data for permanent use of any data that is related and required in order to run the application. Querying is used in order to retrieve information and data in order to be used within the application and to compile reports.

## Text Files

The application uses text files in order to permanently save data in text format in order to present information for users or allows basic features to run using data stored within such text files.

# HARDWARE AND SOFTWARE REQUIREMENTS

## Hardware

### Minimum:

- **CPU:** Intel i3 540 / AMD FX-6300
- **GPU:** Integrated Intel HD Graphics
- **RAM:** 2GB
- **STORAGE:** 1GB free space
- **INTERNET:** 1Mb/s

### Recommended:

- **CPU:** Intel® Core™ i3-7350K / AMD Ryzen 3 2200G
- **GPU:** Integrated Intel HD Graphics
- **RAM:** 4GB
- **STORAGE:** 2GB free space
- **INTERNET:** 5Mb/s

## Software

- **OPERATION SYSTEM:** Windows 7 or newer
- **Jet 4.0 Microsoft Access Driver**
- **SSL Libraries:** libeay32.dll and ssleay32.dll


# BIRDS EYE SECURITY SURVEILLANCE

System Design Document

Ryan Trickett
Gr12 Reddam House Bedfordview
Exam Number: 201112020858

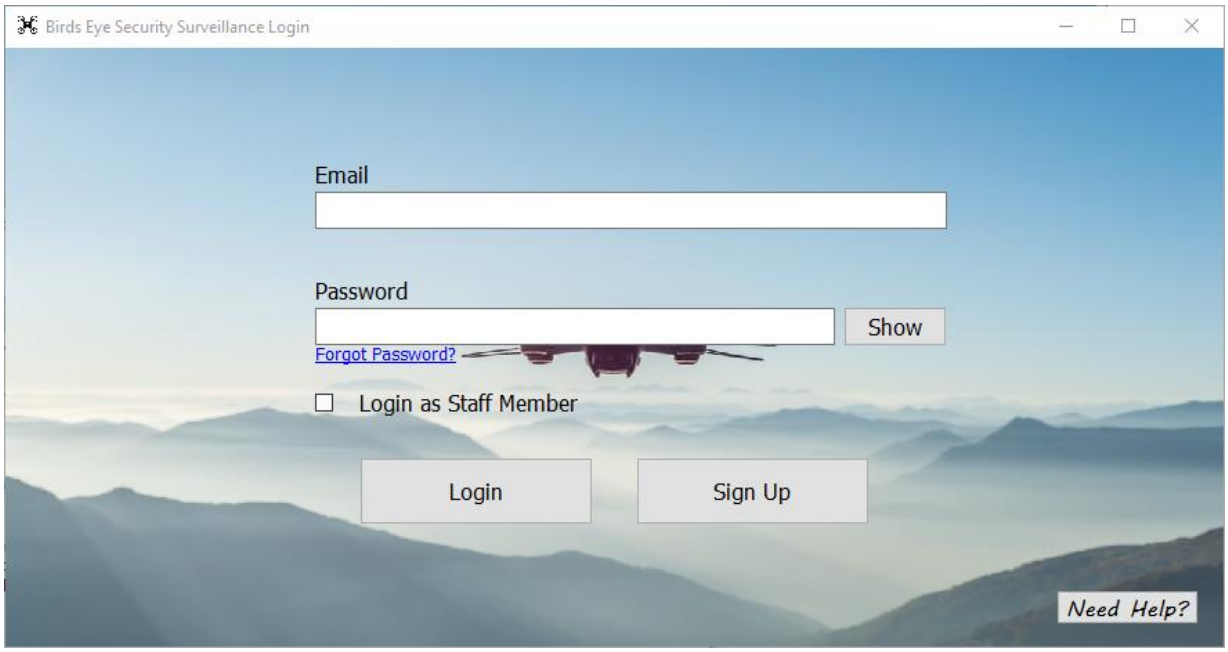
# TABLE OF CONTENTS

# USER INTERFACE DESIGN

## Login Application

This is the form that is shown upon launch of the application. It allows users, clients or staff members, to access the main application through the logging into the application or creating a new account (if they are a client)

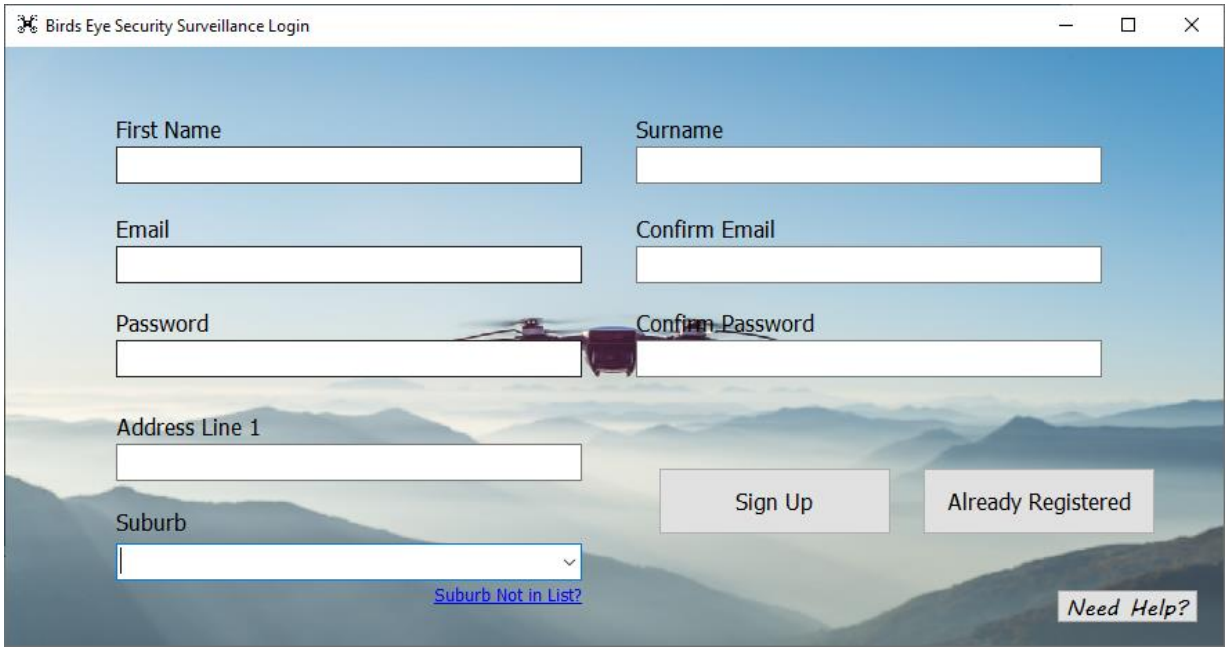
## Login




- The user is required to enter an email and password that is part of an account that has already been created. A user is to check the "Login as Staff Member" if they are a staff member of the company.
- Any information that is entered is validated, and if any information is incorrect an appropriate error message will be displayed.
- A user may display or hide their password if they wish to ensure that they have correctly entered their password.
- If a user does not remember their password, they may access their account via a recovery code.

| Login - Actions | |
|---|---|
| **Action** | **Description** |
| Login | • The “Login” button will log a user in, client or staff, if they have met the following criteria:<br>  ○ The email is part of the application database<br>  ○ The password is correct according to the record within the application’s database |
| Show/Hide | • The “Show/Hide” button will allow a user to see their password without the password key and hide it again |
| Login as Staff Member | • If the “Login as Staff Member” checkbox is unchecked, then a user will be classified as a client and treated as one<br>• If the checkbox is checked then the user will be classified as a staff member and signed in as one |
| Sign Up | • This will allow users to transition to the sign up section where they are able to create a new account |
| Help | The “Help” button will direct the user to a website with specified help for the “Login” page |

## Sign Up



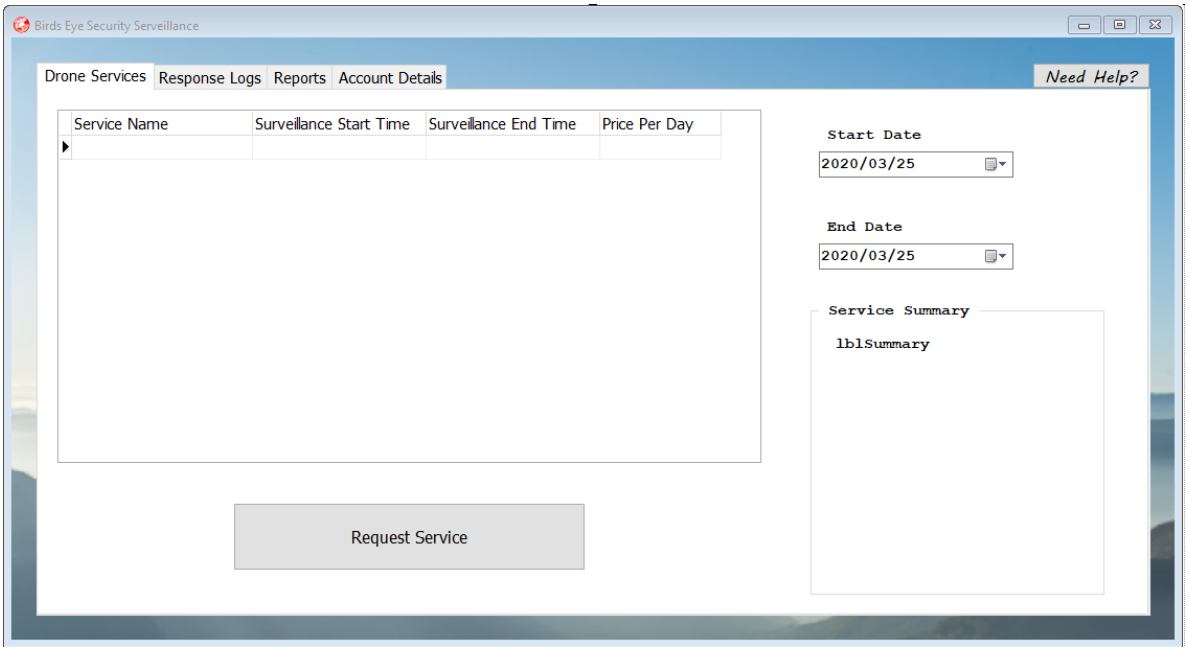

- A user is required to fill in all the field and provide an email that is not already in the applications database.
- They need to confirm their email and password by rewriting their email and password twice.
- If a client's suburb is not listed, then they may recommend their suburb and be added to a mailing list encase their suburb becomes included within the areas that the drones survey.

| Sign Up - Actions | |
|---|---|
| **Action** | **Description** |
| Sign Up | • The "Sign Up" button will create a client account if the following criteria are met:<br>  ○ Email must not be part of the application's database<br>  ○ The password must be between 8 and 15 characters long and must follow a set of guidelines which is stated under of the Help Section<br>  ○ Both the email and password must be entered correctly twice to ensure that there is no misspelling<br>  ○ All fields must be filled with data |
| Suburb Not in List? | • The "Suburb Not in List?" text allows users to recommend their suburb for future expansion<br>• The client is required to enter the suburb and area code they live in<br>  ○ If the suburb and area code match a suburb within the database an error message will be shown stating that their suburb is part of the listed suburbs<br>• An email must then be provided to contact the user if the suburb that they live in becomes part of the surveyed areas |
| Already Registered | • The "Already Registered" button will take a user back to the login section |
| Help | The "Help" button will direct the user to a website with specified help for the "Sign Up" page |

# Client Application

This form allows clients to access all the features from Birds Eye Security Systems, such as drone surveillance services and surveillance reports.

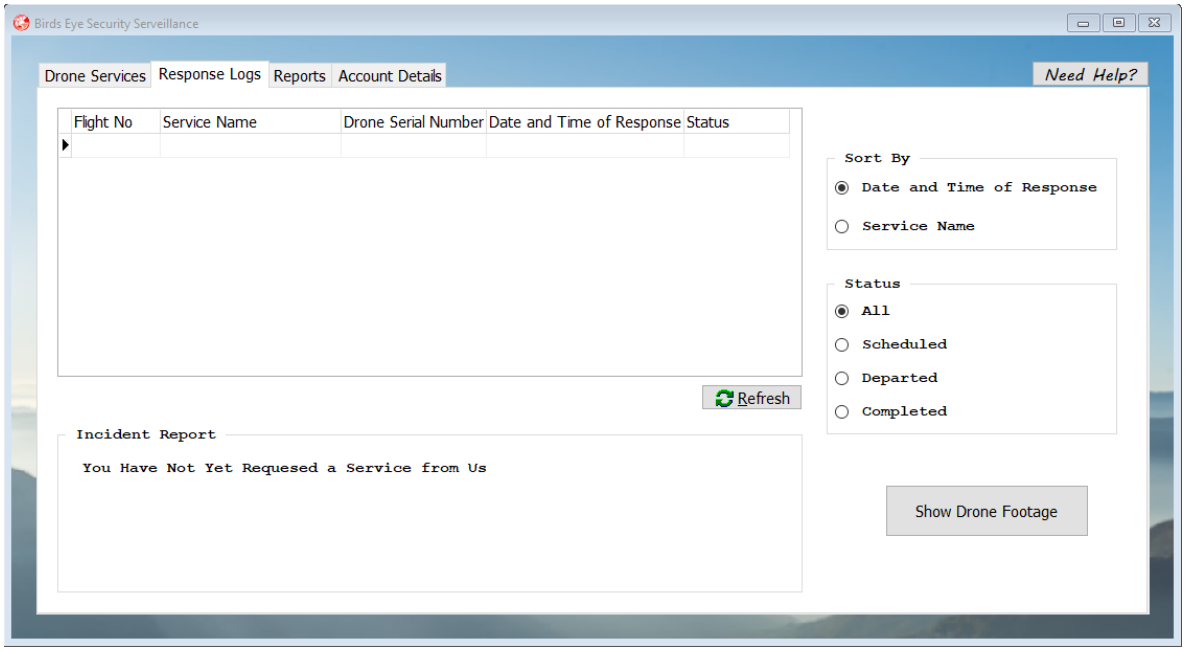
## Drone Services




- The user can request a surveillance service between specific dates, entered by the user.
- A service summary is automatically displayed. Whenever a user changes their service or the dates the summary is updated. The summary displays the description of the service, the amount of days the service will be provided and the total price of the service.
- The information entered by the user is cross checked with their current service requests, and if a request has already been submitted then an error message is displayed.

| Drone Services - Actions | |
|---|---|
| **Action** | **Description** |
| Start Date | • The “Start Date” date picker changes the start date of the service summary and the summary is then updated |
| End Date | • The “End Date” date picker changes the end date of the service summary and the summary is then updated |
| Services Database Grid | • When a cell in the service database grid is clicked, the service summary is updated with new details of the service and the price is updated to the specific services price |
| Request Service | • The “Request Service” button will assign the specified service to the user if the following criteria is met:<br>  ○ The user does not have a service that overlaps with the dates that they specified<br>  ○ Drones are available for the service, meaning they have not been requested by other users between those dates |
| Help | The “Help” button will direct the user to a website with specified help for the “Drone Services” page |

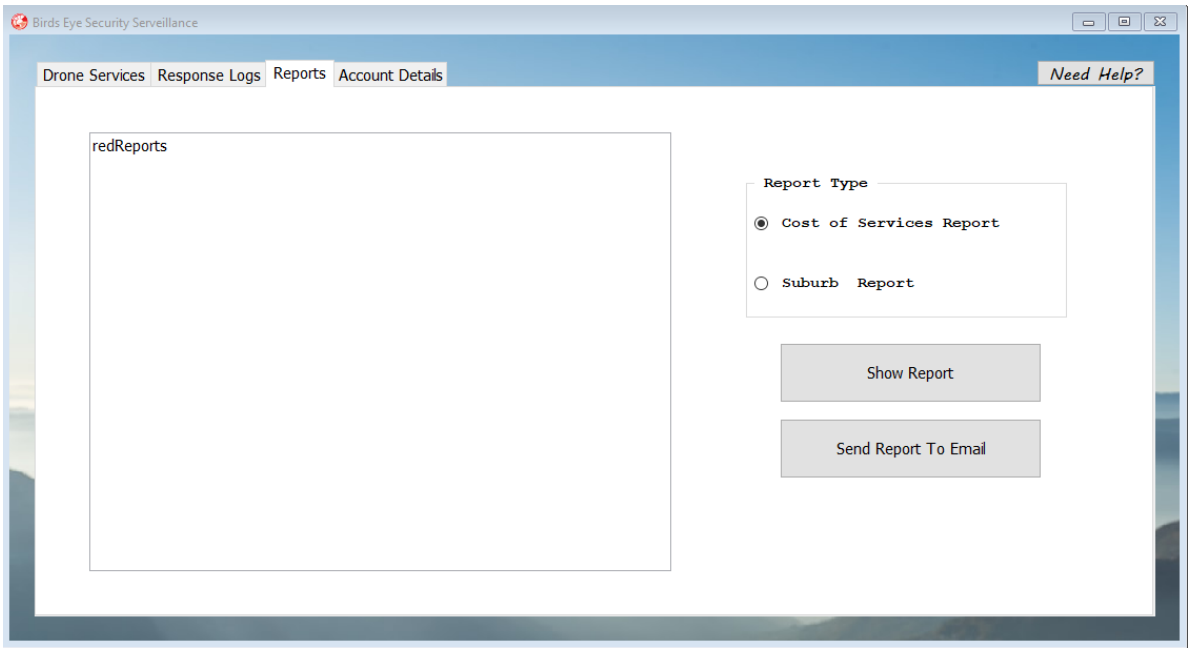
## Response Logs




- The user can see all the drone responses that they have requested, this includes drone flights that are scheduled for future dates.
- The user can order the database table by Date of Response or Service Name, as well as being able to view specific flight statuses or all.
- The user is able to view the incident report for every flight that they have requested, the incident report is entered by a member of staff and summarises the surveillance on a specific drone flight.
- A user is also able to view the footage taken during a specific drone flight.

| Response Logs - Actions | |
|---|---|
| **Action** | **Description** |
| Sort By | • The “Sort By” radio group changes the order in which the response logs database grid is ordered by:<br>  ○ If “Date and Time of Response” selected, then the database grid is ordered by the “Date and Time of Response” field with the latest date at the top<br>  ○ If “Service Name” then the database grid is ordered by the “Service Name” field alphabetically |
| Status | • The “Status” radio group changes the statuses shown in response logs database grid:<br>  ○ If “All” selected, then the database grid will show all the flights<br>  ○ If “Scheduled” then the database grid will show flights which are scheduled for a later date/time<br>  ○ If “Departed” then the database grid will show flights which have departed (drones which are in flight)<br>  ○ If “Completed” then the database grid will show flights, which have been completed |
| Response Logs Database Grid | • When a cell in the flight logs database grid is clicked, the incident report will be displayed for the specific flight which has been selected |
| Refresh | • The “Refresh” button refreshes the data, showing all changes |
| Show Footage | • The “Show Footage” button displays a pop-up form with the footage captured by the drone during a specific flight |
| Help | • The “Help” button will direct the user to a website with specified help for the “Response Logs” page |

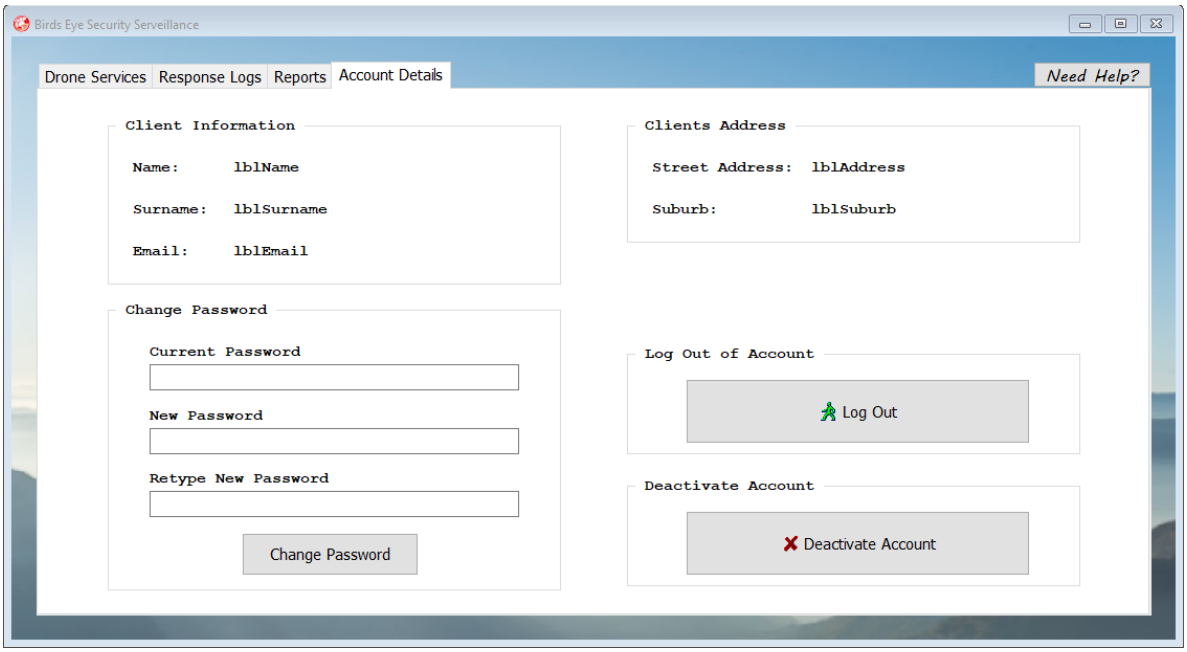
# Reports




- The user can see reports created according to data stored within the database pertaining to the specific user. The user can choose the report they wish to view, and the report is then displayed within the text box.
- The reports can be sent to the user email if they wish to keep the report as reference.
- This page is read only and thus no data can be entered by a user.

| Reports - Actions | |
|---|---|
| **Action** | **Description** |
| Show Report | • The "Show Report" button displays a report according to the report selected within the "Report Type" radio group.<br>  ○ If the report is for "Cost of Service Report" then it will show the user a report with the total cost of all the services, they requested for the current month<br>  ○ If the report is for "Suburb report" then it will show the user a report with the total amount of drone deployments for each drone type in their suburb for the current month |
| Send Report to Email | • The "Send Report to Email" button sends the report to the users email address |
| Help | • The "Help" button will direct the user to a website with specified help for the "Reports" page |

## Account Details



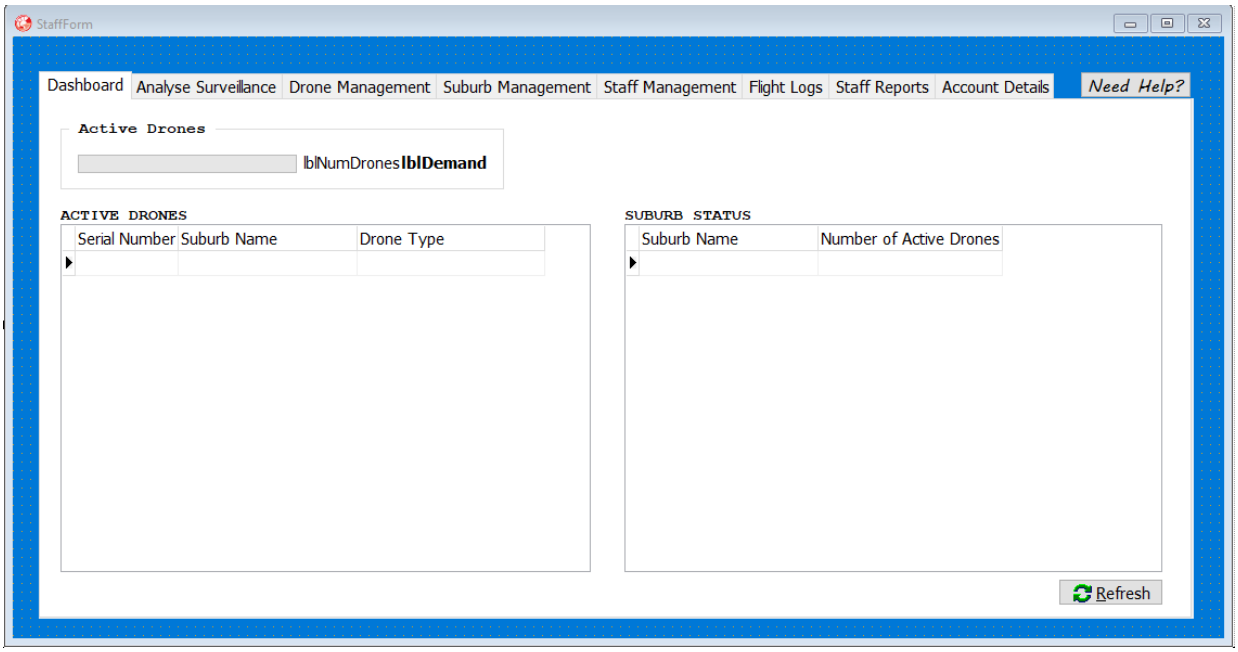

- The user can view the details connected to their account, for example their name and email address.
- If a user wishes to change their password, they may do so by entering their current password and then entering their new password twice to ensure no errors within the password string.
- A user may also log out of the application or deactivate their account.

| Account Details - Actions | |
|---|---|
| **Action** | **Description** |
| Change Password | • The "Change Password" button will change a user's password if:<br>○ The current password entered corresponds with the password connected to the user's account<br>○ The new password must be between 8 and 15 characters long and must follow a set of guidelines which is stated under of the Help Section<br>○ The new password must be the same as the retyped new password |
| Log Out | • The "Log Out" button will log the user out of the application<br>• The user will be redirected back to the login form |
| Deactivate Account | • The "Deactivate Account" button will deactivate a user's account |
| Help | • The "Help" button will direct the user to a website with specified help for the "Account Details" page |

# Staff Application

This form allows staff to access all the features from Birds Eye Security Systems, each staff member is able to access specific tabs of the staff application. For example, surveillance analyse staff cannot add new drones.

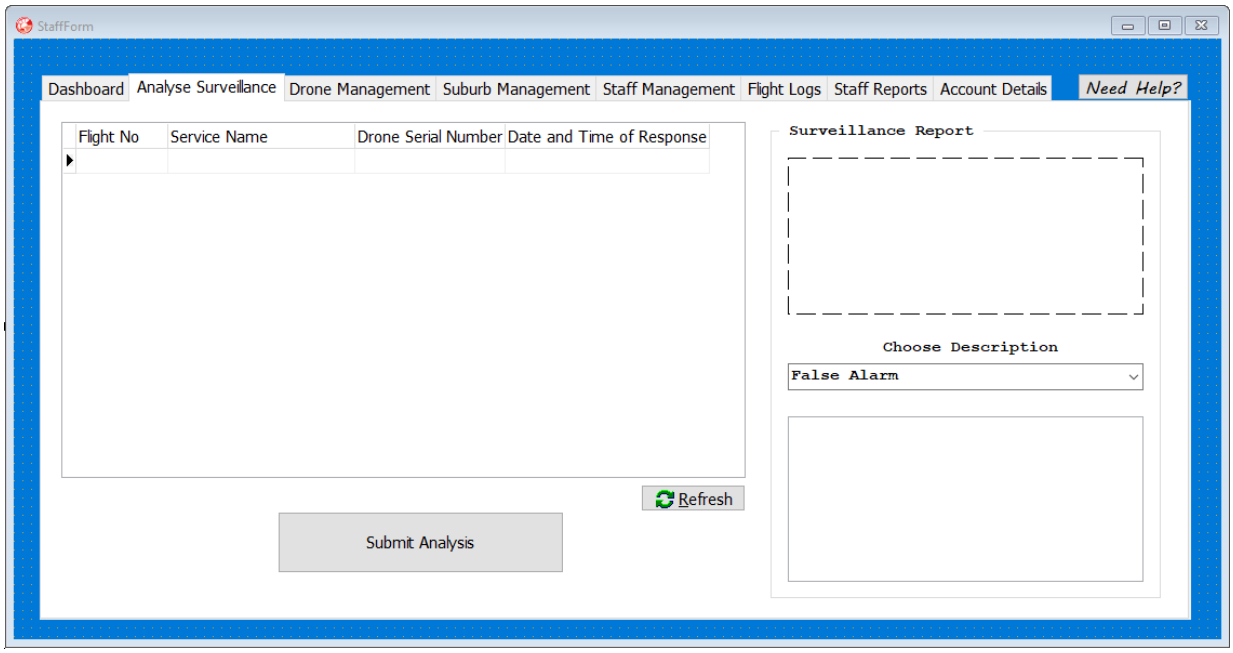
## Dashboard




- The user is unable to enter any data into this page as it is used to give information about the current demand on the companies drone deployment and which suburbs need more drones

| Dashboard - Actions | |
|---|---|
| **Action** | **Description** |
| Refresh | • The "Refresh" button refreshes the data, showing all changes |
| Help | • The "Help" button will direct the user to a website with specified help for the "Dashboard" page |

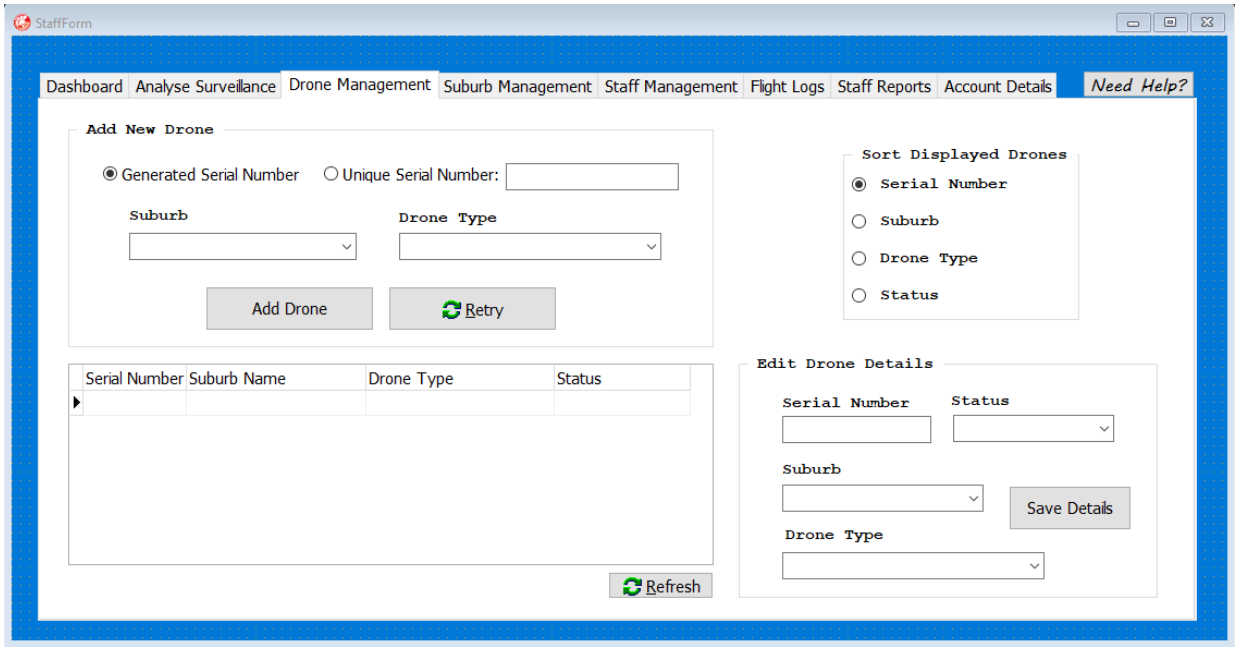
## Analyse Surveillance




- The users, who falls under the "Surveillance Analyst", will be able to view footage from surveillance service requests assigned to them.
- The user may choose one of the options in which to describe the surveillance footage or write their own summary below

| Analyse Surveillance - Actions | |
|---|---|
| **Action** | **Description** |
| Submit Analyses | • The "Submit Analyses" button submits the description of the surveillance footage and time stamps when the footage was analysed |
| Refresh | • The "Refresh" button refreshes the data, showing all changes |
| Help | • The "Help" button will direct the user to a website with specified help for the "Analyse Surveillance" page |

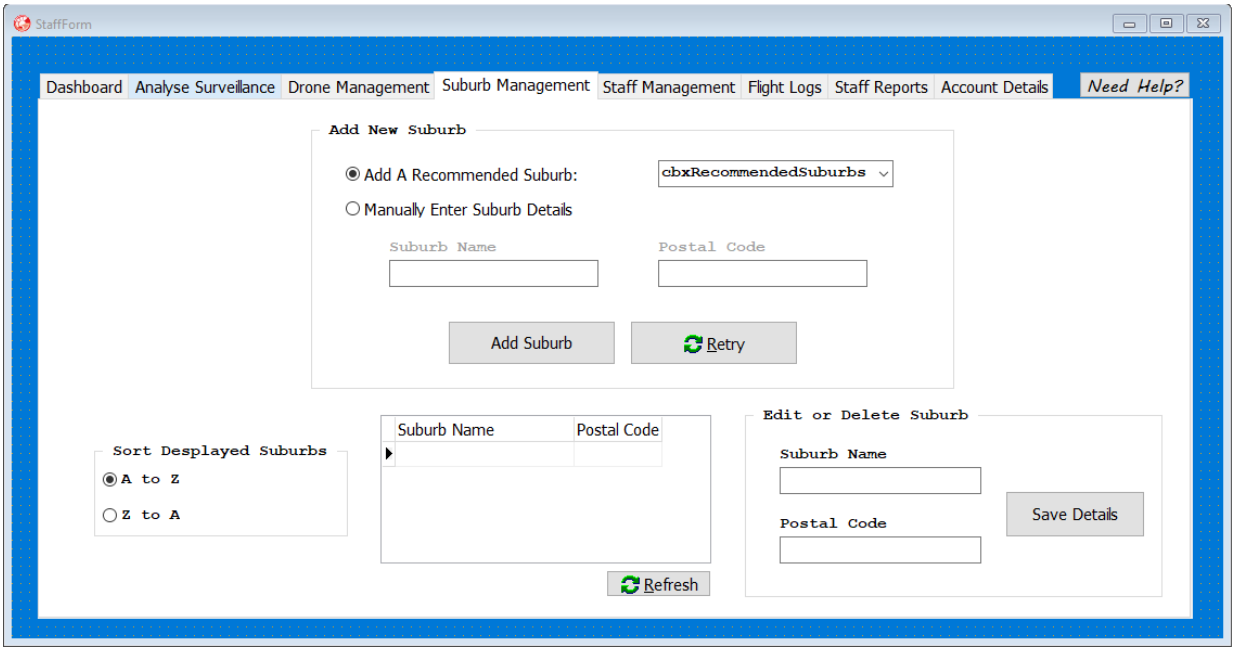
## Drone Management




- The users, who falls under the "Administrator", will be able to add and change details of drones.
- The user may add a drone by adding a specified serial number or having one auto generated by the application.
- All the details of the drones are able to be changed and are immediately updated.

| Drone Management - Actions | |
|---|---|
| **Action** | **Description** |
| Add Drone | • The "Add Drone" button will add a drone if:<br>  ○ The serial number is unique and not connected to any other drone |
| Retry | • The "Retry" button clears all the fields from the "Add Drone" section of the page |
| Sort Displayed Drones | • The "Sort Displayed Drones" radio group will allow for the drones database grid to be sorted by either:<br>  ○ Serial Number<br>  ○ Suburb<br>  ○ Drone Type<br>  ○ Status |
| Save Details | • The "Save Details" button saves all the changes made to the details of the selected drone |
| Refresh | • The "Refresh" button refreshes the data, showing all changes |
| Help | • The "Help" button will direct the user to a website with specified help for the "Drone Management" page |

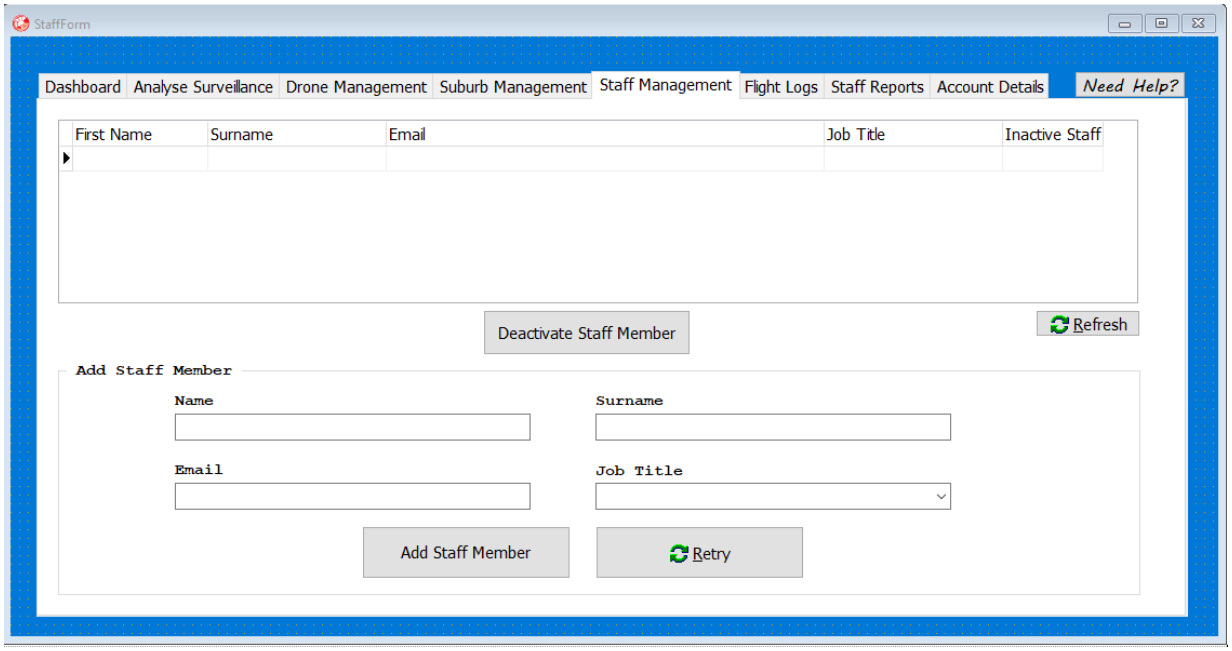
## Suburb Management




- The users, who falls under the "Administrator", will be able to add and change details of suburbs.
- The user may add a suburb by adding a specified postal code or choosing one of the suburbs from the recommended suburbs drop down.
- All the details of the suburbs are able to be changed and are immediately updated.

| Suburb Management - Actions | |
|---|---|
| **Action** | **Description** |
| Add Suburb | • The "Add Suburb" button will add a suburb if:<br>○ The postal code is unique and not connected to any other suburbs |
| Retry | • The "Retry" button clears all the fields from the "Add Suburb" section of the page |
| Sort Displayed Suburbs | • The "Sort Displayed Suburbs" radio group will allow for the suburb's database grid to be sorted from:<br>○ A to Z<br>○ Z to A |
| Save Details | • The "Save Details" button saves all the changes made to the details of the selected suburb |
| Refresh | • The "Refresh" button refreshes the data, showing all changes |
| Help | • The "Help" button will direct the user to a website with specified help for the "Suburb Management" page |

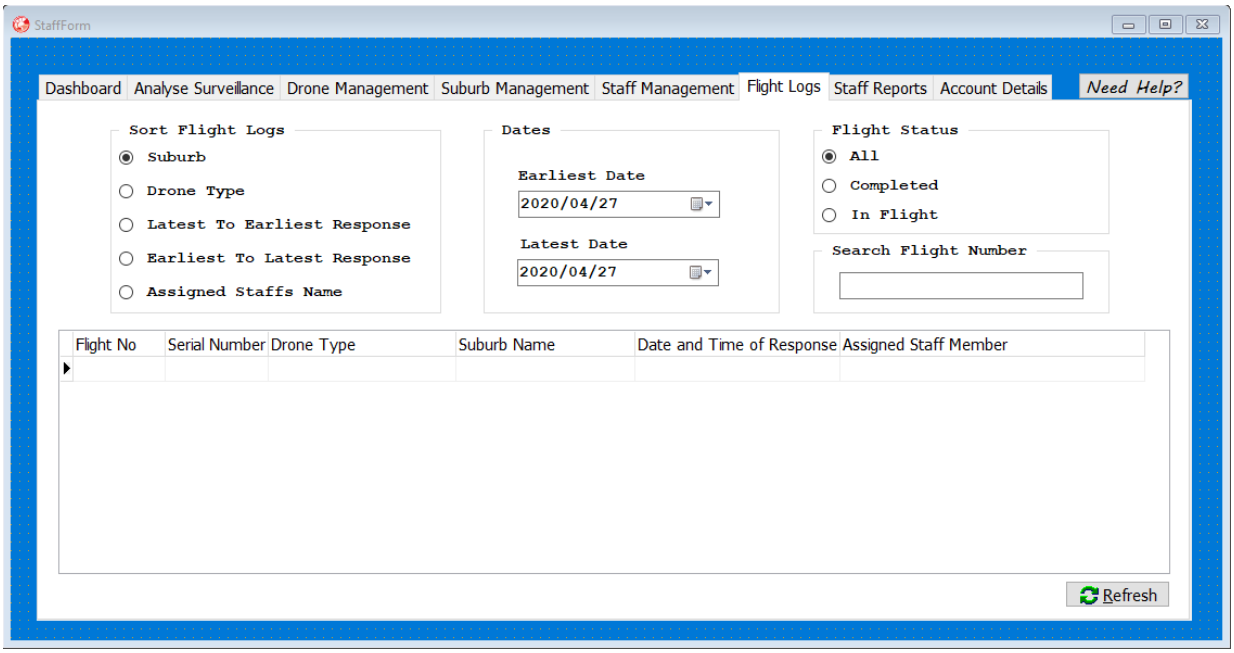
## Staff Management




- The users, who falls under the “Administrator”, will be able to add a new staff member.
- The user may add a staff member by entering all their details and ensuring that their email does not already exist as a staff member.
- The user is able to deactivate a staff members account if they see it fit to do so.

| Staff Management - Actions | |
|---|---|
| **Action** | **Description** |
| Deactivate/Reactivate Staff Member | • If the staff member account selected is activated, then:<br>  ○ The buttons caption will be changed to "Deactivate Staff Member"<br>  ○ The "Deactivate Staff Member" button will deactivate the selected staff members account<br>• If the staff member account selected is inactive, then:<br>  ○ The buttons caption will be changed to "Reactivate Staff Member"<br>  ○ The "Reactivate Staff Member" button will reactivate the selected staff members account |
| Add Staff Member | • The "Add Staff Member" button will add a staff member if:<br>  ○ The email entered is not connected to any other staff member |
| Retry | • The "Retry" button clears all the fields from the "Add Staff Member" section of the page |
| Refresh | • The "Refresh" button refreshes the data, showing all changes |
| Help | • The "Help" button will direct the user to a website with specified help for the "Staff Management" page |

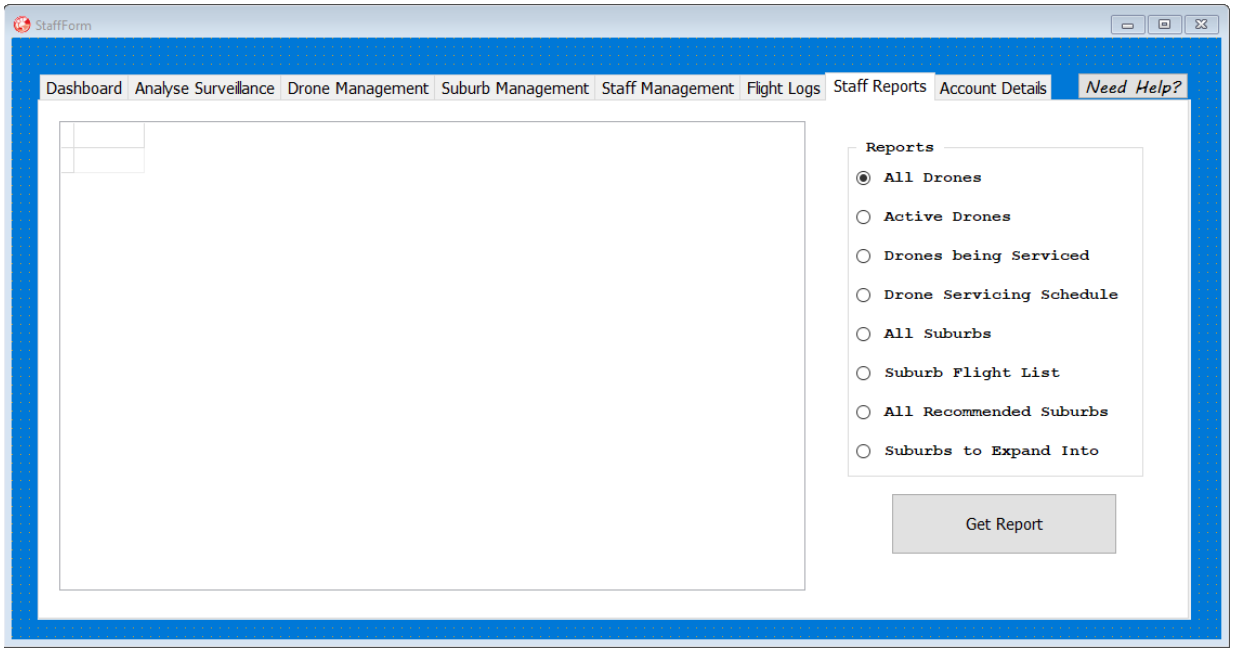
# Flight Logs




- The users are able to see all flight logs that have been requested by clients.
- The flight logs can be sorted by its all their information.
- A user may choose different dates, but the dates are auto set to the first and last flight ever requested by clients.
- The user can also search for a specific flight in order to see which staff member was assigned to it, encase a surveillance report is overdue for a user.

| Flight Logs - Actions | |
|---|---|
| **Action** | **Description** |
| Sort Flight Logs | • The “Sort Flight Logs” radio group will sort the flight logs by either:<br>○ Suburb<br>○ Drone Type<br>○ Latest to Earliest Response<br>○ Earliest to Latest Response<br>○ Assigned Staff Name |
| Earliest Date | • The “Earliest Date” date picker will ensure that all flight logs are greater than or equal to the chosen date |
| Latest Date | • The “Latest Date” date picker will ensure that all flight logs are less than or equal to the chosen date |
| Flight Status | • The “Flight Status” radio group will sort the flight logs status by either:<br>○ All Flights<br>○ Completed<br>○ In Flight (AKA Departed) |
| Search Flight Number | • The “Search Flight Number” edit will ensure that all flight logs flight numbers start with the numbers entered |
| Refresh | • The “Refresh” button refreshes the data, showing all changes |
| Help | • The “Help” button will direct the user to a website with specified help for the “Flight Logs” page |

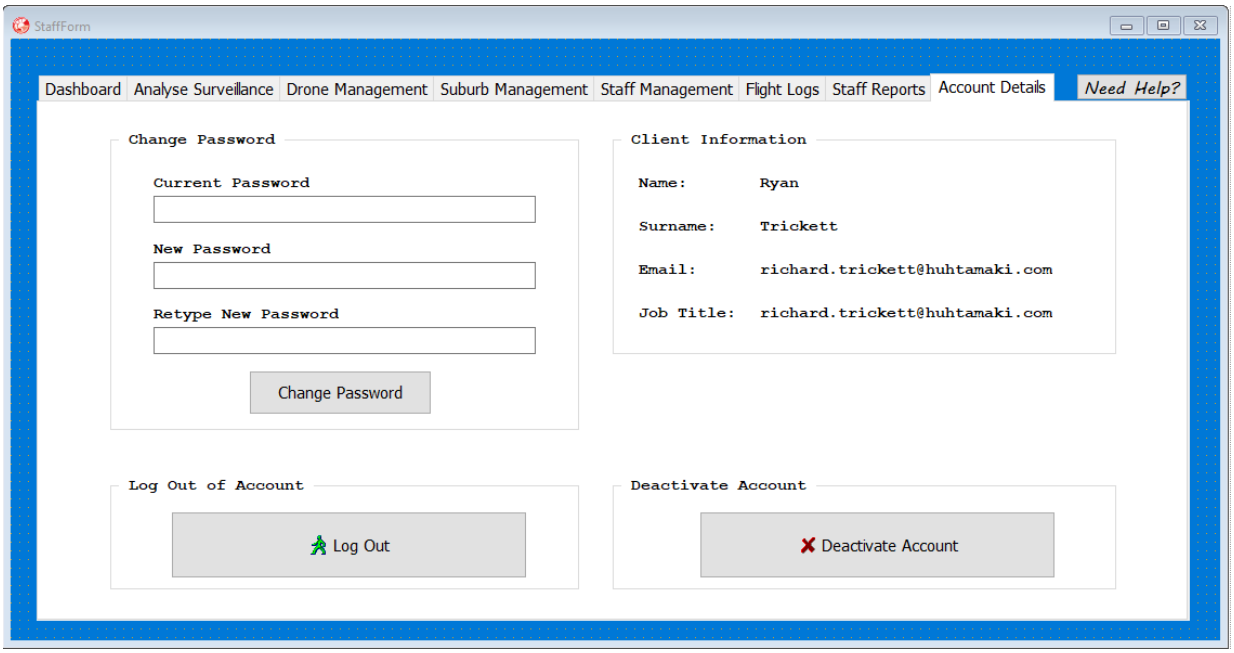
## Staff Reports




- The user can see reports created according to data stored within the database. The user can choose the report they wish to view, and the report is then displayed within the text box.
- All reports all for staff members to see statistics and data that will help them do their jobs.
- This page is read only and thus no data can be entered by a user.

| Reports - Actions | |
|---|---|
| **Action** | **Description** |
| Get Report | • The "Get Report" button displays a report according to the report selected within the "Report Type" radio group.<br>  ○ If the report is for "All Drones" then it will show the user a report with details of all drones<br>  ○ If the report is for "Active Drones" then it will show the user a report with the details of all active drones<br>  ○ If the report is for "Drones Being Serviced" then it will show the user a report with the details of all drones undergoing maintenance<br>  ○ If the report is for "Drone Servicing Schedule" then it will show the user a report with the number of flights until a drone must be serviced<br>  ○ If the report is for "All Suburbs" then it will show the user a report with the details of all suburbs<br>  ○ If the report is for "Suburb Flight List" then it will show the user a report with how many flights have been completed for each suburb ever<br>  ○ If the report is for "All Recommended Suburbs" then it will show the user a report with details of all the recommended suburbs<br>  ○ If the report is for "Suburb to Expand Into" then it will show the user a report with all the recommended suburbs who have enough interest to be viable |
| Help | • The "Help" button will direct the user to a website with specified help for the "Reports" page |

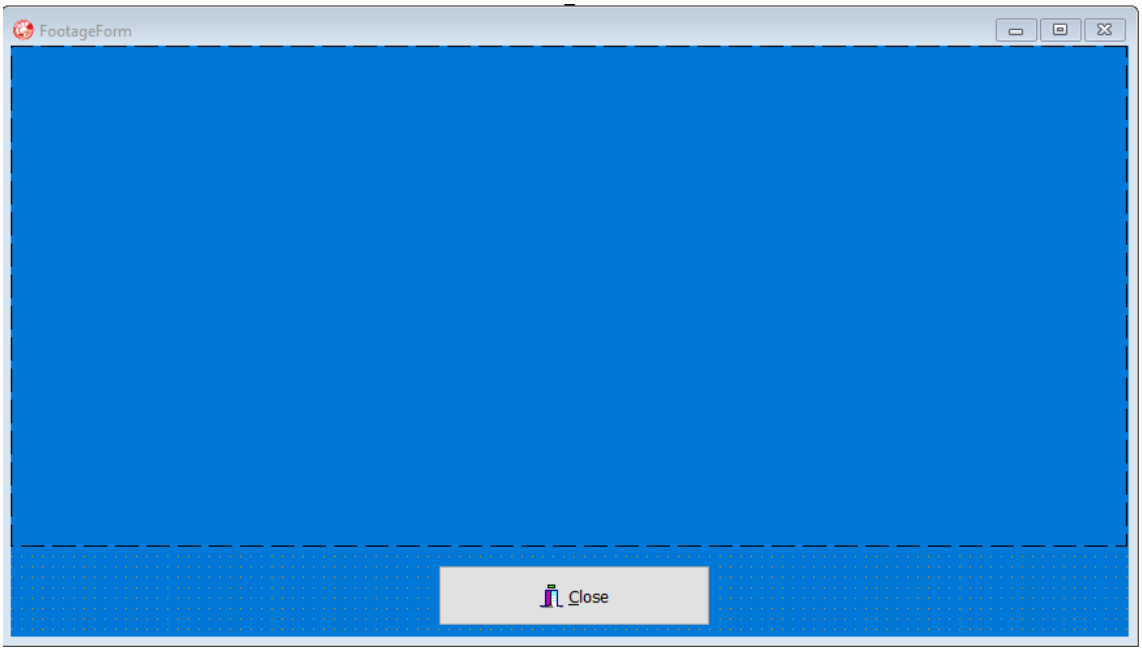
## Account Details




- The user is required to enter an email and password that is part of an account that has already been created. A user is to check the "Login as Staff Member" if they are a staff member of the company
- Any information that is entered is validated, and if any information is incorrect an appropriate error message will be displayed
- A user may display or hide their password if they wish to ensure that they have correctly entered their password
- If a user doesn't remember their password, they may access their account via a recovery code

| Account Details - Actions | |
|---|---|
| **Action** | **Description** |
| Change Password | • The "Change Password" button will change a user's password if:<br>  ○ The current password entered corresponds with the password connected to the user's account<br>  ○ The new password must be between 8 and 15 characters long and must follow a set of guidelines which is stated under of the Help Section<br>  ○ The new password must be the same as the retyped new password |
| Log Out | • The "Log Out" button will log the user out of the application<br>• The user will be redirected back to the login form |
| Deactivate Account | • The "Deactivate Account" button will deactivate a user's account |
| Help | • The "Help" button will direct the user to a website with specified help for the "Account Details" page |

# Footage Form

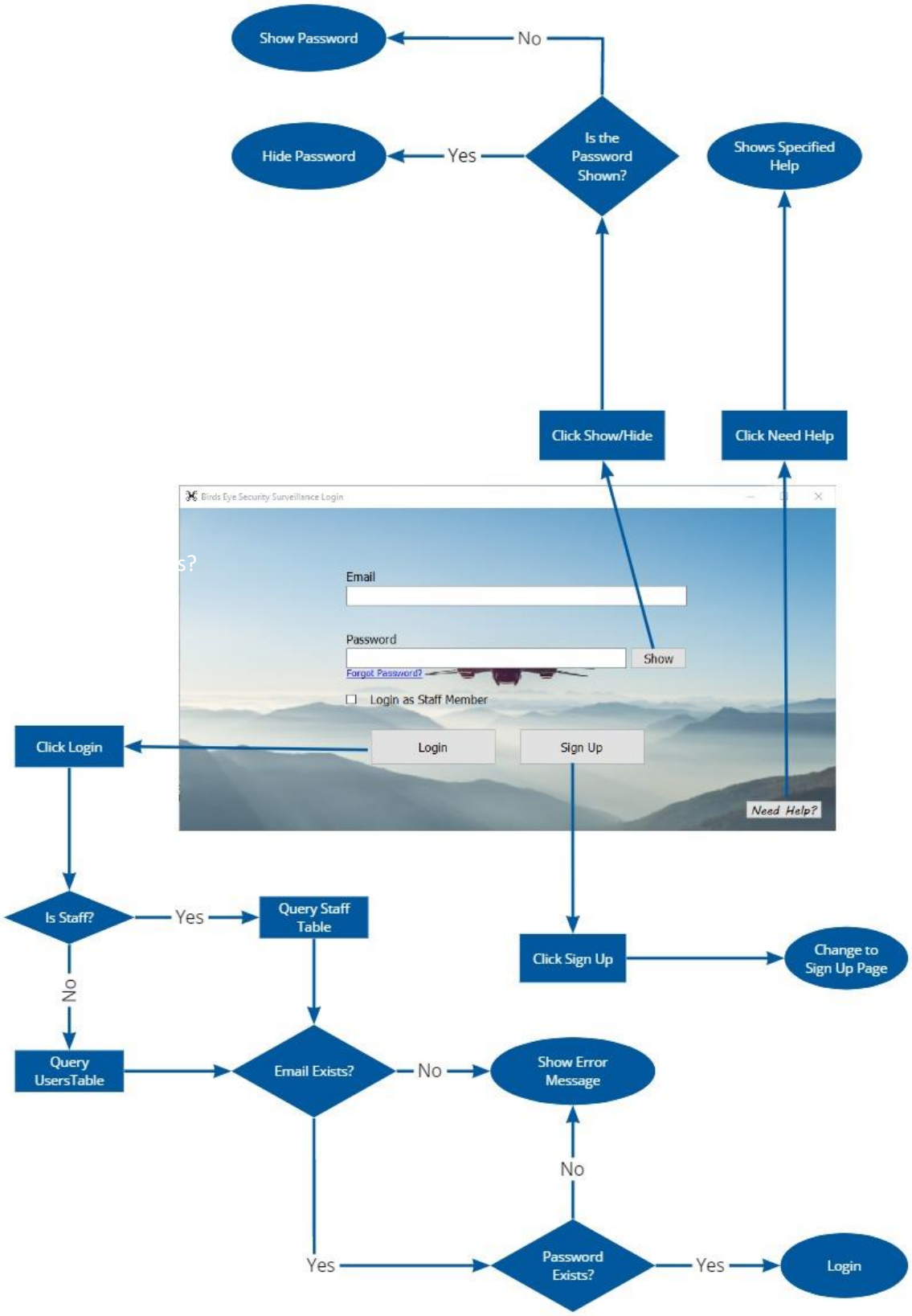



- The client can view surveillance footage of a specific flight in order to see the competition of their service request and they are able to see in detail the state of their property at the specific drone flight time
- A user can close the form after they are done in order to return to the main application

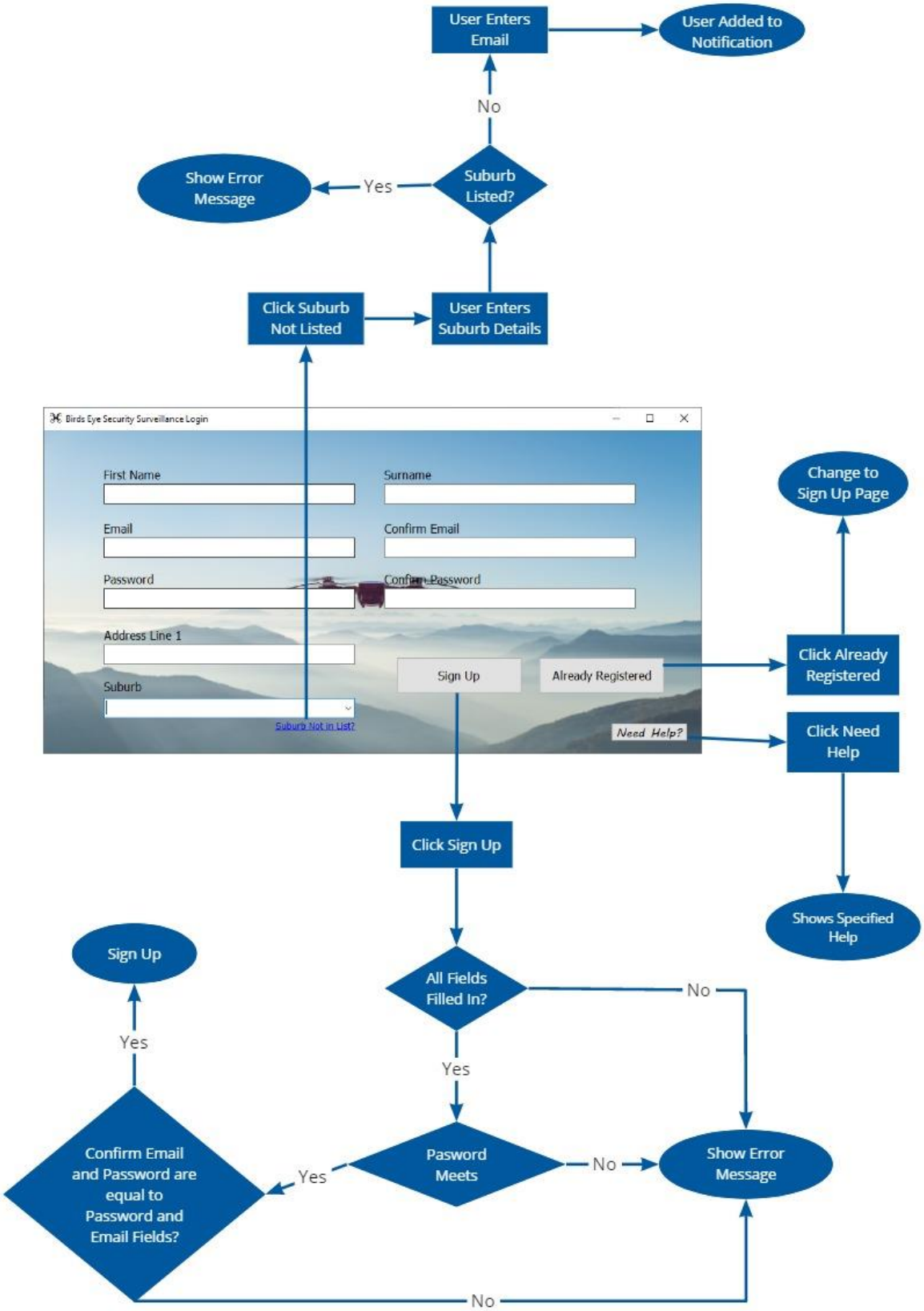
| Footage Form - Actions | |
|---|---|
| **Action** | **Description** |
| Close | • The “Close” button will close the footage form, allowing for a user to go back to the main client application |

# SEQUENCING

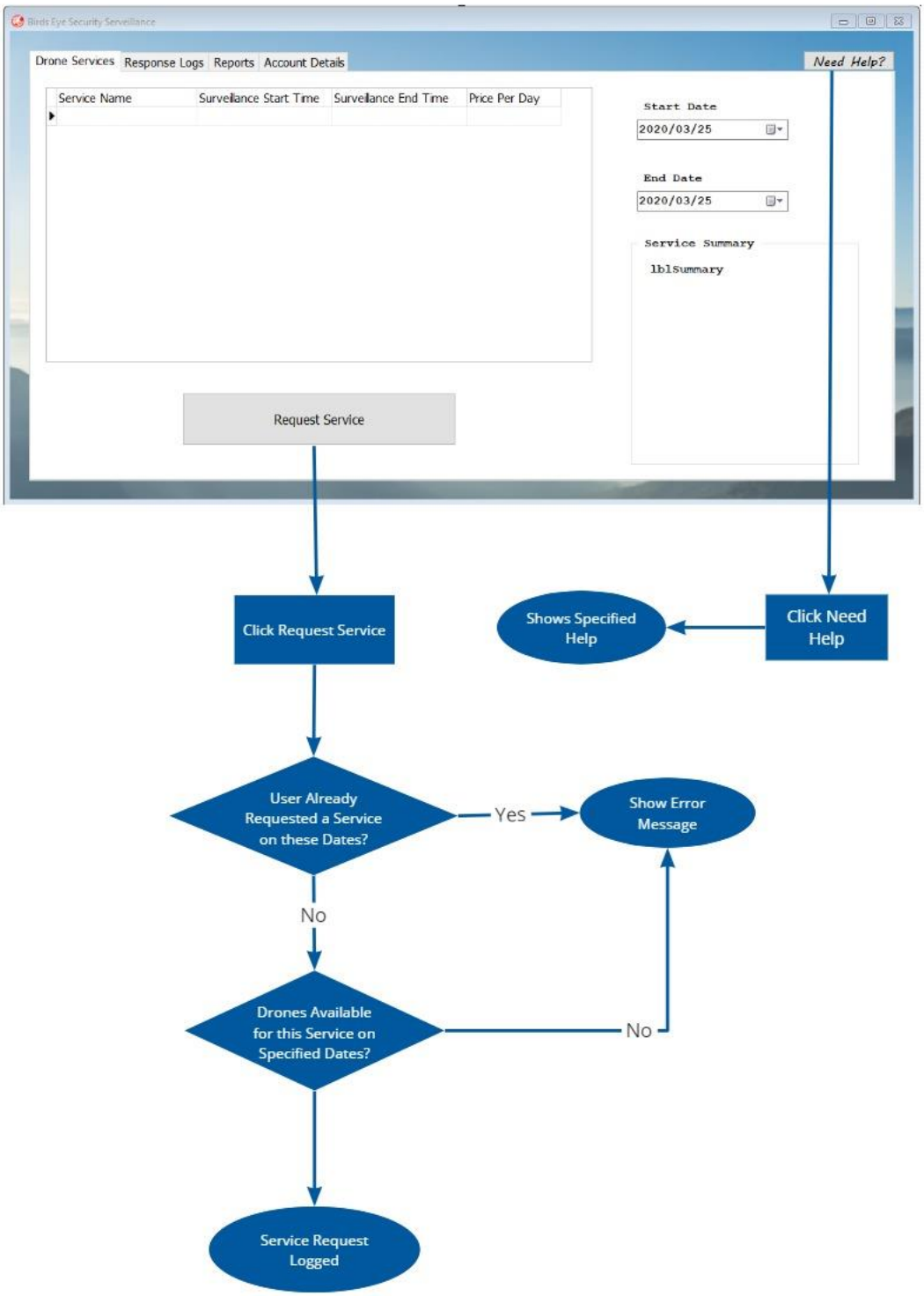
## Login



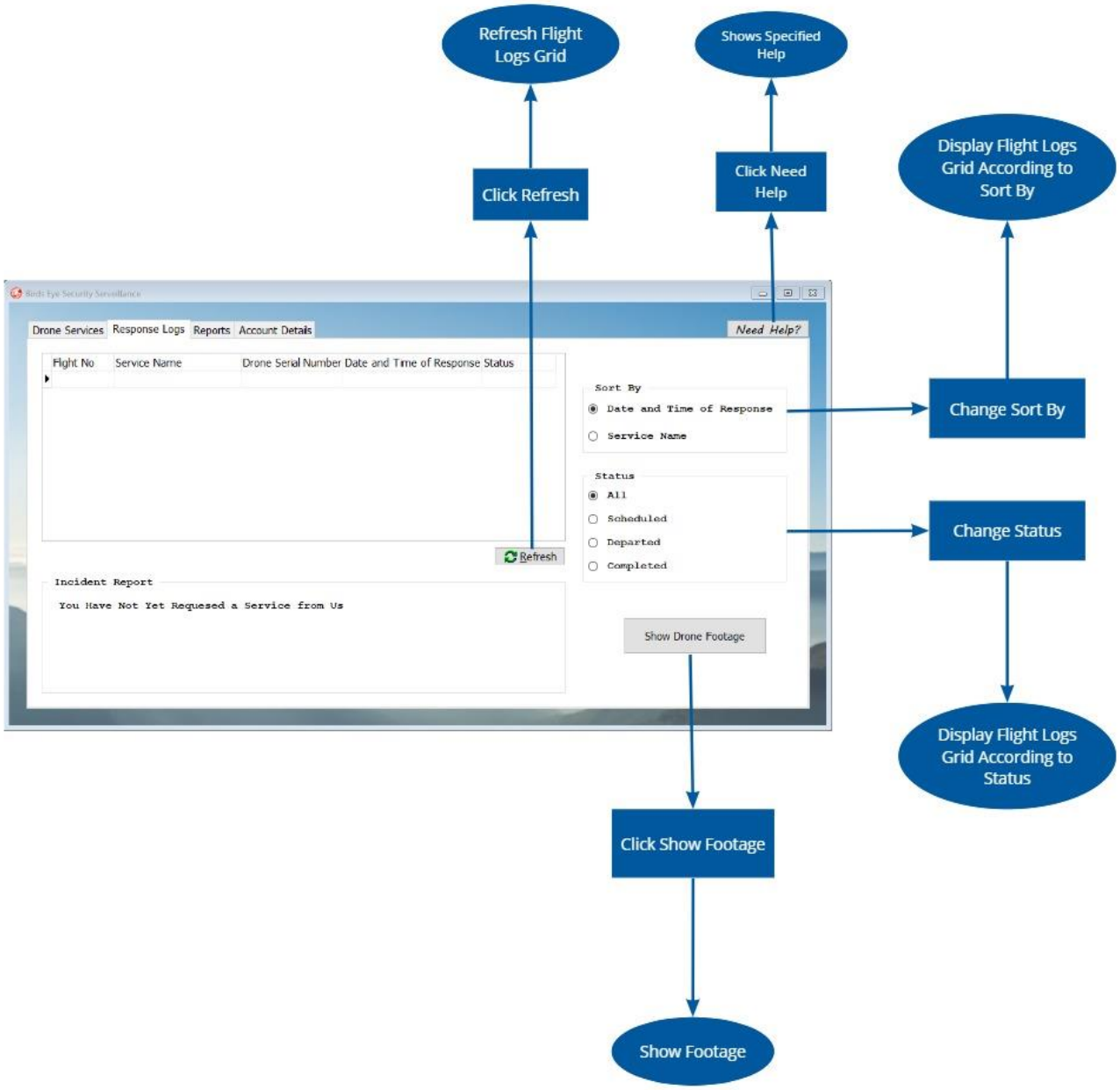
# Sign Up


User Enters Email
User Added to Notification
No
Show Error Message
Yes
Suburb Listed?
Click Suburb Not Listed
User Enters Suburb Details
Birds Eye Security Surveillance Login
First Name
Surname
Email
Confirm Email
Password
Confirm Password
Address Line 1
Suburb
Suburb Not in List?
Sign Up
Already Registered
Need Help?
Change to Sign Up Page
Click Already Registered
Click Need Help
Shows Specified Help
Click Sign Up
Sign Up
All Fields Filled In?
No
Yes
Yes
Confirm Email and Password are equal to Password and Email Fields?
Yes
Pasword Meets
No
Show Error Message
No

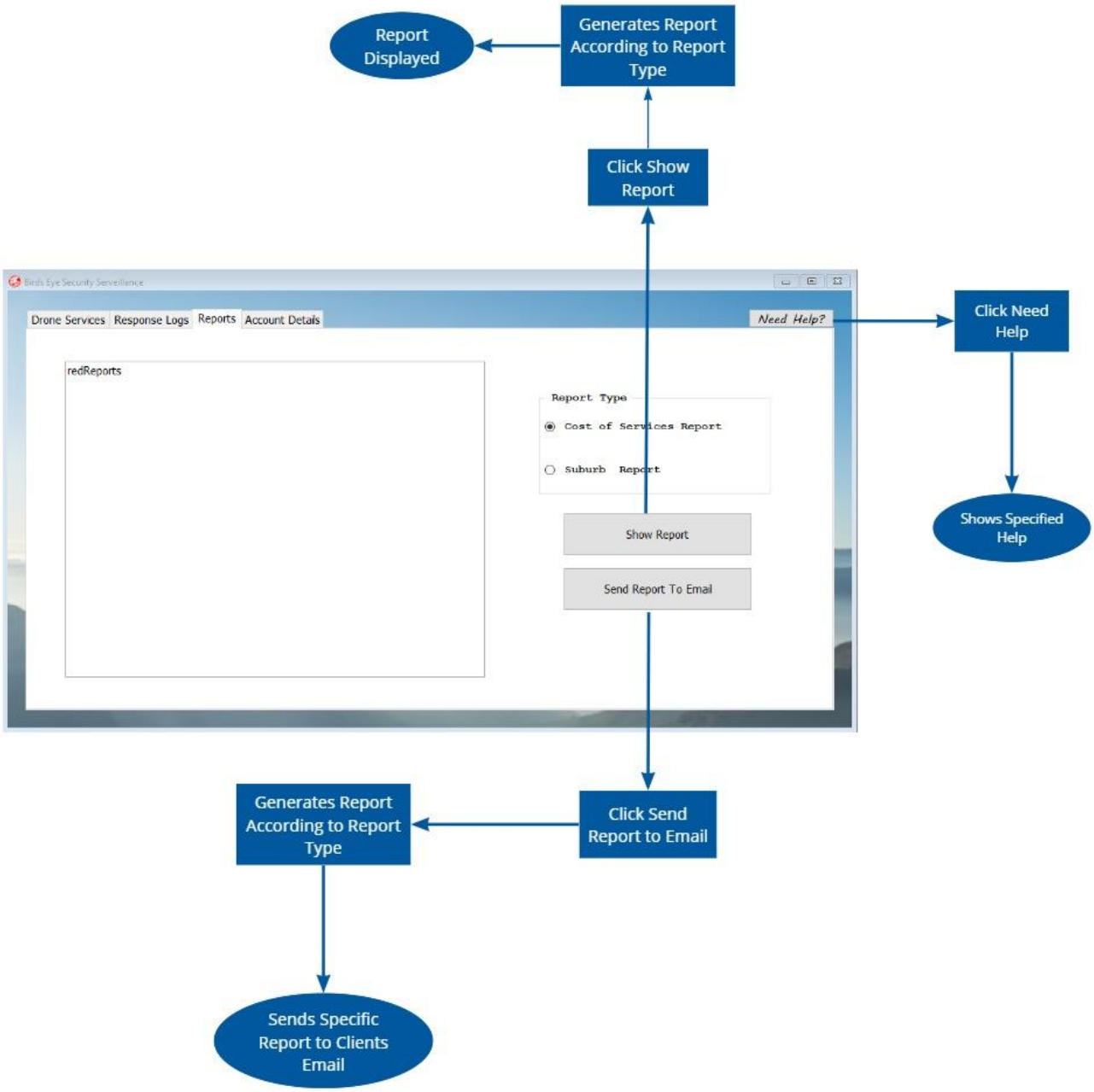
# Drone Services



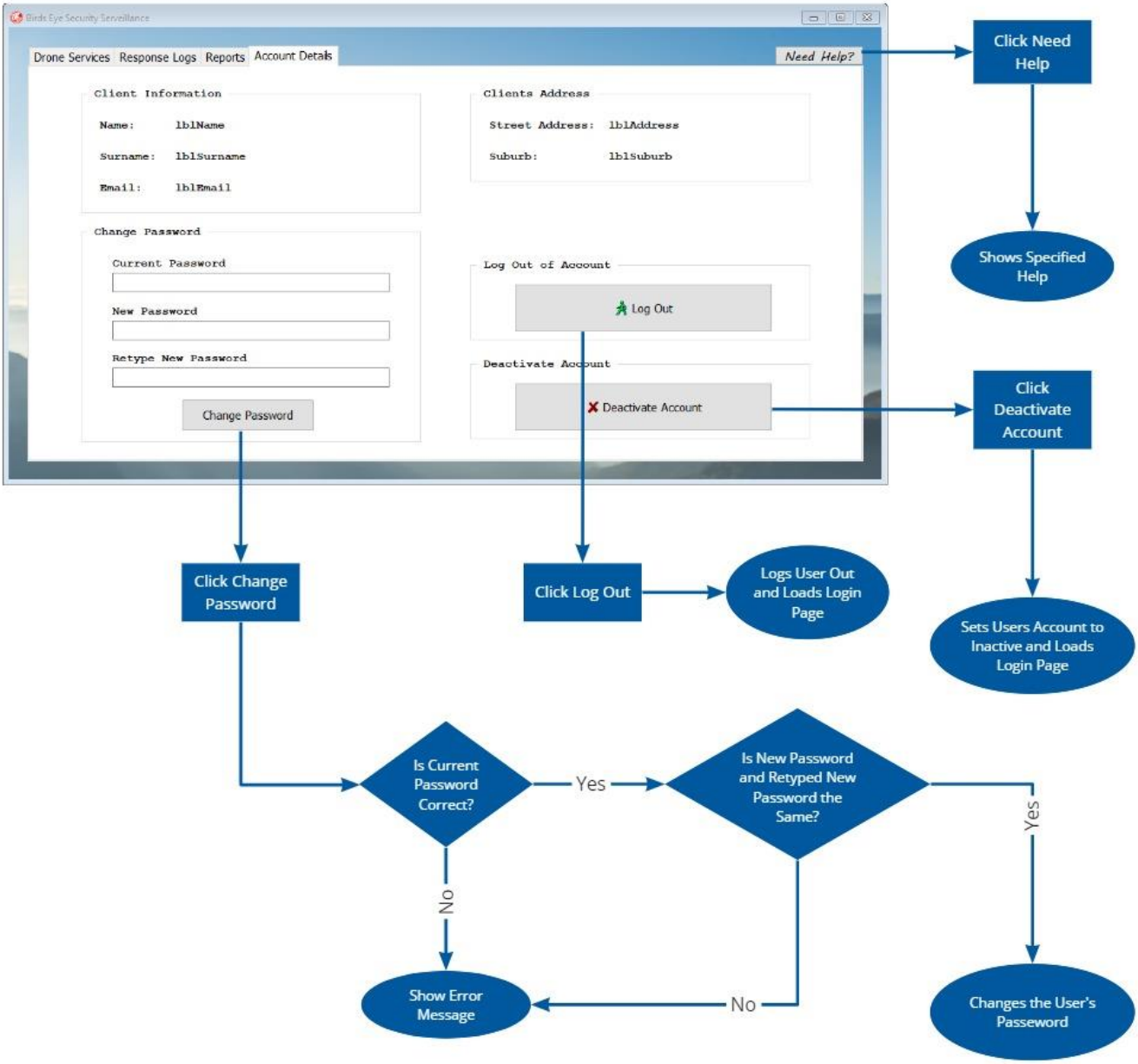
# Response Logs



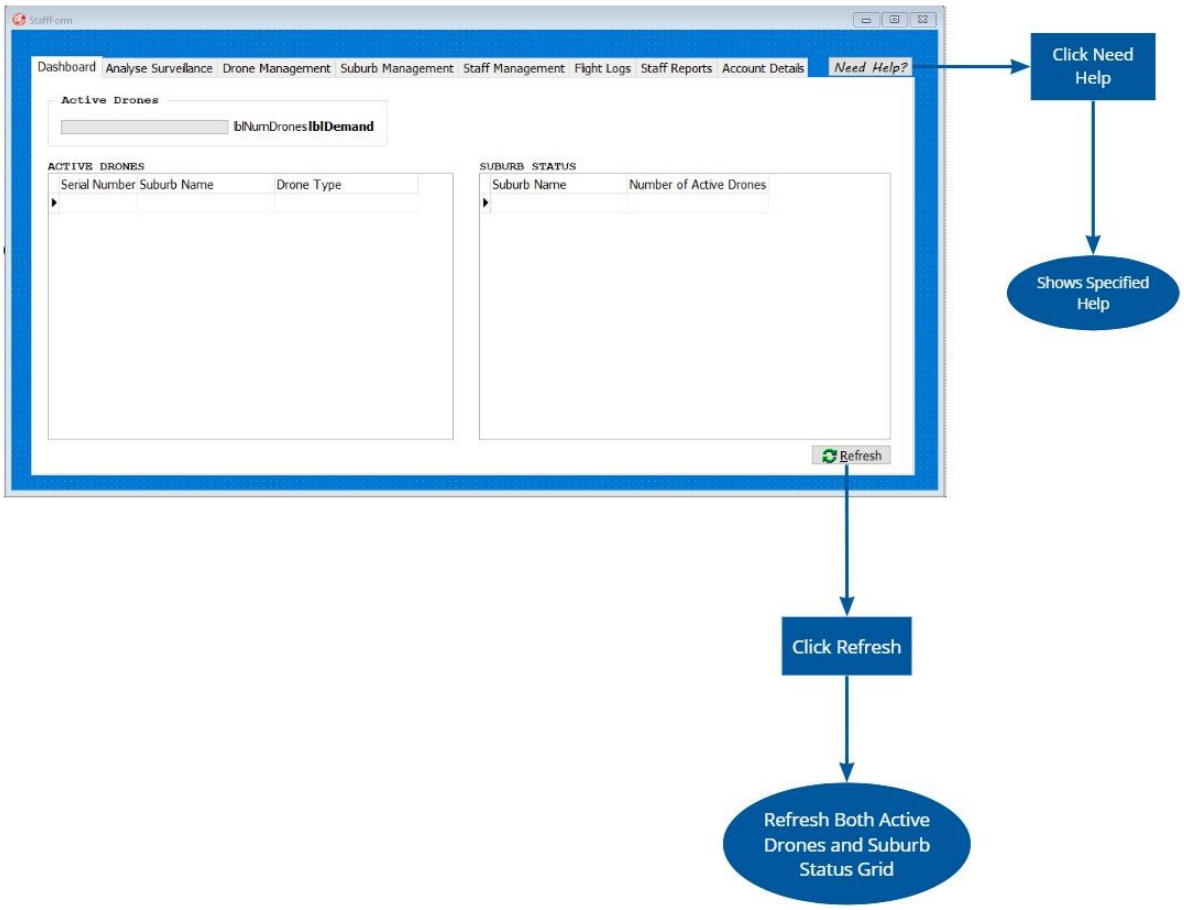
# Reports


Report Displayed
Generates Report According to Report Type
Click Show Report
Birds Eye Security Serveillance
Drone Services
Response Logs
Reports
Account Details
Need Help?
Click Need Help
redReports
Report Type
Cost of Services Report
Suburb Report
Show Report
Send Report To Email
Shows Specified Help
Generates Report According to Report Type
Click Send Report to Email
Sends Specific Report to Clients Email

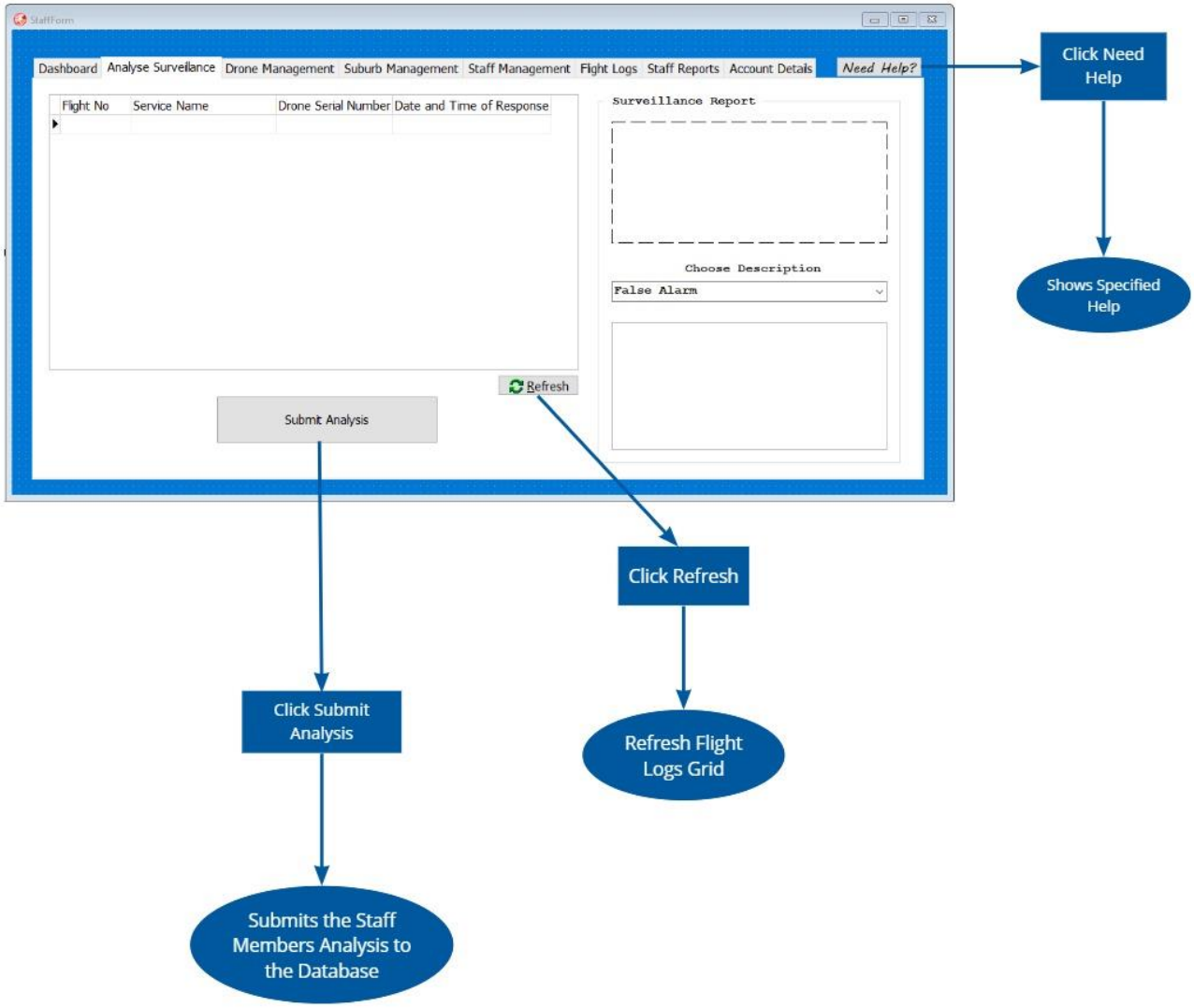
# Account Details



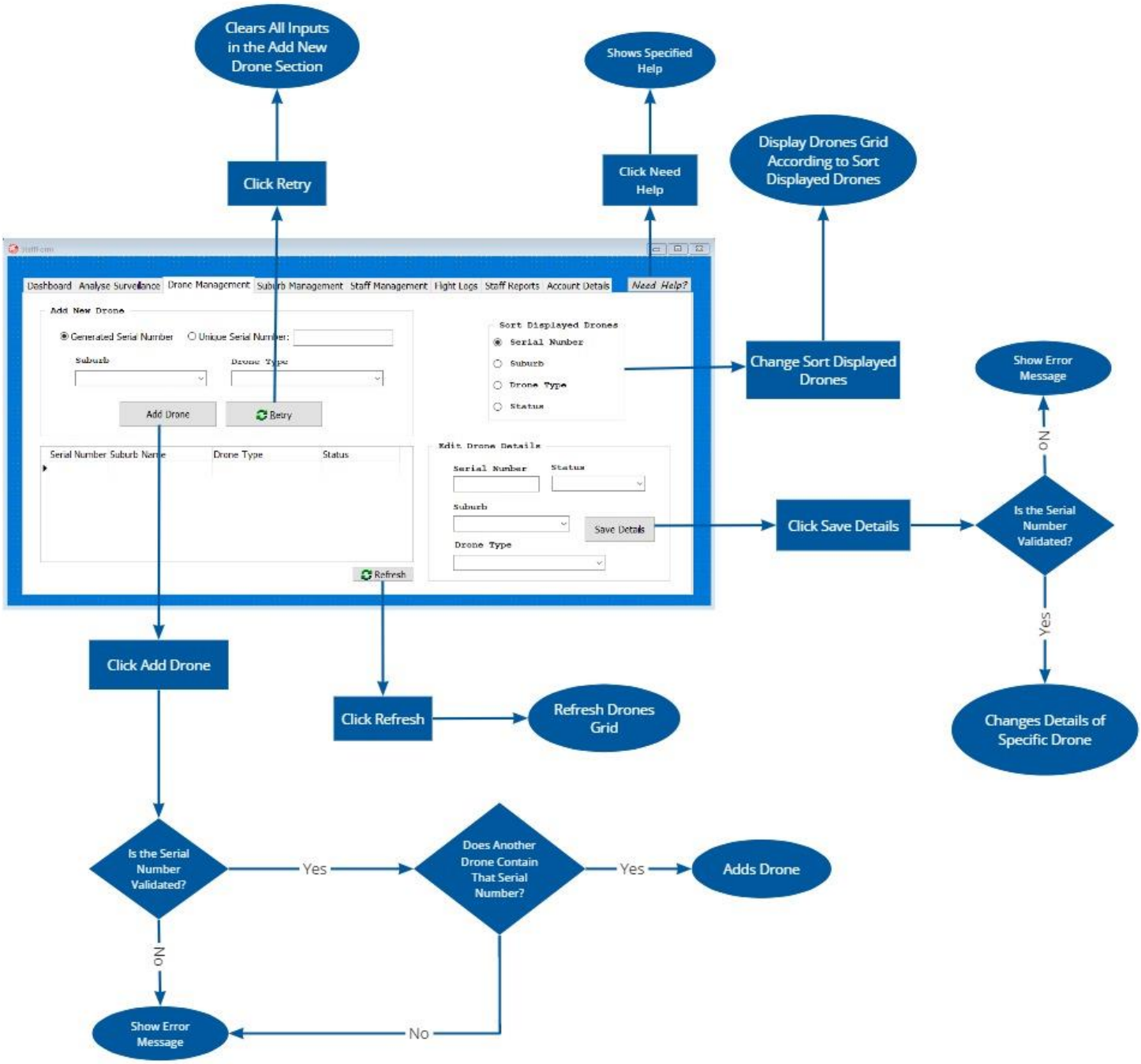
# Dashboard


StaffForm
Dashboard
Analyse Surveillance
Drone Management
Suburb Management
Staff Management
Flight Logs
Staff Reports
Account Details
Need Help?
Active Drones
lblNumDrones
lblDemand
ACTIVE DRONES
Serial Number
Suburb Name
Drone Type
SUBURB STATUS
Suburb Name
Number of Active Drones
Refresh
Click Need Help
Shows Specified Help
Click Refresh
Refresh Both Active Drones and Suburb Status Grid

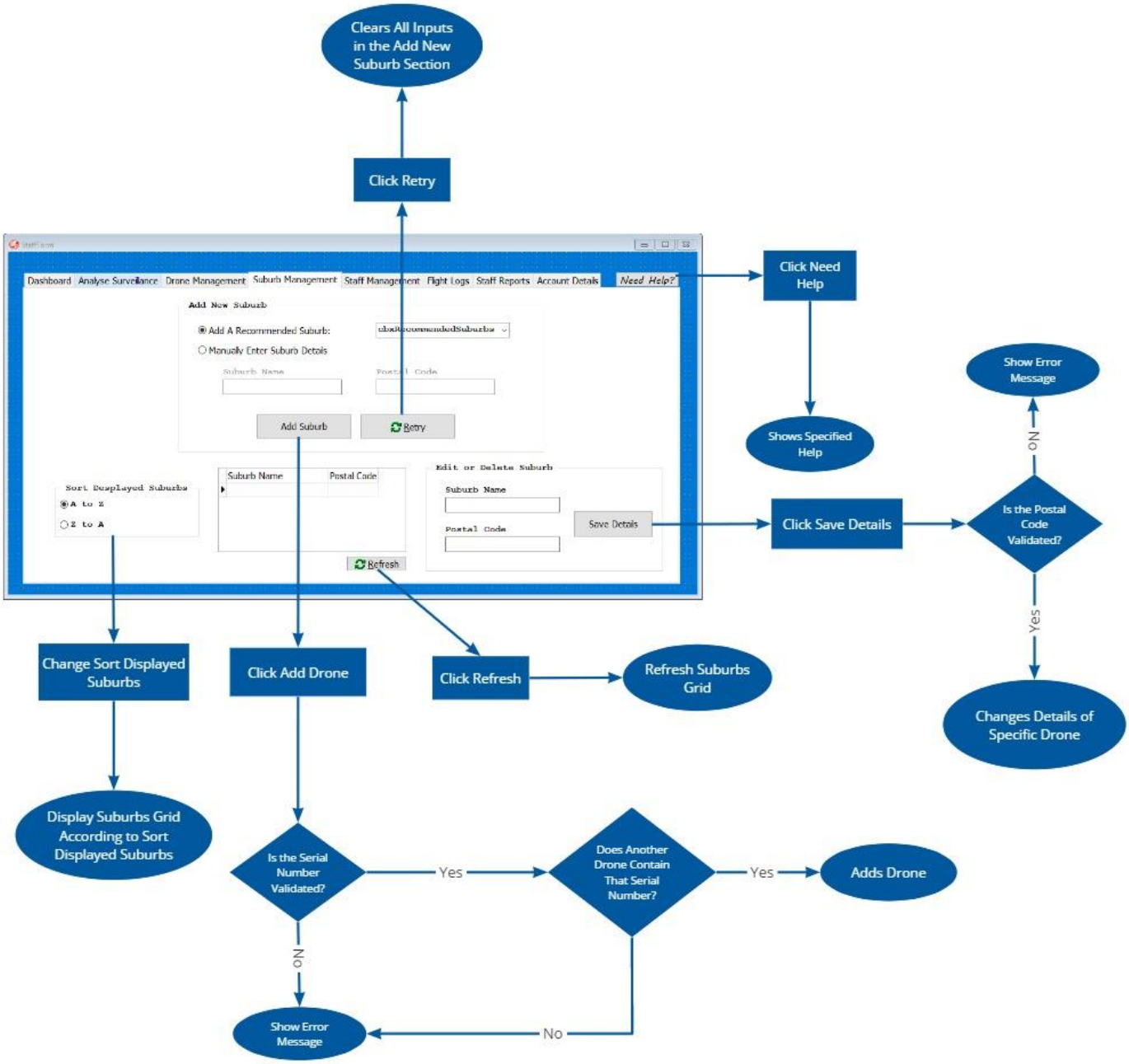
# Analyse Surveillance



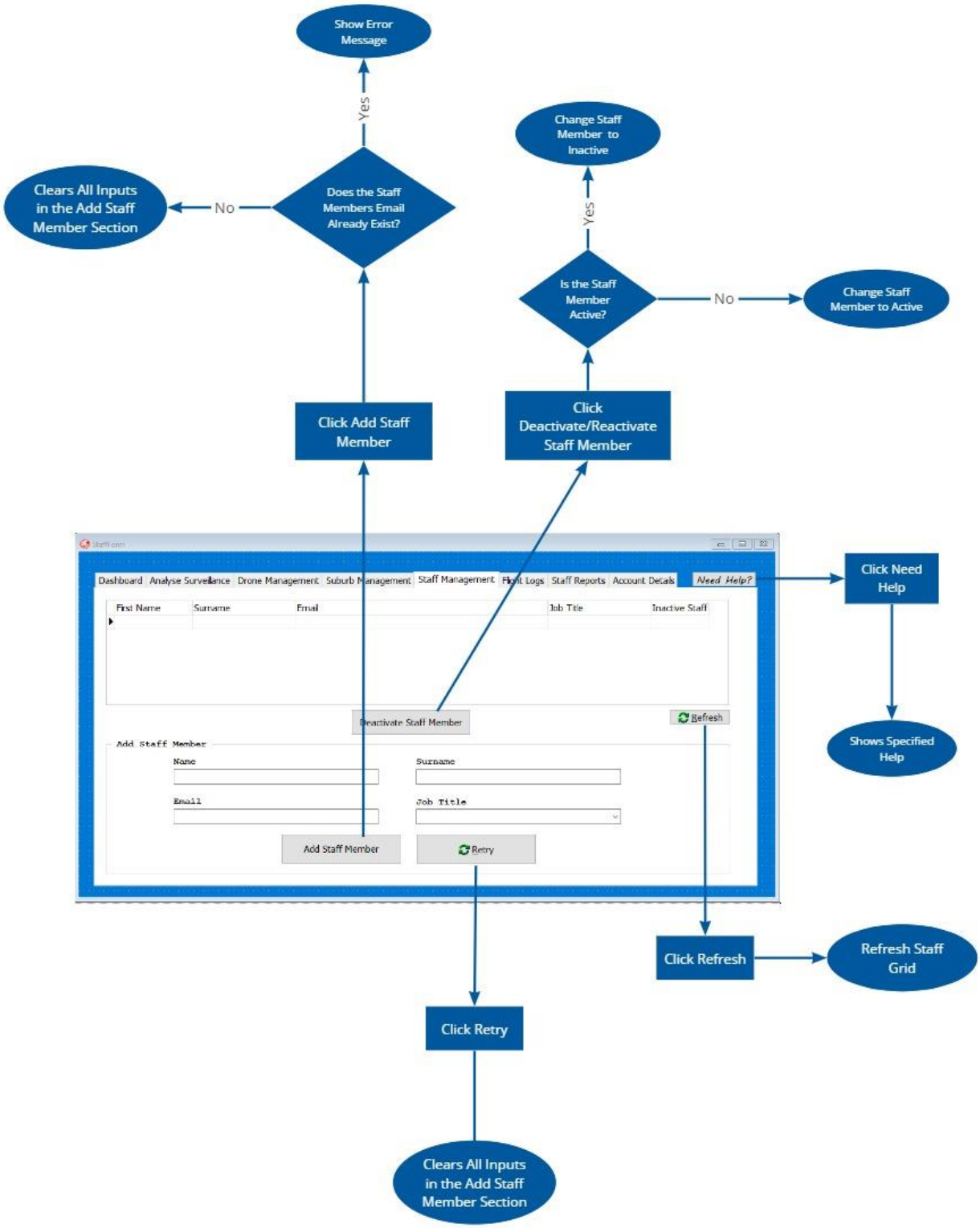
# Drone Management



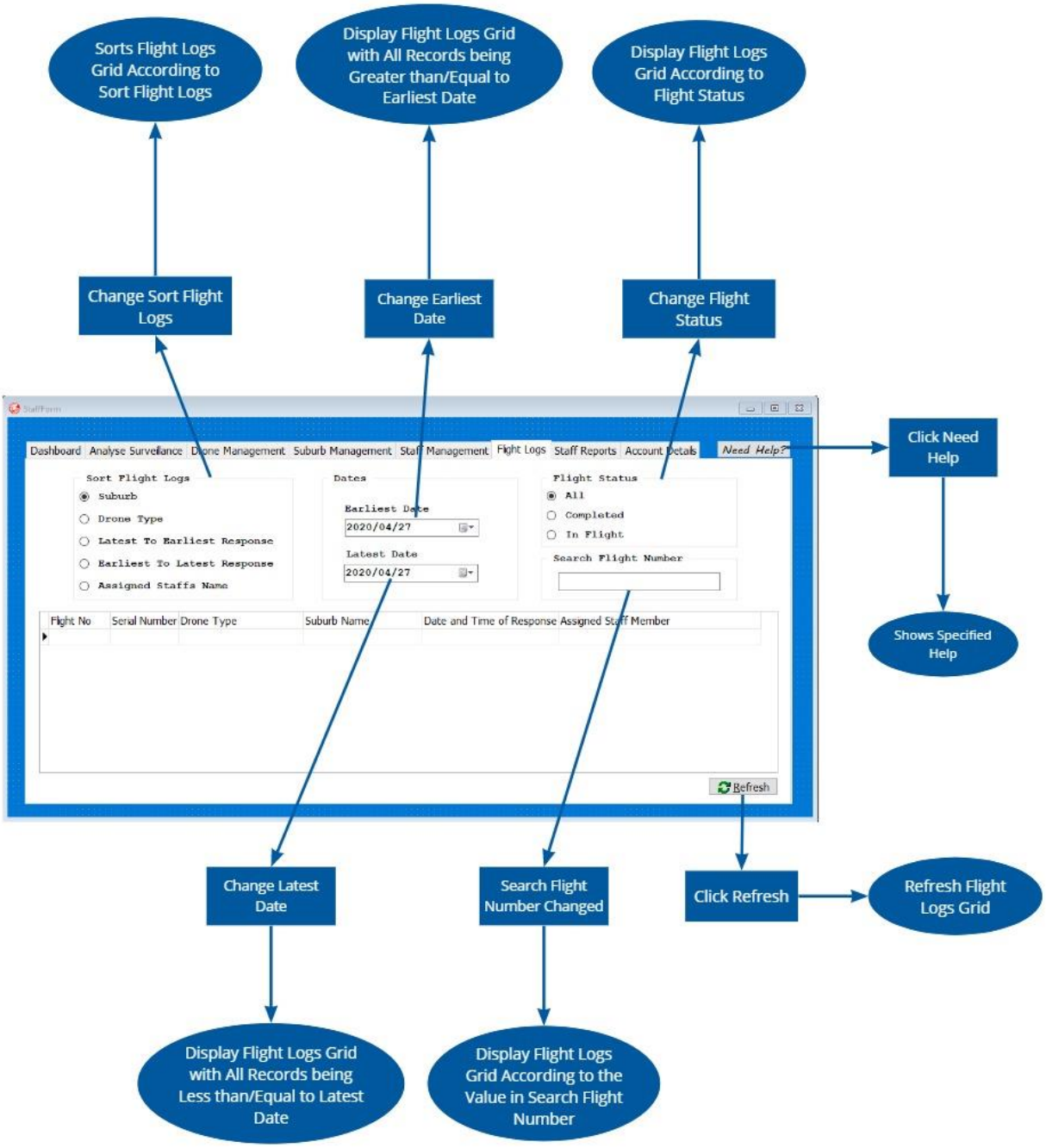
# Suburb Management



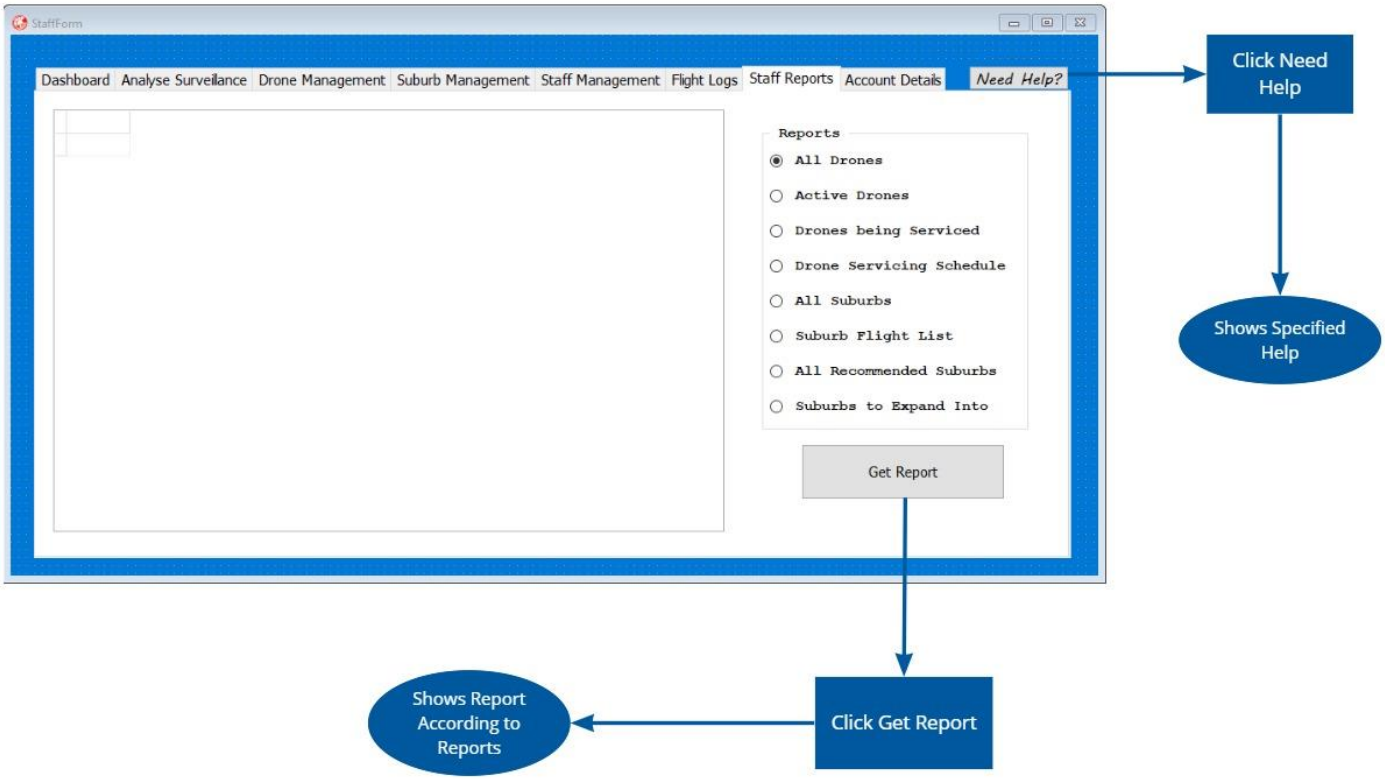
# Staff Management



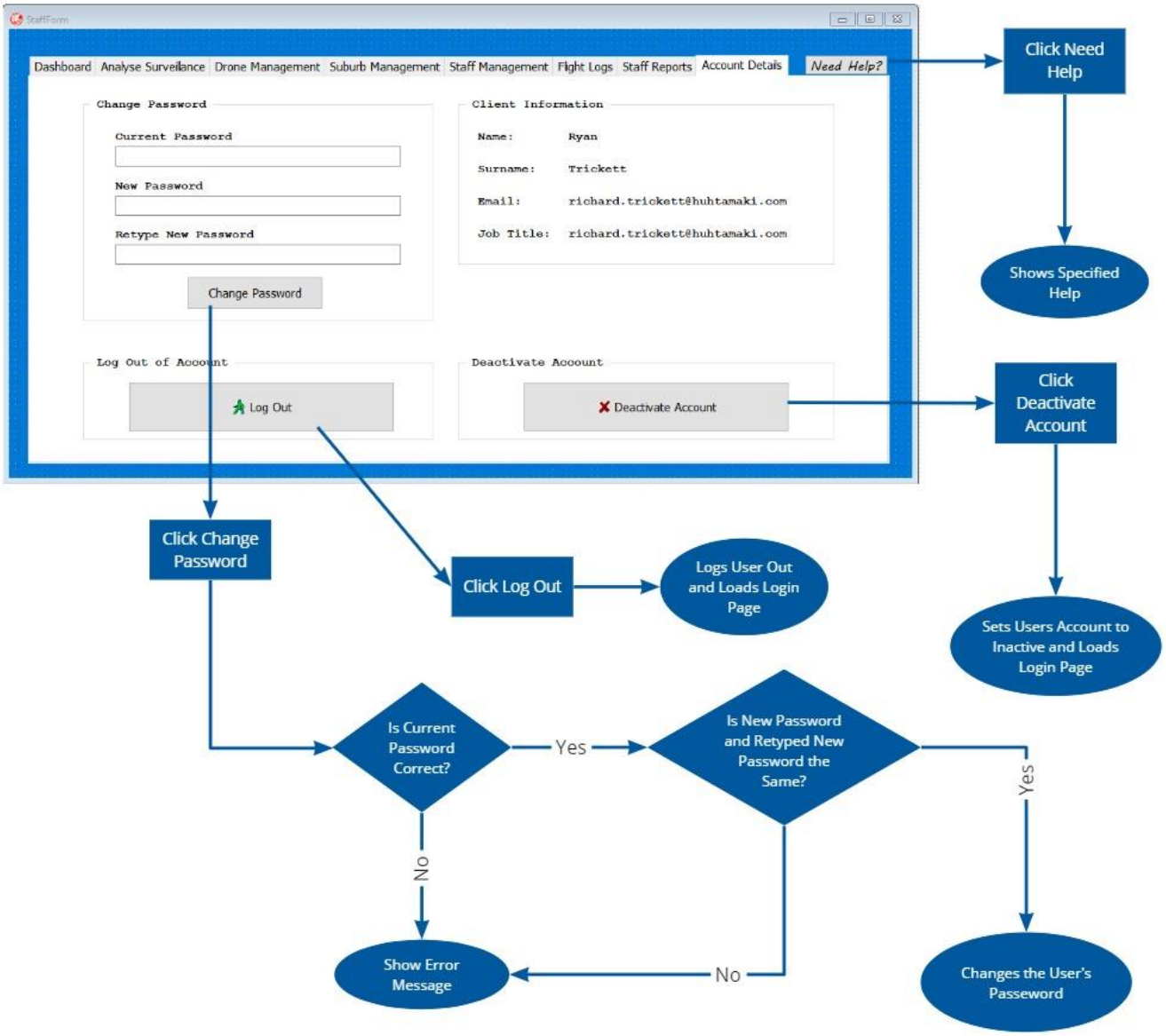
# Flight Logs


Sorts Flight Logs Grid According to Sort Flight Logs
Display Flight Logs Grid with All Records being Greater than/Equal to Earliest Date
Display Flight Logs Grid According to Flight Status
Change Sort Flight Logs
Change Earliest Date
Change Flight Status
Dashboard
Analyse Surveillance
Drone Management
Suburb Management
Staff Management
Flight Logs
Staff Reports
Account Details
Need Help?
Sort Flight Logs
Suburb
Drone Type
Latest To Earliest Response
Earliest To Latest Response
Assigned Staffs Name
Dates
Earliest Date
2020/04/27
Latest Date
2020/04/27
Flight Status
All
Completed
In Flight
Search Flight Number
Flight No
Serial Number
Drone Type
Suburb Name
Date and Time of Response
Assigned Staff Member
Refresh
Click Need Help
Shows Specified Help
Change Latest Date
Search Flight Number Changed
Click Refresh
Refresh Flight Logs Grid
Display Flight Logs Grid with All Records being Less than/Equal to Latest Date
Display Flight Logs Grid According to the Value in Search Flight Number

# Reports


StaffForm
Dashboard
Analyse Surveillance
Drone Management
Suburb Management
Staff Management
Flight Logs
Staff Reports
Account Details
Need Help?
Reports
All Drones
Active Drones
Drones being Serviced
Drone Servicing Schedule
All Suburbs
Suburb Flight List
All Recommended Suburbs
Suburbs to Expand Into
Get Report
Click Need Help
Shows Specified Help
Click Get Report
Shows Report According to Reports

# Account Details



# CLASS DESIGN

## TDBClass

This class manages interactions between the access database and the application, in order to obtain and store data within the tables located within the database.

TDynamicArray = array of string;

| TDBClass | Description |
|---|---|
| – fDbTbl : TADOQuery | The database accessor |
| + Create(NameOfDB : string)<br>+ DoSQL(TheSql : string)<br>+ RecordExist (TheSQL : string) : Boolean<br><br>+ GetDB : TADOQuery<br>+ ToStringHeadings(TheSQL, Delimiter: string) : string<br>+ ToString(TheSQL, Delimiter: string) : string<br>+ ToArray(TheSQL : string) : TDynamicArray | Creates the TDBClass object<br>Executes a SQL statement<br>Determines if a record exists for a SQL statement<br>Returns the fDbTbl variable<br>Returns the headings of a SQL statement in string format separated by a delimiter<br>Returns the records of a SQL statement in string format separated by a delimiter<br>Returns the records of a SQL statement in string format within a dynamic array |

## TStaff

This class stores all the information of a staff member who is currently signed into the application, in order to store their data for later use.

| TStaff | Description |
|---|---|
| – fID : string<br>– fName : string<br>– fSurname : string<br>– fEmail : string<br>– fPassword : string<br>– fJobTitle : string | The staff member's ID<br>The staff member's first name<br>The staff member's surname<br>The staff member's email address<br>The staff member's password<br>The staff member's job title name |
| + Create(Email, Password : string)<br>+ GetStaffID : string<br>+ GetJobTitle : string<br>+ GetName : string<br>+ GetSurname : string<br>+ GetEmail : string<br>+ GetPass : string | Creates the TStaff object<br>Returns the staff member's ID<br>Returns the staff member's job title name<br>Returns the staff member's first name<br>Returns the staff member's surname<br>Returns the staff member's email address<br>Returns the staff member's password |

## TServiceRequest

This class manages all requests from clients for drone surveillance services.

| TServiceRequest | Description |
|---|---|
| – fStartDate : TDate<br>– fEndDate : TDate<br>– fServiceName : string<br>– fPricePerDay : real<br>– fDetailsDictionary : TDictionary<string, string> | The start date of the service request<br>The end date of the service request<br>The service name of the service request<br>The price per day of the service request<br>The dictionary of the details linked to every service names |
| + Create<br>+ SetValues(StartDate, EndDate : TDate; ServiceName : string; PricePerDay : real)<br>+ UpdateServiceSummary : string<br>+ ConfirmServiceRequest : string<br>+ RequestService(UserID, Time : string) | Creates the TServiceRequest object<br>Changes the values of the object specified fields<br>Returns a summary of the service request<br>Returns a confirmation message<br>A service request is created within the database |

## TClient

This class stores all the information of a client who is currently signed into the application, in order to store their data for later use.

| TClient | Description |
|---|---|
| – fID : string<br>– fName : string<br>– fSurname : string<br>– fEmail : string<br>– fPassword : string<br>– fAddressLine1 : string<br>– fSuburbName : string | The client's ID<br>The client's first name<br>The client's surname<br>The client's email address<br>The client's password<br>The client's address line one<br>The client's suburb name |
| + Create(Email, Password : string)<br>+ GetUserID : string<br>+ GetName : string<br>+ GetSurname : string<br>+ GetEmail : string<br>+ GetAddressLine1 : string<br>+ GetSuburbName : string<br>+ GetPass : string | Creates the TClient object<br>Returns the client's ID<br>Returns the client's name<br>Returns the client's surname<br>Returns the client's email address<br>Returns the client's address line one<br>Returns the client's suburb name<br>Returns the client's password |

## TAccountDetails

This class manages all activities that allow a client or staff member to change details of their account.

| TAccountDetails | Description |
|---|---|
| – fLoginForm : TForm<br>– fIsClient : boolean | The TLoginForm object<br>The value showing if a client or staff member is logged in |
| + Create(LoginForm : TForm; IsClient : boolean)<br>+ ChangePassword(OldPass, NewPass, RetypedNewPass, PersonsPass, ID : string)<br>+ DeactivateAcc(ID : string; Form : TForm)<br>+ LogOut(Form : TForm) | Creates the TAccountDetails object<br>Changes the client/staff members password<br>Deactivates the client/staff members account<br>Logs the client/staff member out of the application |

## THelp

This class manages all help that a user might need while using the application.

| TAccountDetails | Description |
|---|---|
| – fHelpExtensionDictionary : TDictionary<string, string> | The dictionary of the help URLs linked to every tab sheet name |
| + Create<br>+ LoadHelp(PageName : string) | Creates the THelp object<br>Loads the help URL for a specific tab sheet |

## TEmails

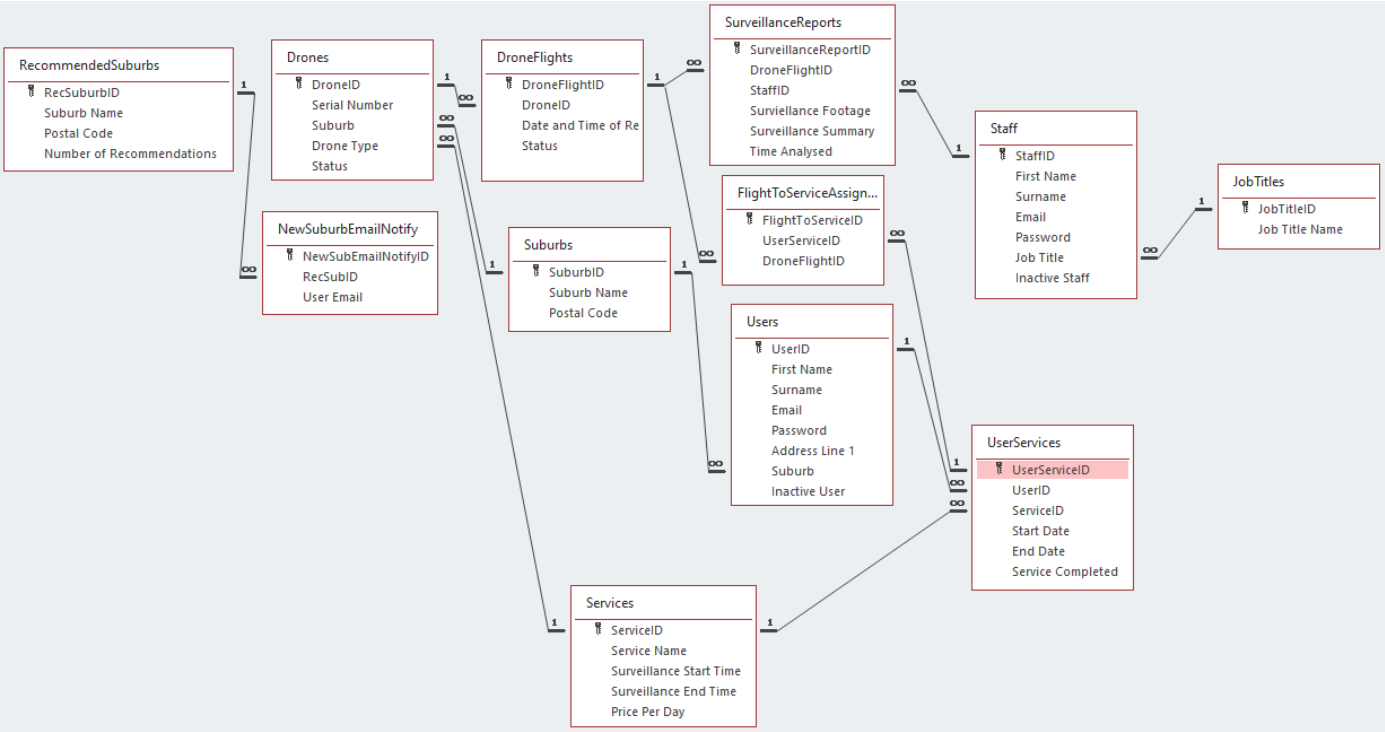
This class manages all emails that are sent to any user.

| TAccountDetails | Description |
|---|---|
| – fSSL : TIdSSLIOHandlerSocketOpenSSL<br>– fSTMP : TIdSMTP | The SSL object<br>The STMP object |
| + Create<br>+ SendEmail(RecipientName, EmailAddress, Subject, BodyMessage :string); | Creates the TEmails object<br>Sends an email to a user to a specified email and name, with a specified subject and body paragraph |

# PERSISTENT STORAGE DESIGN

## Database (Microsoft Access)

### Relationships




## Tables

### Drones

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| DroneID | AutoNumber | ID for specific drone |
| Serial Number | Short Text | A sequence of numbers representing a specific drone |
| Suburb | Number | The ID of the suburb the drone is assigned to |
| Drone Type | Number | The ID of the service which the drone is assigned to |
| Status | Short Text | Operational status of the drone (Active/Inactive/Being Repaired/Decomissioned) |

Sample Data:

| Drones | | | | |
|---|---|---|---|---|
| **DroneID** | **Serial Number** | **Suburb** | **Drone Type** | **Status** |
| 1 | 1221121 | 1 | 1 | Active |
| 2 | 2121232 | 1 | 1 | Active |
| 3 | 3522816 | 1 | 1 | Inactive |

## Users

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| UserID | AutoNumber | ID for specific client |
| First Name | Short Text | The name of the client |
| Surname | Short Text | The surname of the client |
| Email | Short Text | The email address of the client |
| Password | Short Text | The password of the client |
| Address Line 1 | Short Text | The address line one of the client |
| Suburb | Number | The ID of the suburb where the client lives |
| Inactive User | Yes/No | The status of the clients account |

Sample Data:

| Users | | | | | | | |
|---|---|---|---|---|---|---|---|
| **UserID** | **First Name** | **Surname** | **Email** | **Password** | **Address Line 1** | **Suburb** | **Inactive User** |
| 9 | Test | User | test@gmail.com | Password01 | 1 Test Road | 1 | No |
| 10 | Ryan | Trickett | ryanbasiltrickett@gmail.com | MyPassword9 | 1 Queens Street | 1 | Yes |
| 11 | John | Scheet | john.scheet@gmail.com | GreatMan29 | 9 President Road | 4 | No |

## Staff

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| StaffID | AutoNumber | ID for specific staff member |
| First Name | Short Text | The first name of the staff member |
| Surname | Short Text | The surname of the staff member |
| Email | Short Text | The email address of the staff member |
| Password | Short Text | The password of the staff member |
| Job Title | Number | The ID of the job title that the staff member performs |
| Inactive Staff | Yes/No | The status of the staff members account |

Sample Data:

| Staff | | | | | | |
|---|---|---|---|---|---|---|
| **StaffID** | **First Name** | **Surname** | **Email** | **Password** | **Job Title** | **Inactive Staff** |
| 3 | Master | Admin | masteradmin@birdseyesecurity.co.za | MasterAdmin01 | 2 | No |
| 5 | Head | Analyser | headanalyser@birdseyesecurity.co.za | HeadAnalyser | 1 | No |
| 6 | Vlad | Cocovich | vladcocovich@gmail.com | VladIsGood1 | 3 | Yes |

## Suburbs

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| SuburbID | AutoNumber | ID for specific suburb |
| Suburb Name | Short Text | The name of the suburb |
| Postal Code | Short Text | The postal code of the suburb |

Sample Data:

| Suburbs | | |
|---|---|---|
| **SuburbID** | **Suburb Name** | **Postal Code** |
| 1 | Kensignton | 2094 |
| 4 | Springs | 1559 |
| 6 | Pretoria CBD | 1672 |

## Services

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| ServiceID | AutoNumber | ID for specific service |
| Service Name | Short Text | The name of the service |
| Surveillance Start Time | Date/Time | The time at which the surveillance of the service will commence |
| Surveillance End Time | Date/Time | The time at which the surveillance of the service will be completed |
| Price Per Day | Currency | The price per day of the service |

Sample Data:

| Services | | | | |
|---|---|---|---|---|
| **ServiceID** | **Service Name** | **Surveillance Start Time** | **Surveillance End Time** | **Price Per Day** |
| 1 | Emergency Response | | | R15.00 |
| 2 | Night Surveillance | 20:00:00 | 23:59:59 | R30.00 |
| 3 | Early Bird Survey | 00:00:00 | 04:59:59 | R25.00 |
| 4 | Morning Surveillance | 05:00:00 | 07:59:59 | R10.00 |
| 5 | Full Day Survey | 08:00:00 | 19:59:59 | R50.00 |

## JobTitles

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| JobTitleID | AutoNumber | ID for specific job title |
| Job Title Name | Short Text | The name of the job |

Sample Data:

| JobTitles | |
|---|---|
| **JobTitleID** | **Job Title Name** |
| 1 | Surveillance Analyst |
| 2 | Administrator |
| 3 | Development Manager |

## UserServices

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| UserServiceID | AutoNumber | ID for specific user service request |
| UserID | Number | The ID of the client for the service request |
| ServiceID | Number | The ID of the service for the service request |
| Start Date | Date/Time | The date at which the service will commence for a client |
| End Date | Date/Time | The date at which the service will be completed for a client |

Sample Data:

| UserServices | | | | |
|---|---|---|---|---|
| **UserServiceID** | **UserID** | **ServiceID** | **Start Date** | **End Date** |
| 91 | 9 | 1 | 2020/06/30 | 2020/06/30 |
| 92 | 9 | 2 | 2020/06/30 | 2020/06/30 |
| 93 | 10 | 1 | 2020/06/30 | 2020/06/30 |
| 94 | 10 | 2 | 2020/06/30 | 2020/07/02 |

## FlightToServiceAssignment

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| FlightToServiceID | AutoNumber | ID for specific flight to service request |
| UserServiceID | Number | The ID of the service request |
| DroneFlightID | Number | The ID of the flight for the service request |

Sample Data:

| FlightToServiceAssignment | | |
|---|---|---|
| **FlightToServiceID** | **UserServiceID** | **DroneFlightID** |
| 56 | 91 | 73 |
| 57 | 92 | 74 |
| 58 | 93 | 75 |
| 59 | 94 | 76 |
| 60 | 94 | 77 |

## DroneFlights

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| DroneFlightID | AutoNumber | ID for a specific flight |
| DroneID | Number | The ID of the drone for the flight |
| Date and Time of Response | Date/Time | The date and time that the flight will depart/has departed |
| Status | Short Text | The status of the flight (Scheduled/Departed/Completed) |

Sample Data:

| DroneFlights | | | |
|---|---|---|---|
| **DroneFlightID** | **DroneID** | **Date and Time of Response** | **Status** |
| 73 | 5 | 2020/06/30 12:47:36 | Completed |
| 74 | 7 | 2020/06/30 20:00:00 | Scheduled |
| 75 | 4 | 2020/06/30 13:01:07 | Departed |
| 76 | 6 | 2020/06/30 20:00:00 | Scheduled |
| 77 | 7 | 2020/07/01 20:00:00 | Scheduled |
| 78 | 7 | 2020/07/02 20:00:00 | Scheduled |

## SurveillanceReports

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| SurveillanceReportID | AutoNumber | ID for specific surveillance report |
| DroneFlightID | Number | The ID of the flight for the surveillance report |
| StaffID | Number | The ID of the staff assigned to report on the surveillance |
| Surveillance Footage | Short Text | The footage collected during surveillance for the surveillance report |
| Surveillance Summary | Short Text | The summary of the surveillance report |
| Time Analysed | Date/Time | The time at which the surveillance report was submitted |

Sample Data:

| SurveillanceReports | | | | | |
|---|---|---|---|---|---|
| **SurveillanceReportID** | **DroneFlightID** | **StaffID** | **Surveillance Footage** | **Surveillance Summary** | **Time Analysed** |
| 47 | 73 | 5 | SurvFootage1.jpg | Waiting for Incident Report... | |
| 48 | 74 | 5 | SurvFootage3.jpg | No Activity | 2020/07/01 15:01:23 |
| 49 | 75 | 5 | SurvFootage1.jpg | Waiting for Incident Report... | |
| 50 | 76 | 5 | SurvFootage2.jpg | Waiting for Incident Report... | |
| 51 | 77 | 5 | | Drone Has Not Been Deployed | |
| 52 | 78 | 5 | | Drone Has Not Been Deployed | |

## RecommendedSuburbs

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| RecSuburbID | AutoNumber | ID for specific recommended suburb |
| Suburb Name | Short Text | The name of the recommended suburb |
| Postal Code | Short Text | The postal code of the recommended suburb |
| Number of Recommendations | Number | The number of people who have recommended the suburb |

Sample Data:

| RecommendedSuburbs | | | |
|---|---|---|---|
| **RecSuburbID** | **Suburb Name** | **Postal Code** | **Number of Recommendations** |
| 1 | Bedfordview | 2007 | 5 |
| 2 | Cape Town CBD | 6665 | 10 |
| 3 | Durban CBD | 4013 | 2 |

## NewSuburbEmailNotify

Design View:

| Field Name | Data Type | Description (Optional) |
|---|---|---|
| NewSubEmailNotifyID | AutoNumber | ID for specific suburb notification |
| RecSubID | Number | The ID of the recommended suburb |
| User Email | Short Text | The email of the user to be notified |

Sample Data:

| NewSuburbEmailNotify | | |
|---|---|---|
| **NewSubEmailNotifyID** | **RecSubID** | **User Email** |
| 1 | 1 | joesimson@gmail.com |
| 2 | 1 | ryan.trickett@reddam.house |
| 3 | 2 | peter@gmail.com |

# Text Files

All the text files hold important information which the application uses in order to perform action or provide information to the user. All these files are stored withing the project folder.

**Text Files:**

- ServiceDetails.txt
- PageAccess.txt
- HelpURLs.txt

**Text File Layout Examples**

All text files use a "#" as a delimiter in order to separate important information and allow for the permanent storage of relatable data across one line.

```
Login#https://birdseyesecuity.wixsite.com/application/login-page-help
Register#https://birdseyesecuity.wixsite.com/application/register-page-help
Drone Services#https://birdseyesecuity.wixsite.com/application/drone-services-help
Response Logs#https://birdseyesecuity.wixsite.com/application/response-logs-help
Reports#https://birdseyesecuity.wixsite.com/application/reports-help
Account Details#https://birdseyesecuity.wixsite.com/application/account-details-help
Dashboard#https://birdseyesecuity.wixsite.com/application/dashboard-help
```

```
Emergency Response#A tactical and response drone will be deployed in order to ensure your safety during a crime or dangerous situation.
Night Surveillance#A night drone will respond ever night in order to ensure your safety while you sleep.
Early Bird Survey#A drone specially fitted in order to fly during early morning sun rise will keep you sleep in the early hours.
Morning Surveillance#A drone which allows for surveillance flight just after sunrise will be deployed to your home to keep you safe.
Full Day Survey#Between the morning and evening a drone will be deplaoyed to keep your home safe while your are at work.
```

# EXPLANATION OF STORAGE DESIGN

## Database

The application uses a database as a form of permanent storage in order to store vital data. This includes user information and all flight logs performed (service requests).

Reasons why the application uses a database:

- Databases allows for the permanent storage of data
- Databases allow for easier querying and thus allow for more accurate retrieval of data
- Databases allow for organisation of data
- Databases can store large amounts of vital data
- Database querying allows for calculations and formatting in order to perform processes and thus lessening processing strain on the main application

## Text Files

The application uses text files in order to store text, which allows for permanent storage. This includes service details and help URLs.

Reasons why the application uses text files:

- Text files allow for easy corrections or updates to data
- Text files allow for storage of static data, and thus they cannot be changed within the application


# BIRDS EYE SECURITY SURVEILLANCE

Technical Document

Ryan Trickett
Gr12 Reddam House Bedfordview
Exam Number: 201112020858

# TABLE OF CONTENTS

# EXTERNALLY SOURCED CODE

## uDB_Source

This unit was given to me via Ms Aletta Foster, our Information Technology teacher. This allowed for me to build upon her basic code which was crucial in order to query the applications database.

## SSL Libraries

These libraries store code vital which is used in order to send emails with data to users. This means that without these libraries the use of email communication would be incapable.

## Wix Website Code

The website used for the help functionality was created via “wix.com” and thus all code for the website, from HTML to JavaScript has been created via their servers. This made it possible to create a help page simple and fast.

# EXPLANATION OF DIFFICULT ALGORITHM

## Request Service Algorithm

This algorithm allows for the client to request a service from the company. This means that the algorithm must schedule flight and ensure drones are available for the specified days.

1. The algorithm queries the database in order to return the ServiceID of a specific record according to the services name.
2. The algorithm queries the database in order to return the ServiceID of a specific record according to the services name.
3. The algorithm queries the database in order to ascertain whether the user already has a service request that may overlap with the service they wish to request. This querying looks at the start and end dates, service type of the service request.
4. The algorithm queries the database in order to ascertain whether there are available drones for the service they wish to request. This querying looks at the start and end dates, service type and suburb of the service request.
5. The algorithm then records a service request, which relates a specific user to a service between specified dates.
6. The algorithm then checks whether the service request is for an emergency service (which the ONLY instant service).
7. The algorithm then schedules a flight for everyday which the service has been requested for. Thus, recording each scheduled flight within the database.
8. A message is created in order to tell the user their service request has been successful (as long as there weren't any overlaps or unavailable drones)
9. Error messages are created if a user's service overlaps with another service (of the same service name) or no drones are available for the specified days.

# Login Algorithm

This allows users to log into the application in order to access all the applications features. This therefore is the starting algorithm, which secures the application.

1. The algorithm checks if the IsStaff parameter is true in order to create verified SQL statements for the different database tables. The SQL statement uses the email parameter.
2. The algorithm checks if the record exists and thus, if it returns as true, the email exists in the database.
3. The algorithm checks if the IsStaff parameter is true in order to create verified SQL statements for the different database tables. The SQL statement uses the password parameter.
4. The algorithm queries the database in order to get the password connected to the specified email address.
5. The algorithm checks if the password is the same as the one entered, including case sensitivity. A specific form pertaining to the type of user is also instantiated and the instantiated forms setup method is called in order to allow for variable initialisation.
6. The created form is shown, and the Login Form is hidden.
7. Error messages are created if a user's password or email is incorrect.

# Complete Flights Algorithm

This algorithm sets flights to completed as well as assigns specific surveillance footage to completed drone flights. (This algorithm shows simulation)

1. The algorithm queries the database to find all the flights which flight end date and time have passed the current time on the user's device. All records are placed within an array.
2. If the length of the array is 0 then nothing happens, but if the length is greater than 0 then the algorithm continues.
3. The algorithm updates all records in the database which flight end date and time have passed the current time on the user's device.
4. The algorithm then assigns a random surveillance footage file path to each completed drone flight.

# ADVANCED SKILLS

## Indy Components

These components are used in the application in order to create a secure connection in order to send data to users via email. This uses googles STMP services in order to send emails from a gmail address associated with Birds Eye Security.

## Multiple Forms

This allows for more than one form to be running at a given moment. Allowing for application to process different threads at the same time. This also allows for the dynamic creation of forms, thus meaning that the application is more optimised.

## DBGrid Formatting

By formatting the DBGrid's different columns allows for the application to look presentable even when SQL statements are changed during the applications processes. This means that data from the database is always easily visible for users.

## Dictionaries

This datatype is used in order to store data which can be correlated to a string value. This allows for easy acquiring of data through the use of names of objects or service types, instead of having to convert them to an integer to acquire data through a array.

## Timers

This allows for up to date departing of drones and completion of flights. This ensures all data showed to the user is as up to date as possible and therefore the user will always be kept up to date with the latest information from the applications database.

## Dynamic Instantiation of Components

By creating components during the running of the application it allows for optimisation as unused components are not instantiated until they are needed in order to perform an action or display specific data.

## Application Setup File

This file allows the entire application to be stored in one file and then the data and files needed in order to run the application are extracted from the file once the application setup (.exe) file is opened. Once the files have been set up the setup file is no longer needed.

## ShellAPI

This allows for the application to open the users default browser in order to launch a help URL. Thus this helps provide users with interactive web help.

BIRDS EYE SECURITY SURVEILLANCE

# BIRDS EYE SECURITY SURVEILLANCE

Coding Document

Ryan Trickett
Gr12 Reddam House Bedfordview
Exam Number: 201112020858

# TABLE OF CONTENTS

# CODING

## Forms

### Login Form

```
unit frmLogin;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, uLoginFormManager, jpeg, uAutoDroneDeployment,
  uHelp, uLibrary;

type
  TLoginForm = class(TForm)
    imgBackground: TImage;
    pnlLogin: TPanel;
    pnlRegister: TPanel;
    ledEmail: TLabeledEdit;
    ledPassword: TLabeledEdit;
    btnLogin: TButton;
    btnSignUp: TButton;
    cbxStaff: TCheckBox;
    btnShowHide: TButton;
    ledFirstName: TLabeledEdit;
    ledSurname: TLabeledEdit;
    ledPasswordSignUp: TLabeledEdit;
    cbxSuburbs: TComboBox;
    ledPasswordConfirm: TLabeledEdit;
    ledEmailSignUp: TLabeledEdit;
    ledEmailConfirm: TLabeledEdit;
    lblSuburbs: TLabel;
    ledAddressLine1: TLabeledEdit;
    btnRegister: TButton;
    btnAlreadyRegistered: TButton;
    lblRecommenedSuburb: TLabel;
    tmrAuto: TTimer;
    btnHelpLogin: TButton;
    btnHelpRegister: TButton;
    lblForgotPass: TLabel;
    procedure btnShowHideClick(Sender: TObject);
    procedure btnLoginClick(Sender: TObject);
    procedure btnSignUpClick(Sender: TObject);
    procedure btnAlreadyRegisteredClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure btnRegisterClick(Sender: TObject);
    procedure lblRecommenedSuburbClick(Sender: TObject);
    procedure ledPasswordKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure FormShow(Sender: TObject);
    procedure tmrAutoTimer(Sender: TObject);
    procedure btnHelpLoginClick(Sender: TObject);
```

```
    procedure btnHelpRegisterClick(Sender: TObject);
    procedure lblForgotPassClick(Sender: TObject);
  private
    { Private declarations }
    procedure LoginUser;
    procedure ClearForm;

  public
    { Public declarations }
  end;

var
  LoginForm: TLoginForm;

implementation

{$R *.dfm}
```

```
procedure TLoginForm.btnHelpLoginClick(Sender: TObject);
begin
  LoginHelp;
end;
```

```
procedure TLoginForm.btnHelpRegisterClick(Sender: TObject);
begin
  SignUpHelp;
end;
```

```
procedure TLoginForm.btnLoginClick(Sender: TObject);
begin
  LoginUser;
end;
```

```
procedure TLoginForm.btnRegisterClick(Sender: TObject);
var
  sName, sSurname, sEmail, sPassword, sAddressLine1, sSuburbAndCode,
  sSuburb, sPostCode, sRetypedPass, sRetypedEmail : string;
  iPos : byte;
begin
  sPassword := Trim(ledPasswordSignUp.Text);
  sName := Trim(ledFirstName.Text);
  sSurname := Trim(ledSurname.Text);
  sEmail := Trim(ledEmailSignUp.Text);
  sAddressLine1 := Trim(ledAddressLine1.Text);
  sSuburbAndCode := Trim(cbxSuburbs.Items[cbxSuburbs.ItemIndex]);

  sRetypedPass := Trim(ledPasswordConfirm.Text);
  sRetypedEmail := Trim(ledEmailConfirm.Text);

  iPos := POS(',', sSuburbAndCode);
  sSuburb := Copy(sSuburbAndCode, 1, iPos - 1);
  Delete(sSuburbAndCode, 1, iPos + 1);
  sPostCode := sSuburbAndCode;
```

```
  if RequiredFieldsFilled(sName, sSurname, sEmail, sRetypedEmail,
  sPassword, sRetypedPass, sAddressLine1, sSuburb, sPostCode) =
  true then
    if PasswordCriteriaMet(sPassword) = true then
      if FieldsConfirmed(sPassword, sRetypedPass, sEmail, sRetypedEmail) =
      true then
        if EmailVerified(sEmail) = true then
          RegisterNewUser(sName, sSurname, sEmail, sPassword, sAddressLine1,
          sSuburb, sPostCode, Self);
end;
```

```
procedure TLoginForm.btnShowHideClick(Sender: TObject);
begin
  if ledPassword.PasswordChar = '*' then
  begin
    ledPassword.PasswordChar := #0;
    btnShowHide.Caption := 'Hide';
  end
  else
  begin
    ledPassword.PasswordChar := '*';
    btnShowHide.Caption := 'Show';
  end;
end;
```

```
procedure TLoginForm.btnSignUpClick(Sender: TObject);
begin
  pnlRegister.Visible := true;
  pnlLogin.Visible := false;
  ClearForm;
  ledFirstName.SetFocus;
end;
```

```
//-------------------------------------CLEARS THE FORM AS IF IT WAS NEW
procedure TLoginForm.ClearForm;
begin
  ledEmail.Clear;
  ledPassword.Clear;
  cbxStaff.Checked := false;
  ledFirstName.Clear;
  ledSurname.Clear;
  ledEmailSignUp.Clear;
  ledEmailConfirm.Clear;
  ledPasswordSignUp.Clear;
  ledPasswordConfirm.Clear;
  ledAddressLine1.Clear;
  cbxSuburbs.ItemIndex := 0;
end;
```

```
procedure TLoginForm.FormCreate(Sender: TObject);
begin
  CreateObjs;
  cbxSuburbs.Items.Text := PopulateSuburbDropDown;
  cbxSuburbs.ItemIndex := 0;
  CompleteFlights;
  DeployDrones;
end;
```

```
procedure TLoginForm.FormShow(Sender: TObject);
begin
  pnlLogin.Visible := true;
  pnlRegister.Visible := false;
  ClearForm;
  ledEmail.SetFocus;
end;
```

```
procedure TLoginForm.lblForgotPassClick(Sender: TObject);
var
  sEmail, sStaff : string;
  bStaff : boolean;
begin
  if MessageDlg('Have you forgot your password?', mtInformation, mbYesNo,
  0) = mrYes then
  begin
    sEmail := InputBox('Birds Eye Security Surveillance', 'Please ' +
    'enter the email your account is linked to.', '');

    repeat
      sStaff := InputBox('Birds Eye Security Surveillance', 'Are you part ' +
      'of our staff? (Yes/No)', 'No');
    until (UPPERCASE(sStaff) = 'YES') OR (UPPERCASE(sStaff) = 'NO');
    bStaff := UPPERCASE(sStaff) = 'YES';

    ForgotPass(sEmail, bStaff);
  end;
end;
```

```
procedure TLoginForm.lblRecommenedSuburbClick(Sender: TObject);
var
  sSuburb, sPostCode, sEmail : string;
begin
  if MessageDlg('Is the suburb you wish to register for not listed?',
  mtConfirmation, mbYesNo, 0) = mrYes then
  begin
    sSuburb := InputBox('Birds Eye Security Surveillance',
    'Which suburb do you wish to signup for?', '');
    sPostCode := InputBox('Birds Eye Security Surveillance', 'What is ' +
    'the postal code of the suburb?', '');

    while ValidatePostal(sPostCode) = false do
        sPostCode := InputBox('Birds Eye Security Surveillance',
        'You entered an invalid postal code, please enter your suburbs' +
        ' postal code.', '');
```

```
    if AlreadyInSuburbs(sPostCode) = false then
    begin
      sEmail := InputBox('Birds Eye Security Surveillance', 'Please enter ' +
        'your email', '');

      while EmailVerified(sEmail) = false do
        sEmail := InputBox('Birds Eye Security Surveillance', 'You ' +
        'entered an invalid email, please enter a valid email.', '');

      MessageDlg('Thank you for your interest in our service in your area.' +
      ' We shall contact you as soon as we are operational in your suburb.',
      mtInformation, mbOKCancel, 0);

      RecommendSuburb(sSuburb, sPostCode, sEmail);
    end
    else
      MessageDlg('The area you wish to register for is already on the list' +
      ' of suburbs we operate in.', mtInformation, mbOKCancel, 0);
  end;
end;
```

```
procedure TLoginForm.ledPasswordKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if Key = 13 then
    LoginUser;
end;
```

```
procedure TLoginForm.LoginUser;
var
  sEmail, sPassword : string;
  bStaff : boolean;
begin
  sEmail := Trim(ledEmail.Text);
  sPassword := Trim(ledPassword.Text);
  bStaff := cbxStaff.Checked;
  Login(sEmail, sPassword, bStaff, Self);
end;
```

```
procedure TLoginForm.tmrAutoTimer(Sender: TObject);
begin
  DeployDrones;
  CompleteFlights;
end;
```

```
procedure TLoginForm.btnAlreadyRegisteredClick(Sender: TObject);
begin
  pnlLogin.Visible := true;
  pnlRegister.Visible := false;
  ClearForm;
  ledEmail.SetFocus;
end;
```

```
end.
```

## Staff Form

```
unit frmStaff;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, uStaff, StdCtrls, DB, ADODB, Grids, DBGrids, uDashboard,
  ExtCtrls, uAnalyseSurveillance, Buttons, uDroneManagement, uSuburbManagement,
  uStaffReportsAndLogs, uStaffManagement, uStaffPageAccess, uAccountDetails,
  uAutoDroneDeployment, uHelp;

type
  TStaffForm = class(TForm)
    pgcStaff: TPageControl;
    tbsDashboard: TTabSheet;
    tbsAnalyseSurveillance: TTabSheet;
    tbsDroneManagement: TTabSheet;
    tbsSuburbManagement: TTabSheet;
    tbsStaffManagement: TTabSheet;
    tbsFlightLogs: TTabSheet;
    tbsReports: TTabSheet;
    tbsAccountDetails: TTabSheet;
    grpDroneStatus: TGroupBox;
    lblDemand: TLabel;
    lblNumberDrones: TLabel;
    pgbDrones: TProgressBar;
    dbgActiveDrones: TDBGrid;
    dbgSuburbStatus: TDBGrid;
    lblActiveDrones: TLabel;
    lblSuburbStatus: TLabel;
    adoActiveDrones: TADOQuery;
    dtsActiveDrones: TDataSource;
    adoSuburbStatus: TADOQuery;
    dtsSuburbStatus: TDataSource;
    dbgDroneSurveillance: TDBGrid;
    dtsAnalyseSurveillance: TDataSource;
    imgSurveillance: TImage;
    adoAnalyseSurveillance: TADOQuery;
    grpSurveillanceReport: TGroupBox;
    btnSubmit: TButton;
    cbxSummary: TComboBox;
    redSummary: TRichEdit;
    lblDescription: TLabel;
    bmbRefreshAnalyseSurv: TBitBtn;
    bmbRefreshDashboard: TBitBtn;
    dbgDrones: TDBGrid;
    dsrDrones: TDataSource;
    adoDrones: TADOQuery;
    bmbRefreshDrones: TBitBtn;
    grpNewDrone: TGroupBox;
    grpChangeDroneStatus: TGroupBox;
    rdbGenerateSerialNum: TRadioButton;
```

```
rdbUniqueSerailNum: TRadioButton;
edtNewSerialNumber: TEdit;
lblSuburbs: TLabel;
cbxSuburbsDrones: TComboBox;
cbxDroneType: TComboBox;
lblDroneType: TLabel;
bmbDroneRetry: TBitBtn;
btnAddDrone: TButton;
rgpSortDrones: TRadioGroup;
ledEditSerialNumber: TLabeledEdit;
cbxEditSuburb: TComboBox;
lblSuburb: TLabel;
cbxEditDroneType: TComboBox;
lblEditDroneType: TLabel;
btnSaveDetails: TButton;
cbxDroneStatus: TComboBox;
lblDroneStatus: TLabel;
grpAddSuburb: TGroupBox;
rdbRecommendedSuburb: TRadioButton;
rdbEnterSuburbDetails: TRadioButton;
bmbSuburbRetry: TBitBtn;
btnAddSuburb: TButton;
cbxRecommendedSuburbs: TComboBox;
ledNewSuburbName: TLabeledEdit;
ledNewSubPostCode: TLabeledEdit;
dbgSuburbs: TDBGrid;
rgpSortSuburbs: TRadioGroup;
grpEditOrDelete: TGroupBox;
ledSuburbName: TLabeledEdit;
ledPostalCode: TLabeledEdit;
btnSaveEditSuburb: TButton;
dsrSuburbs: TDataSource;
adoSuburbs: TADOQuery;
bmbRefreshSuburbs: TBitBtn;
dbgFlightLogs: TDBGrid;
rgpSortFlightLogs: TRadioGroup;
dsrFlightLogs: TDataSource;
adoFlightLogs: TADOQuery;
grpFlightLogDates: TGroupBox;
dtpEarliestDate: TDateTimePicker;
lblLatestDate: TLabel;
bmbRefreshFlighLogs: TBitBtn;
lblDate: TLabel;
dtpLatestDate: TDateTimePicker;
rgpFlighStatus: TRadioGroup;
dbgReports: TDBGrid;
rgpReportOptions: TRadioGroup;
btnGetReport: TButton;
dtsReports: TDataSource;
adoReports: TADOQuery;
grpSearchFlightLogs: TGroupBox;
edtFlighNumberSearch: TEdit;
grpAddStaffMember: TGroupBox;
ledStaffName: TLabeledEdit;
ledStaffSurname: TLabeledEdit;
ledStaffEmail: TLabeledEdit;
```

```
cbxJobTitles: TComboBox;
lblJobTitle: TLabel;
btnAddStaffMember: TButton;
bmbRetryAddStaff: TBitBtn;
dbgStaff: TDBGrid;
btnDeactivateReactivateStaff: TButton;
dtsStaff: TDataSource;
adoStaff: TADOQuery;
bmbRefreshStaff: TBitBtn;
grpDetails: TGroupBox;
lblNameTitle: TLabel;
lblSurnameTitle: TLabel;
lblEmailTitle: TLabel;
lblName: TLabel;
lblSurname: TLabel;
lblEmail: TLabel;
grpChangePassword: TGroupBox;
ledCurrentPass: TLabeledEdit;
ledNewPass: TLabeledEdit;
ledRetypedNewPass: TLabeledEdit;
btnChangePass: TButton;
grpDeactivateAccount: TGroupBox;
bmbDeactivateAcc: TBitBtn;
lblJob: TLabel;
lblJobTitleTitle: TLabel;
grpLogout: TGroupBox;
bmbLogOut: TBitBtn;
tmrAuto: TTimer;
btnHelp: TButton;
procedure FormCreate(Sender: TObject);
procedure cbxSummaryChange(Sender: TObject);
procedure btnSubmitClick(Sender: TObject);
procedure bmbRefreshDashboardClick(Sender: TObject);
procedure bmbRefreshAnalyseSurvClick(Sender: TObject);
procedure rdbGenerateSerialNumClick(Sender: TObject);
procedure rdbUniqueSerailNumClick(Sender: TObject);
procedure btnAddDroneClick(Sender: TObject);
procedure bmbRefreshDronesClick(Sender: TObject);
procedure rgpSortDronesClick(Sender: TObject);
procedure dbgDronesCellClick(Column: TColumn);
procedure btnSaveDetailsClick(Sender: TObject);
procedure rdbRecommendedSuburbClick(Sender: TObject);
procedure rdbEnterSuburbDetailsClick(Sender: TObject);
procedure bmbDroneRetryClick(Sender: TObject);
procedure bmbSuburbRetryClick(Sender: TObject);
procedure btnAddSuburbClick(Sender: TObject);
procedure rgpSortSuburbsClick(Sender: TObject);
procedure btnSaveEditSuburbClick(Sender: TObject);
procedure bmbRefreshSuburbsClick(Sender: TObject);
procedure dbgSuburbsCellClick(Column: TColumn);
procedure dtpEarliestDateChange(Sender: TObject);
procedure dtpLatestDateChange(Sender: TObject);
procedure rgpSortFlightLogsClick(Sender: TObject);
procedure bmbRefreshFlighLogsClick(Sender: TObject);
procedure rgpFlighStatusClick(Sender: TObject);
procedure btnGetReportClick(Sender: TObject);
```

```
    procedure edtFlighNumberSearchChange(Sender: TObject);
    procedure bmbRetryAddStaffClick(Sender: TObject);
    procedure btnAddStaffMemberClick(Sender: TObject);
    procedure dbgStaffCellClick(Column: TColumn);
    procedure btnDeactivateReactivateStaffClick(Sender: TObject);
    procedure bmbRefreshStaffClick(Sender: TObject);
    procedure bmbDeactivateAccClick(Sender: TObject);
    procedure btnChangePassClick(Sender: TObject);
    procedure bmbLogOutClick(Sender: TObject);
    procedure dbgDroneSurveillanceCellClick(Column: TColumn);
    procedure tmrAutoTimer(Sender: TObject);
    procedure btnHelpClick(Sender: TObject);
  private
    { Private declarations }
    procedure FormatFlightLogs;

    procedure PopulateDashboard;
    procedure CollectInfoForDroneAnalyses;
    procedure ShowDroneFootage;
    procedure PopulateDroneManagement;
    procedure PopulateDroneGrid;
    procedure PopulateEditDrones;
    procedure PopulateRecommendedSuburbs;
    procedure RecommendedSuburbErrorHandling;
    procedure PopulateEditSuburbs;
    procedure PopulateSuburbGrid;
    procedure PopulateFlightLogsDatePickers;
    procedure PopulateFlightLogsGrid;
    procedure PopulateStaffManagement;
    procedure ChangeStaffButton;
    procedure MakePagesVisible;
    procedure SetupAccountDetails;

    procedure RefreshAnalyseSurv;
    procedure RefreshDrones;
    procedure RefreshSuburbGrid;
    procedure RefreshStaff;
    procedure RefreshFlightLogs;
  public
    procedure FormSetup(StaffObj: TStaff; AccountDetialsObj : TAccountDetails;
    HelpObj : THelp);
  end;

var
  StaffForm: TStaffForm;
  objStaff : TStaff;
  objAccounDetails : TAccountDetails;
  objHelp : THelp;
  bGeneratedSerialNumber : boolean;

implementation

{$R *.dfm}

{ TStaffForm }
```

```
procedure TStaffForm.bmbRefreshFlighLogsClick(Sender: TObject);
begin
  RefreshFlightLogs;
end;
```

```
procedure TStaffForm.bmbDeactivateAccClick(Sender: TObject);
begin
  objAccounDetails.DeactivateAcc(objStaff.GetStaffID, Self);
end;
```

```
procedure TStaffForm.bmbDroneRetryClick(Sender: TObject);
begin
  rdbGenerateSerialNum.Checked := true;
  rdbUniqueSerailNum.Checked := false;
  edtNewSerialNumber.Clear;
  cbxSuburbsDrones.ItemIndex := 0;
  cbxDroneType.ItemIndex := 0;
end;
```

```
procedure TStaffForm.bmbLogOutClick(Sender: TObject);
begin
  objAccounDetails.LogOut(Self);
end;
```

```
procedure TStaffForm.bmbRefreshAnalyseSurvClick(Sender: TObject);
begin
  RefreshAnalyseSurv;
  ShowDroneFootage;
end;
```

```
procedure TStaffForm.bmbRefreshDashboardClick(Sender: TObject);
begin
  PopulateDashboard;
end;
```

```
procedure TStaffForm.bmbRefreshDronesClick(Sender: TObject);
begin
  RefreshDrones;
end;
```

```
procedure TStaffForm.bmbRefreshStaffClick(Sender: TObject);
begin
  RefreshStaff;
end;
```

```
procedure TStaffForm.bmbRefreshSuburbsClick(Sender: TObject);
begin
  RefreshSuburbGrid;
end;
```

```
procedure TStaffForm.bmbRetryAddStaffClick(Sender: TObject);
begin
  ledStaffName.Clear;
  ledStaffSurname.Clear;
  ledStaffEmail.Clear;
  cbxJobTitles.ItemIndex := 0;
end;
```

```
procedure TStaffForm.bmbSuburbRetryClick(Sender: TObject);
begin
  rdbRecommendedSuburb.Checked := true;
  rdbEnterSuburbDetails.Checked := false;
  cbxRecommendedSuburbs.ItemIndex := 0;
  ledNewSuburbName.Clear;
  ledNewSubPostCode.Clear;
end;
```

```
procedure TStaffForm.btnAddDroneClick(Sender: TObject);
var
  sSerialNumber, sSuburb, sDroneType : string;
begin
  if rdbGenerateSerialNum.Checked = true then
    sSerialNumber := GenerateSerialNumber
  else
    sSerialNumber := edtNewSerialNumber.Text;

  sSuburb := cbxSuburbsDrones.Items[cbxSuburbsDrones.ItemIndex];
  sDroneType := cbxDroneType.Items[cbxDroneType.ItemIndex];
  AddNewDrone(sSerialNumber, sSuburb, sDroneType);
end;
```

```
procedure TStaffForm.btnAddStaffMemberClick(Sender: TObject);
var
  sName, sSurname, sEmail, sJobTitle : string;
begin
  sName := ledStaffName.Text;
  sSurname := ledStaffSurname.Text;
  sEmail := ledStaffEmail.Text;
  sJobTitle := cbxJobTitles.Items[cbxJobTitles.ItemIndex];
  AddStaffMember(sName, sSurname, sEmail, sJobTitle);
end;
```

```
procedure TStaffForm.btnAddSuburbClick(Sender: TObject);
var
  sSuburb, sPostalCode : string;
  iIndex : byte;
begin
  iIndex := cbxRecommendedSuburbs.ItemIndex;
  if rdbRecommendedSuburb.Checked = true then
  begin
    SplitSuburbAndPostCode(cbxRecommendedSuburbs.Items[iIndex], sSuburb,
                           sPostalCode);
  end
  else
  begin
    sSuburb := ledNewSuburbName.Text;
    sPostalCode := ledNewSubPostCode.Text;
  end;

  AddSuburb(sSuburb, sPostalCode);
  cbxRecommendedSuburbs.Items.Delete(iIndex);
  cbxRecommendedSuburbs.ItemIndex := 0;
  RefreshSuburbGrid;
end;
```

```
procedure TStaffForm.btnChangePassClick(Sender: TObject);
var
  sOldPass, sNewPass, sRetypedPass : string;
  bCompleted : boolean;
begin
  sOldPass := Trim(ledCurrentPass.Text);
  sNewPass := Trim(ledNewPass.Text);
  sRetypedPass := Trim(ledRetypedNewPass.Text);
  objAccounDetails.ChangePassword(sOldPass, sNewPass, sRetypedPass,
  objStaff.GetPass, objStaff.GetStaffID, objStaff.GetName + ' ' +
  objStaff.GetSurname, objStaff.GetEmail, bCompleted);

  if bCompleted = true then
  begin
    objStaff.SetPass(sNewPass);
    SetupAccountDetails;
  end;
end;
```

```
procedure TStaffForm.btnDeactivateReactivateStaffClick(Sender: TObject);
var
  sEmail : string;
begin
  sEmail := adoStaff['Email'];
  if adoStaff['Inactive Staff'] = 'True' then
    ReactivateStaffMember(sEmail)
  else
    DeactiveStaffMember(sEmail);
end;
```

```
procedure TStaffForm.btnSaveDetailsClick(Sender: TObject);
var
  sOldSerialNumber, sNewSerialNumber, sStatus, sSuburb, sDroneType : string;
begin
  sOldSerialNumber := adoDrones['Serial Number'];
  sNewSerialNumber := ledEditSerialNumber.Text;
  sStatus := cbxDroneStatus.Items[cbxDroneStatus.ItemIndex];
  sSuburb := cbxEditSuburb.Items[cbxEditSuburb.ItemIndex];
  sDroneType := cbxDroneType.Items[cbxDroneType.ItemIndex];
  EditDrone(sOldSerialNumber, sNewSerialNumber, sStatus, sSuburb, sDroneType);
end;
```

```
procedure TStaffForm.btnSaveEditSuburbClick(Sender: TObject);
var
  sSuburbName, sPostalCode, sNewSuburbName, sNewPostalCode : string;
begin
  sSuburbName := adoSuburbs['Suburb Name'];
  sPostalCode := adoSuburbs['Postal Code'];
  sNewSuburbName := ledNewSuburbName.Text;
  sNewPostalCode := ledPostalCode.Text;
  SaveSuburbDetails(sSuburbName, sPostalCode, sNewSuburbName, sNewPostalCode);
  RefreshSuburbGrid;
end;
```

```
procedure TStaffForm.btnSubmitClick(Sender: TObject);
var
  sSummary, sFlightID : string;
begin
  if redSummary.Enabled = false then
    sSummary := cbxSummary.Items[cbxSummary.ItemIndex]
  else
    sSummary := redSummary.Text;

  sFlightID := adoAnalyseSurveillance['Flight No'];
  SubmitSurveillanceAnalyses(sFlightID, sSummary);
  RefreshAnalyseSurv;
end;
```

```
procedure TStaffForm.btnGetReportClick(Sender: TObject);
begin
  adoReports.Active := false;
  adoReports.SQL.Text := Report(rgpReportOptions.ItemIndex);
  adoReports.Active := true;
end;
```

```
procedure TStaffForm.btnHelpClick(Sender: TObject);
begin
  objHelp.LoadHelp(pgcStaff.ActivePage.Caption);
end;
```

```
procedure TStaffForm.cbxSummaryChange(Sender: TObject);
begin
  if cbxSummary.ItemIndex <> 4 then
    redSummary.Enabled := false
  else
    redSummary.Enabled := true;
end;
```

```
procedure TStaffForm.ChangeStaffButton;
begin
  if adoStaff.RecordCount <> 0 then
    if adoStaff['Inactive Staff'] = 'True' then
      btnDeactivateReactivateStaff.Caption := 'Reactivate Staff Member'
    else
      btnDeactivateReactivateStaff.Caption := 'Deactivate Staff Member';
end;
```

```
procedure TStaffForm.CollectInfoForDroneAnalyses;
begin
  adoAnalyseSurveillance.Active := false;
  adoAnalyseSurveillance.SQL.Text := SurveillanceToBeAnalysed(
                                                    objStaff.GetStaffID);
  adoAnalyseSurveillance.Active := true;
  if adoAnalyseSurveillance.RecordCount <> 0 then
  begin
    btnSubmit.Enabled := true;
    ShowDroneFootage;
  end
  else
    btnSubmit.Enabled := false;
end;
```

```
procedure TStaffForm.dbgDronesCellClick(Column: TColumn);
begin
  PopulateEditDrones;
end;
```

```
procedure TStaffForm.dbgDroneSurveillanceCellClick(Column: TColumn);
begin
  ShowDroneFootage;
end;
```

```
procedure TStaffForm.dbgStaffCellClick(Column: TColumn);
begin
  ChangeStaffButton;
end;
```

```
procedure TStaffForm.dbgSuburbsCellClick(Column: TColumn);
begin
  PopulateEditSuburbs;
end;
```

```
procedure TStaffForm.dtpEarliestDateChange(Sender: TObject);
begin
  PopulateFlightLogsGrid;
end;
```

```
procedure TStaffForm.dtpLatestDateChange(Sender: TObject);
begin
  PopulateFlightLogsGrid;
end;
```

```
procedure TStaffForm.edtFlighNumberSearchChange(Sender: TObject);
begin
  PopulateFlightLogsGrid;
end;
```

```
//-------------------------------------FORMATS THE FLIGHT LOGS GRID
procedure TStaffForm.FormatFlightLogs;
begin
  dbgFlightLogs.Columns.Items[4].Alignment := taRightJustify;
  dbgFlightLogs.Columns.Items[5].Width := 200;
end;
```

```
procedure TStaffForm.FormCreate(Sender: TObject);
begin
  //-------------------------------------MAKES THIS FORM THE MAIN FORM
  Pointer((@Application.MainForm)^) := Self;

  adoActiveDrones.Active := true;
  adoSuburbStatus.Active := true;
  adoSuburbs.Active := true;
  adoStaff.Active := true;
  adoFlightLogs.Active := true;

  PopulateDashboard;
  PopulateDroneGrid;
  PopulateDroneManagement;
  PopulateEditDrones;
  PopulateRecommendedSuburbs;
  RecommendedSuburbErrorHandling;
  PopulateEditSuburbs;
  PopulateSuburbGrid;
  PopulateFlightLogsDatePickers;
  PopulateFlightLogsGrid;
  PopulateStaffManagement;
end;
```

```
//-------------------------------------ASSIGNS OBJECTS AND WELCOMES USER
procedure TStaffForm.FormSetup(StaffObj: TStaff; AccountDetialsObj :
TAccountDetails; HelpObj : THelp);
begin
  objStaff := StaffObj;
  objAccounDetails := AccountDetialsObj;
  objHelp := HelpObj;
  CollectInfoForDroneAnalyses;
  MakePagesVisible;
  SetupAccountDetails;

  MessageDlg('Welcome ' + objStaff.GetName + ' ' + objStaff.GetSurname +
  ' you have been succesfully logged in!', mtInformation, mbOKCancel, 0);
end;
```

```
procedure TStaffForm.MakePagesVisible;
begin
  SetUpDictionary;
  AccessPages(pgcStaff, objStaff.GetJobTitle);
end;
```

```
procedure TStaffForm.PopulateDashboard;
var
  iTotDrones, iActiveDrones : word;
  sDemand : string;
begin
  DroneStatus(iTotDrones, iActiveDrones, sDemand);
  pgbDrones.Max := iTotDrones;
  pgbDrones.Position := iActiveDrones;
  lblNumberDrones.Caption := IntToStr(iActiveDrones) + '/' +
                             IntToStr(iTotDrones);
  lblDemand.Caption := sDemand;
  adoActiveDrones.Active := false;
  adoSuburbStatus.Active := false;
  adoActiveDrones.Active := true;
  adoSuburbStatus.Active := true;
end;
```

```
procedure TStaffForm.PopulateDroneGrid;
begin
  adoDrones.Active := false;
  adoDrones.SQL.Text := DroneOrderBy(rgpSortDrones.Items[
                                                 rgpSortDrones.ItemIndex]);
  adoDrones.Active := true;
end;
```

```
procedure TStaffForm.PopulateDroneManagement;
var
  sSuburbs, sDroneTypes : string;
begin
  sSuburbs := ToStringSuburbs;
  sDroneTypes := ToStringDroneTypes;

  cbxSuburbsDrones.Items.Text := sSuburbs;
  cbxDroneType.Items.Text := sDroneTypes;
  cbxSuburbsDrones.ItemIndex := 0;
  cbxDroneType.ItemIndex := 0;

  cbxEditSuburb.Items.Text := sSuburbs;
  cbxEditDroneType.Items.Text := sDroneTypes;
end;
```

```
procedure TStaffForm.PopulateEditDrones;
begin
  if adoDrones.RecordCount <> 0 then
  begin
    ledEditSerialNumber.Text := adoDrones['Serial Number'];
    btnSaveDetails.Enabled := true;
  end
  else
    btnSaveDetails.Enabled := false;

  cbxEditDroneType.ItemIndex := 0;
  cbxEditSuburb.ItemIndex := 0;
  cbxDroneStatus.ItemIndex := 0;
end;
```

```
procedure TStaffForm.PopulateEditSuburbs;
begin
  if adoSuburbs.RecordCount <> 0 then
  begin
    btnSaveEditSuburb.Enabled := true;
    ledSuburbName.Text := adoSuburbs['Suburb Name'];
    ledPostalCode.Text := adoSuburbs['Postal Code'];
  end
  else
  begin
    btnSaveEditSuburb.Enabled := false;
  end;
end;
```

```
procedure TStaffForm.PopulateFlightLogsDatePickers;
var
  dEarliestDate, dLatestDate : TDate;
begin
  dEarliestDate := EarliestDate;
  dLatestDate := LatestDate;

  dtpEarliestDate.Date := dEarliestDate;
  dtpLatestDate.Date := dLatestDate;

  dtpEarliestDate.MinDate := dEarliestDate;
  dtpEarliestDate.MaxDate := dLatestDate;

  dtpLatestDate.MinDate := dEarliestDate;
  dtpLatestDate.MaxDate := dLatestDate;
end;
```

```
procedure TStaffForm.PopulateFlightLogsGrid;
var
  sStartDate, sEndDate, sFlightNumber : string;
  iOrderByIndex, iStatusIndex : byte;
begin
  iStatusIndex := rgpFlighStatus.ItemIndex;
  iOrderByIndex := rgpSortFlightLogs.ItemIndex;
  sStartDate := DateToStr(dtpEarliestDate.Date);
  sEndDate := DateToStr(dtpLatestDate.Date);
  sFlightNumber := Trim(edtFlighNumberSearch.Text);

  adoFlightLogs.Active := false;
  adoFlightLogs.SQL.Text := FlightLogsOrderBy(iOrderByIndex, iStatusIndex,
  sStartDate, sEndDate, sFlightNumber);
  adoFlightLogs.Active := true;

  FormatFlightLogs;
end;
```

```
procedure TStaffForm.PopulateStaffManagement;
begin
  cbxJobTitles.Items.Text := JobTitles;
  cbxJobTitles.ItemIndex := 0;
  if adoStaff.RecordCount <> 0 then
    btnDeactivateReactivateStaff.Enabled := true;
end;
```

```
procedure TStaffForm.PopulateSuburbGrid;
begin
  adoSuburbs.Active := false;
  if rgpSortSuburbs.ItemIndex = 0 then
    adoSuburbs.SQL.Text := SuburbsOrderBy(true)
  else
    adoSuburbs.SQL.Text := SuburbsOrderBy(false);
  adoSuburbs.Active := true;
end;
```

```
procedure TStaffForm.PopulateRecommendedSuburbs;
begin
  cbxRecommendedSuburbs.Items.Text := RecommendedSuburbs;
  cbxRecommendedSuburbs.ItemIndex := 0;
end;
```

```
procedure TStaffForm.rdbRecommendedSuburbClick(Sender: TObject);
begin
  if (rdbRecommendedSuburb.Checked = true) AND (
  cbxRecommendedSuburbs.Items.Count <> 0) then
  begin
    rdbEnterSuburbDetails.Checked := false;
    ledNewSuburbName.Enabled := false;
    ledNewSubPostCode.Enabled := false;
    cbxRecommendedSuburbs.Enabled := true;
    cbxRecommendedSuburbs.SetFocus;
  end;

  RecommendedSuburbErrorHandling;
end;
```

```
procedure TStaffForm.rdbEnterSuburbDetailsClick(Sender: TObject);
begin
  if rdbEnterSuburbDetails.Checked = true then
  begin
    rdbRecommendedSuburb.Checked := false;
    ledNewSuburbName.Enabled := true;
    ledNewSubPostCode.Enabled := true;
    cbxRecommendedSuburbs.Enabled := false;
  end;
end;
```

```
procedure TStaffForm.rdbGenerateSerialNumClick(Sender: TObject);
begin
  if rdbGenerateSerialNum.Checked = true then
  begin
    rdbUniqueSerailNum.Checked := false;
    edtNewSerialNumber.Enabled := false;
    bGeneratedSerialNumber := true;
  end;
end;
```

```
procedure TStaffForm.rdbUniqueSerailNumClick(Sender: TObject);
begin
  if rdbUniqueSerailNum.Checked = true then
  begin
    rdbGenerateSerialNum.Checked := false;
    edtNewSerialNumber.Enabled := true;
    edtNewSerialNumber.SetFocus;
    bGeneratedSerialNumber := false;
  end;
end;
```

```
procedure TStaffForm.RecommendedSuburbErrorHandling;
begin
  if (rdbRecommendedSuburb.Checked = true) AND (
  cbxRecommendedSuburbs.Items.Count = 0) then
  begin
    rdbEnterSuburbDetails.Checked := true;
    rdbRecommendedSuburb.Checked := false;
  end;
end;
```

```
procedure TStaffForm.RefreshAnalyseSurv;
begin
  adoAnalyseSurveillance.Active := false;
  adoAnalyseSurveillance.Active := true;
end;
```

```
procedure TStaffForm.RefreshDrones;
begin
  adoDrones.Active := false;
  adoDrones.Active := true;
end;
```

```
procedure TStaffForm.RefreshFlightLogs;
begin
  adoFlightLogs.Active := false;
  adoFlightLogs.Active := true;
end;
```

```
procedure TStaffForm.RefreshStaff;
begin
  adoStaff.Active := false;
  adoStaff.Active := true;
  ChangeStaffButton;
end;
```

```
procedure TStaffForm.RefreshSuburbGrid;
begin
  adoSuburbs.Active := false;
  adoSuburbs.Active := true;
end;
```

```
procedure TStaffForm.rgpFlighStatusClick(Sender: TObject);
begin
  PopulateFlightLogsGrid;
end;
```

```
procedure TStaffForm.rgpSortDronesClick(Sender: TObject);
begin
  PopulateDroneGrid;
end;
```

```
procedure TStaffForm.rgpSortFlightLogsClick(Sender: TObject);
begin
  PopulateFlightLogsGrid;
end;
```

```
procedure TStaffForm.rgpSortSuburbsClick(Sender: TObject);
begin
  PopulateSuburbGrid;
end;
```

```
procedure TStaffForm.SetupAccountDetails;
begin
  lblName.Caption := objStaff.GetName;
  lblSurname.Caption := objStaff.GetSurname;
  lblEmail.Caption := objStaff.GetEmail;
  lblJob.Caption := objStaff.GetJobTitle;
end;
```

```
procedure TStaffForm.ShowDroneFootage;
var
  sFileName, sFlightID : string;
begin
  if adoAnalyseSurveillance.RecordCount <> 0 then
  begin
    sFlightID := adoAnalyseSurveillance['Flight No'];
    sFileName := GetSurveillanceFootage(sFlightID);
    imgSurveillance.Picture.LoadFromFile(sFileName);
  end;
end;
```

```
procedure TStaffForm.tmrAutoTimer(Sender: TObject);
begin
  DeployDrones;
  CompleteFlights;
  PopulateDashboard;
  RefreshAnalyseSurv;
  RefreshDrones;
  RefreshSuburbGrid;
  RefreshStaff;
  RefreshFlightLogs;
end;
```

```
end.
```

## Client Form

```
unit frmClient;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, jpeg, ExtCtrls, DB, ADODB, Grids, DBGrids, StdCtrls,
  uClient, uServiceRequest, Buttons, uClientReportsAndLogs, uAccountDetails,
  uAutoDroneDeployment, uHelp, uLibrary, DateUtils;

type
  TClientForm = class(TForm)
    imgBackground: TImage;
    pgcClient: TPageControl;
    tbsDroneServices: TTabSheet;
    tbsResponseLogs: TTabSheet;
    tbsReports: TTabSheet;
    tbsAccountDetails: TTabSheet;
    dbgServices: TDBGrid;
    adoServices: TADOQuery;
    dtsServices: TDataSource;
    dtpStart: TDateTimePicker;
    dtpEnd: TDateTimePicker;
    lblStartDate: TLabel;
    lblEndDate: TLabel;
    grbSummary: TGroupBox;
    btnRequestService: TButton;
    lblSummary: TLabel;
    dbgDroneFlights: TDBGrid;
    adoResponeLogs: TADOQuery;
    dtsResponseLogs: TDataSource;
    grpIncidentSummary: TGroupBox;
    lblIncidentSummary: TLabel;
    bmbRefreshLogs: TBitBtn;
    rgpSort: TRadioGroup;
    btnShowFootage: TButton;
    rgpReportType: TRadioGroup;
    redReports: TRichEdit;
    btnSendToEmail: TButton;
    Button1: TButton;
    grpChangePassword: TGroupBox;
    ledCurrentPass: TLabeledEdit;
    ledNewPass: TLabeledEdit;
    ledRetypedNewPass: TLabeledEdit;
    btnChangePass: TButton;
    grpDetails: TGroupBox;
    lblNameTitle: TLabel;
    lblSurnameTitle: TLabel;
    lblEmailTitle: TLabel;
    lblName: TLabel;
    lblSurname: TLabel;
    lblEmail: TLabel;
```

```
    grpAddress: TGroupBox;
    lblAddressTitle: TLabel;
    lblSuburbTitle: TLabel;
    lblAddress: TLabel;
    lblSuburb: TLabel;
    grpDeactivateAccount: TGroupBox;
    bmbDeactivateAcc: TBitBtn;
    grpLogout: TGroupBox;
    bmbLogOut: TBitBtn;
    tmrAuto: TTimer;
    rgpStatus: TRadioGroup;
    btnHelp: TButton;
    procedure dbgServicesCellClick(Column: TColumn);
    procedure dtpStartChange(Sender: TObject);
    procedure dtpEndChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure btnRequestServiceClick(Sender: TObject);
    procedure bmbRefreshLogsClick(Sender: TObject);
    procedure dbgDroneFlightsCellClick(Column: TColumn);
    procedure rgpSortClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure btnChangePassClick(Sender: TObject);
    procedure bmbDeactivateAccClick(Sender: TObject);
    procedure bmbLogOutClick(Sender: TObject);
    procedure tmrAutoTimer(Sender: TObject);
    procedure rgpStatusClick(Sender: TObject);
    procedure btnSendToEmailClick(Sender: TObject);
    procedure btnShowFootageClick(Sender: TObject);
    procedure btnHelpClick(Sender: TObject);
  private
    procedure FormatServiceDBGrid;
    procedure FormatFlightLogsDBGrid;
    procedure FormatReportsRichEdit;

    procedure UpdateSummary;
    procedure PopulateServiceResponse;
    procedure SurveillanceReport;
    procedure AccountDetailsSetup;

    procedure RefreshFlightLogs;
  public
    procedure FormSetup(ClientObj: TClient; AccountDetialsObj :
    TAccountDetails; HelpObj : THelp);
  end;

var
  ClientForm: TClientForm;
  objClient : TClient;
  objServiceRequest : TServiceRequest;
  objAccountDetails : TAccountDetails;
  objHelp : THelp;

implementation

{$R *.dfm}
```

```
{ TClientForm }
```

```
procedure TClientForm.AccountDetailsSetup;
begin
  lblName.Caption := objClient.GetName;
  lblSurname.Caption := objClient.GetSurname;
  lblEmail.Caption := objClient.GetEmail;
  lblAddress.Caption := objClient.GetAddressLine1;
  lblSuburb.Caption := objClient.GetSuburbName;
end;
```

```
procedure TClientForm.bmbDeactivateAccClick(Sender: TObject);
begin
  objAccountDetails.DeactivateAcc(objClient.GetUserID, Self);
end;
```

```
procedure TClientForm.bmbLogOutClick(Sender: TObject);
begin
  objAccountDetails.LogOut(Self);
end;
```

```
procedure TClientForm.bmbRefreshLogsClick(Sender: TObject);
begin
  RefreshFlightLogs;
end;
```

```
procedure TClientForm.btnChangePassClick(Sender: TObject);
var
  sOldPass, sNewPass, sRetypedPass : string;
  bCompleted : boolean;
begin
  sOldPass := Trim(ledCurrentPass.Text);
  sNewPass := Trim(ledNewPass.Text);
  sRetypedPass := Trim(ledRetypedNewPass.Text);
  objAccountDetails.ChangePassword(sOldPass, sNewPass, sRetypedPass,
  objClient.GetPass, objClient.GetUserID, objClient.GetName + ' ' +
  objClient.GetSurname, objClient.GetEmail, bCompleted);

  if bCompleted = true then
  begin
    objClient.SetPass(sNewPass);
    AccountDetailsSetup;
  end;
end;
```

```
procedure TClientForm.btnHelpClick(Sender: TObject);
begin
  objHelp.LoadHelp(pgcClient.ActivePage.Caption);
end;
```

```
procedure TClientForm.btnRequestServiceClick(Sender: TObject);
var
  sStartTime : string;
begin
  if MessageDlg(objServiceRequest.ConfirmServiceRequest, mtConfirmation,
  mbYesNo, 0) = mrYes then
  begin
    sStartTime := adoServices['Surveillance Start Time'];
    objServiceRequest.RequestService(objClient.GetUserID, sStartTime);
  end;
end;
```

```
procedure TClientForm.btnSendToEmailClick(Sender: TObject);
var
  sReportName, sFullName : string;
begin
  MessageDlg('Sending Email Please Wait...', mtInformation, mbOKCancel, 0);
  sFullName := objClient.GetName + ' ' + objClient.GetSurname;
  sReportName := Trim(rgpReportType.Items[rgpReportType.ItemIndex]);
  objEmails.SendEmail(sFullName, objClient.GetEmail, sReportName +
  ' for ' + LongMonthNames[MonthOf(Date())], 'Dear ' + sFullName + ',' +
  #13 + 'Here is the following report you requested.' + #13 +
  redReports.Text + #13 + #13 + 'Regards,' + #13 + 'Birds Eye Security');
end;
```

```
procedure TClientForm.btnShowFootageClick(Sender: TObject);
var
  sDroneFlightID : string;
begin
  sDroneFlightID := adoResponeLogs['Flight No'];
  ShowSurvFootage(sDroneFlightID);
end;
```

```
procedure TClientForm.Button1Click(Sender: TObject);
begin
  if rgpReportType.ItemIndex = 0 then
    redReports.Text := CostOfServicesReport(objClient.GetUserID)
  else
    redReports.Text := SuburbReport(objClient.GetUserID);
  btnSendToEmail.Enabled := true;
end;
```

```
//-------------------------------------ASSIGNS OBJECTS AND WELCOMES USER
procedure TClientForm.FormSetup(ClientObj: TClient; AccountDetialsObj :
TAccountDetails; HelpObj : THelp);
begin
  objClient := ClientObj;
  objAccountDetails := AccountDetialsObj;
  objHelp := HelpObj;
  PopulateServiceResponse;
  SurveillanceReport;
  AccountDetailsSetup;

  MessageDlg('Welcome ' + objClient.GetName + ' ' + objClient.GetSurname +
  ' you have been succesfully logged in!', mtInformation, mbOKCancel, 0);
end;
```

```
procedure TClientForm.dbgDroneFlightsCellClick(Column: TColumn);
begin
  SurveillanceReport;
end;
```

```
procedure TClientForm.dbgServicesCellClick(Column: TColumn);
begin
  UpdateSummary;
end;
```

```
procedure TClientForm.dtpEndChange(Sender: TObject);
begin
  if dtpStart.Date > dtpEnd.Date then
    dtpStart.Date := dtpEnd.Date;

  UpdateSummary;
end;
```

```
procedure TClientForm.dtpStartChange(Sender: TObject);
begin
  if dtpStart.Date > dtpEnd.Date then
    dtpEnd.Date := dtpStart.Date;

  UpdateSummary;
end;
```

```
//-------------------------------------FORMATS THE FLIGHT LOGS GRID
procedure TClientForm.FormatFlightLogsDBGrid;
begin
  dbgDroneFlights.Columns.Items[0].Width := 65;

  dbgDroneFlights.Columns.Items[3].Alignment := taRightJustify;
end;
```

```
//-------------------------------------FORMATS THE REPORTS RICH EDIT
procedure TClientForm.FormatReportsRichEdit;
begin
  redReports.Paragraph.TabCount := 2;
  redReports.Paragraph.Tab[0] := 140;
  redReports.Paragraph.Tab[1] := 315;
end;
```

```
//-------------------------------------FORMATS THE REQUEST SERVICES GRID
procedure TClientForm.FormatServiceDBGrid;
begin
  dbgServices.Columns.Items[1].Width := 160;
  dbgServices.Columns.Items[2].Width := 160;
  dbgServices.Columns.Items[3].Width := 120;

  dbgServices.Columns.Items[1].Alignment := taRightJustify;
  dbgServices.Columns.Items[2].Alignment := taRightJustify;
  dbgServices.Columns.Items[3].Alignment := taRightJustify;
end;
```

```
procedure TClientForm.FormCreate(Sender: TObject);
begin
  //-------------------------------------MAKES THIS FORM THE MAIN FORM
  Pointer((@Application.MainForm)^) := Self;

  objServiceRequest := TServiceRequest.Create;

  adoServices.Active := true;

  dtpStart.MinDate := Date();
  dtpEnd.MinDate := Date();
  UpdateSummary;
  FormatServiceDBGrid;
  FormatReportsRichEdit;
end;
```

```
procedure TClientForm.PopulateServiceResponse;
begin
  adoResponeLogs.Active := false;
  adoResponeLogs.SQL.Text := UserFlightReports(objClient.GetUserID,
                              rgpSort.ItemIndex, rgpStatus.ItemIndex);
  adoResponeLogs.Active := true;
  FormatFlightLogsDBGrid;
end;
```

```
procedure TClientForm.RefreshFlightLogs;
begin
  adoResponeLogs.Active := false;
  adoResponeLogs.Active := true;
  FormatFlightLogsDBGrid;
end;
```

```
procedure TClientForm.rgpSortClick(Sender: TObject);
begin
  PopulateServiceResponse;
end;
```

```
procedure TClientForm.rgpStatusClick(Sender: TObject);
begin
  PopulateServiceResponse;
end;
```

```
procedure TClientForm.SurveillanceReport;
var
  sFlightID : string;
begin
  if adoResponeLogs.RecordCount <> 0 then
  begin
    sFlightID := adoResponeLogs['Flight No'];
    lblIncidentSummary.Caption := SurveillanceSummary(sFlightID);

    if adoResponeLogs['Status'] = 'Completed' then
      btnShowFootage.Enabled := true
    else
      btnShowFootage.Enabled := false;
  end;
end;
```

```
procedure TClientForm.tmrAutoTimer(Sender: TObject);
begin
  DeployDrones;
  CompleteFlights;
  RefreshFlightLogs;
end;
```

```
procedure TClientForm.UpdateSummary;
var
  startDate, endDate : TDate;
  sService : string;
  rPricePerDay : real;
begin
  startDate := dtpStart.Date;
  endDate := dtpEnd.Date;
  sService := adoServices['Service Name'];
  rPricePerDay := adoServices['Price Per Day'];
  objServiceRequest.SetValues(startDate, endDate, sService, rPricePerDay);
  lblSummary.Caption := objServiceRequest.UpdateServiceSummary;
end;
```

```
end.
```

## Footage Form

```
unit frmFootage;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls;

type
  TFootageForm = class(TForm)
    imgFootage: TImage;
    bmbClose: TBitBtn;
    procedure bmbCloseClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure ShowFootage(FilePath : string);
  end;

var
  FootageForm: TFootageForm;

implementation

{$R *.dfm}

procedure TFootageForm.bmbCloseClick(Sender: TObject);
begin
  FootageForm.Destroy;
end;

procedure TFootageForm.ShowFootage(FilePath: string);
begin
  imgFootage.Picture.LoadFromFile(FilePath);
end;

end.
```

# Other Units

## Suburb Management Unit

```
unit uSuburbManagement;
```

```
interface
  uses
    uLibrary, Dialogs, uDB_Source;

  //-----------------------------------Add New Suburb Methods
  function RecommendedSuburbs : string;
  procedure SplitSuburbAndPostCode(SuburbAndCode : string; var
  Suburb, PostCode : string);
  function IsRecommendedSuburb(SuburbName, PostalCode :
  string) : boolean;
  procedure AddSuburb(SuburbName, PostalCode : string);
  procedure NotifyOfNewSuburb(RecommendedSubID, SuburbName,
  PostalCode : string);

 //-----------------------------------Edit Suburb Methods
  function SuburbsOrderBy(IsAscending : boolean) : string;
  procedure SaveSuburbDetails(SuburbName, PostalCode,
  NewSuburbName, NewPostalCode : string);

implementation
```

```
//-----------------------------------RETURNS ALL RECOMMENDED SUBURBS
function RecommendedSuburbs : string;
var
  sSQL : string;
begin
  sSQL := 'Select [Suburb Name], [Postal Code] '
        + 'from RecommendedSuburbs';
  Result := objDB.ToString(sSQL, SUBURB_DELIMETER);
end;
```

```
//-----------------------------------SPLITS THE POSTAL CODE AND SUBURB NAME
procedure SplitSuburbAndPostCode(SuburbAndCode : string; var Suburb, PostCode :
string);
var
  iPos : byte;
begin
  iPos := POS(SUBURB_DELIMETER, SuburbAndCode);
  Suburb := Copy(SuburbAndCode, 1, iPos - 1);
  Delete(SuburbAndCode, 1, iPos + Length(SUBURB_DELIMETER) -
  1);
  PostCode := SuburbAndCode;
end;
```

```
//-----------------------------------CHECKS IF THE ENTERED SUBURB IS A
//-----------------------------------RECOMMENDED SUBURB
function IsRecommendedSuburb(SuburbName, PostalCode : string) : boolean;
var
  sSQL : string;
begin
  sSQL := 'Select * '
        + 'from RecommendedSuburbs '
         + 'where [Suburb Name] = "' + SuburbName + '" AND [Postal Code] = "' +
          PostalCode + '"';
  Result := objDB.RecordExist(sSQL);
end;
```

```
//-----------------------------------ADDS A NEW SUBURB
procedure AddSuburb(SuburbName, PostalCode : string);
var
  sSQL, sRecSubID : string;
begin
  sSQL := 'Select * '
        + 'from Suburbs '
        + 'where [Postal Code] = "' + PostalCode + '"';

  //-----------------------------------CHECKS IF SUBURB EXISTS
  if objDB.RecordExist(sSQL) = false then
  begin
    sSQL := 'Insert into Suburbs '
          + '([Suburb Name], [Postal Code]) '
          + 'Values("' + SuburbName + '", "' + PostalCode + '")';
    objDB.DoSQL(sSQL);

    //-----------------------------------CHECKS IF A USER ENTERED A
    //-----------------------------------RECOMMENDED SUBURB
    if IsRecommendedSuburb(SuburbName, PostalCode) = true then
    begin
      sSQL := 'Select [RecSuburbID] '
            + 'from RecommendedSuburbs '
            + 'where [Suburb Name] = "' + SuburbName + '" AND ' +
              '[Postal Code] = "' + PostalCode + '"';
      sRecSubID := objDB.ToString(sSQL, '');

      NotifyOfNewSuburb(sRecSubID, SuburbName, PostalCode);
    end;

    sSQL := 'Delete from RecommendedSuburbs '
          + 'where [Suburb Name] = "' + SuburbName + '" AND ' +
            '[Postal Code] = "' + PostalCode + '"';
    objDB.DoSQL(sSQL);

    MessageDlg('Suburb Successfully Added', mtInformation, mbOKCancel, 0);
  end
  else
    MessageDlg('The suburb you are trying to enter already exists ' +
    'within the database', mtError, mbOKCancel, 0);
end;
```

```
//-----------------------------------NOTIFIES PEOPLE VIA EMAIL
procedure NotifyOfNewSuburb(RecommendedSubID, SuburbName, PostalCode
  : string);
var
  sSQL : string;
  arrEmails : TDynamicArray;
  K : byte;
  iLength : word;
begin
  sSQL := 'Select [User Email] '
        + 'from NewSuburbEmailNotify '
        + 'where [RecSubID] = ' + RecommendedSubID;
  arrEmails := objDB.ToArray(sSQL);

  //-----------------------------------CHECKS IF THERE ARE EMAILS TO SEND
  iLength := Length(arrEmails);
  if iLength <> 0 then
  begin
    MessageDlg('Sending Emails Please Wait...', mtInformation, mbOKCancel, 0);
    for K := 0 to iLength - 1 do
    begin
      //-----------------------------------SENDS EMAILS
      objEmails.SendEmail('', arrEmails[K],
      'Your Suburb Has Been Added to Our Service Areas', 'We are ' +
      'excited to announce that ' + SuburbName + ' with postal code ' +
      PostalCode + ' has been added to our service areas!' + #13 + #13 +
      'In order to sign up, press the Sign Up button and fill out ' +
      'all your information.' + #13 + 'You will now be able to ' +
      'select your suburb from the drop down.' + #13 + #13 + 'Regards,' +
      #13 + 'Birds Eye Security');
    end;
  end;
end;
```

```
//-----------------------------------ORDERS SUBURBS TABLE
function SuburbsOrderBy(IsAscending : boolean) : string;
begin
  Result := 'Select [Suburb Name], [Postal Code] '
          + 'from Suburbs ';

  if IsAscending = true then
    Result := Result + 'Order By [Suburb Name]'
  else
    Result := Result + 'Order By [Suburb Name] DESC';
end;
```

```
//-----------------------------------SAVES A SUBURBS DETAILS
procedure SaveSuburbDetails(SuburbName, PostalCode, NewSuburbName,
NewPostalCode : string);
var
  sSQL : string;
begin
  //-----------------------------------ENSURES POSTAL CODE IS VALID
  if ValidatePostal(PostalCode) = true then
  begin
    sSQL := 'Update Suburbs '
          + 'Set [Postal Code] = "' + NewPostalCode + '" '
          + 'where [Suburb Name] = "' + SuburbName + '" AND [Postal Code] ' +
            '= "' + PostalCode + '"';
    objDB.DoSQL(sSQL);

    sSQL := 'Update Suburbs '
          + 'Set [Suburb Name] = "' + NewSuburbName + '" '
          + 'where [Suburb Name] = "' + SuburbName + '" AND [Postal Code] ' +
            '= "' + PostalCode + '"';
    objDB.DoSQL(sSQL);

    MessageDlg('Suburb Details Changed Successfully', mtInformation,
    mbOKCancel, 0);
  end;
end;
```

```
end.
```

## Staff Reports and Logs Unit

```
unit uStaffReportsAndLogs;

interface
  uses
    uLibrary, SysUtils;

  const
    //-------------------------------------ARRAY OF ORDER BY SQL
    FLIGHT_LOGS_ORDER_BY_SQL : array[0..4] of string =
    (
        'Order by [Suburb Name], [DroneFlights.DroneFlightID]',
        'Order by [Drone Type], [DroneFlights.DroneFlightID]',
        'Order by [Date and Time of Response] DESC, ' +
        '[DroneFlights.DroneFlightID] DESC',
        'Order by [Date and Time of Response], [DroneFlights.DroneFlightID]',
        'Order by [First Name], [Surname], [DroneFlights.DroneFlightID]'
    );

    //-------------------------------------ARRAY OF STATUS SQL
    FLIGHT_LOGS_STATUS_SQL : array[0..2] of string =
    (
        '',
        '[DroneFlights.Status] = "Completed" AND ',
        '[DroneFlights.Status] = "Departed" AND '
    );

    //-------------------------------------ARRAY OF DIFFERENT REPORT SQL
    REPORTS_SQL : array[0..7] of string =
    (
          'Select [Serial Number], [Suburb Name], [Service Name] AS ' +
          '[Drone Type], [Status] '
        + 'from Drones, Suburbs, Services '
        + 'where [Suburb] = [SuburbID] AND [Drone Type] = [ServiceID]',

          'Select [Serial Number], [Suburb Name], [Service Name] AS ' +
          '[Drone Type], [Status] '
        + 'from Drones, Suburbs, Services '
        + 'where [Suburb] = [SuburbID] AND [Drone Type] = [ServiceID] AND' +
          ' [Status] = "Active"',

          'Select [Serial Number], [Suburb Name], [Service Name] AS ' +
          '[Drone Type], [Status] '
        + 'from Drones, Suburbs, Services '
        + 'where [Suburb] = [SuburbID] AND [Drone Type] = [ServiceID] AND' +
          ' [Status] = "Being Serviced"',

          'SELECT [Serial Number], [Suburb Name], [Service Name] AS ' +
          '[Drone Type], (40 - Count([DroneFlightID])) AS ' +
          '[Flights Till Service] '
        + 'FROM Drones, DroneFlights, Suburbs, Services '
        + 'WHERE [Drones.DroneID] = [DroneFlights.DroneID] AND ' +
          '[Suburb] = [SuburbID] AND [Drone Type] = [ServiceID] '
```

```
        + 'GROUP BY [Serial Number], [Suburb Name], [Service Name]',
          'Select [Suburb Name], [Postal Code], Count([UserID]) AS ' +
          '[Number of Users]'
        + 'from Suburbs, Users '
        + 'where [Suburbs.SuburbID] = [Users.Suburb] '
        + 'Group By [Suburb Name], [Postal Code]',

          'Select [Suburb Name], [Postal Code], [Serial Number], ' +
          'Format([Date and Time of Response], "Long Time"), ' +
          '[DroneFlights.Status] '
        + 'from DroneFlights, Drones, Suburbs '
        + 'where [Drones.DroneID] = [DroneFlights.DroneID] AND ' +
          '[Drones.Suburb] = [Suburbs.SuburbID]',

          'Select [Suburb Name], [Postal Code], [Number of Recommendations] '
        + 'from RecommendedSuburbs',

          'Select [Suburb Name], [Postal Code], [Number of Recommendations] '
        + 'from RecommendedSuburbs '
        + 'where [Number of Recommendations] > 100'
    );

  function EarliestDate : TDate;
  function LatestDate : TDate;
  function FlightLogsOrderBy(OrderIndex, StatusIndex : byte; StartDate,
  EndDate, FlightNumber : string) : string;
  function Report(ReportID : byte) : string;

implementation
```

```
//-------------------------------------RETURNS THE DATE OF THE EARLIEST FLIGHT
function EarliestDate : TDate;
var
  sSQL, sDate : string;
begin
  sSQL := 'Select DISTINCT Top 1 Format([Date and Time of Response]' +
          ', "Short Date") '
        + 'from DroneFlights '
        + 'order by Format([Date and Time of Response]' +
          ', "Short Date")';
  sDate := Copy(objDB.ToString(sSQL, ''), 1, 10);
  if sDate <> '' then
    Result := StrToDate(sDate)
  else
    Result := Date();
end;
```

```
//-------------------------------------RETURNS THE DATE OF THE LATEST FLIGHT
function LatestDate : TDate;
var
  sSQL, sDate : string;
begin
  sSQL := 'Select DISTINCT Top 1 Format([Date and Time of Response]' +
          ', "Short Date") '
        + 'from DroneFlights '
        + 'order by Format([Date and Time of Response]' +
          ', "Short Date") DESC';
  sDate := Copy(objDB.ToString(sSQL, ''), 1, 10);
  if sDate <> '' then
    Result := StrToDate(sDate)
  else
    Result := Date();
end;
```

```
//-------------------------------------RETURNS THE SQL FOR THE FLIGHT LOGS
//-------------------------------------USING ALL THE PARAMETERS
function FlightLogsOrderBy(OrderIndex, StatusIndex : byte; StartDate,
  EndDate, FlightNumber : string) : string;
var
  sOrderBy, sStatus : string;
begin
  sOrderBy := FLIGHT_LOGS_ORDER_BY_SQL[OrderIndex];
  sStatus := FLIGHT_LOGS_STATUS_SQL[StatusIndex];
  Result := 'Select [DroneFlights.DroneFlightID] AS [Flight No], ' +
            '[Serial Number], [Service Name] AS [Drone Type], ' +
            '[Suburb Name], [Date and Time of Response], ' +
            '[First Name] & " " & [Surname] AS [Assigned Staff Member] '
          + 'from DroneFlights, Drones, Services, Suburbs, ' +
            'SurveillanceReports, Staff '
          + 'where ' + sStatus + '[DroneFlights.DroneID] = ' +
            '[Drones.DroneID] AND [Drone Type] = [ServiceID] AND ' +
            '[Drones.Suburb] = [Suburbs.SuburbID] AND ' +
            '[DroneFlights.DroneFlightID] = ' +
            '[SurveillanceReports.DroneFlightID] AND ' +
            '[SurveillanceReports.StaffID] = [Staff.StaffID] AND ' +
            '[Date and Time of Response] >= #' + StartDate + '# AND ' +
            '[Date and Time of Response] <= #' + EndDate + '# AND ' +
            'cStr([DroneFlights.DroneFlightID]) Like "' + FlightNumber +
            '%" '
          + sOrderBy;
end;
```

```
//-------------------------------------RETURNS THE SQL FOR A SPECIFIC REPORT
function Report(ReportID : byte) : string;
begin
  Result := REPORTS_SQL[ReportID];
end;

end.
```

## Staff Page Access Unit

```
unit uStaffPageAccess;

interface
  uses
    ComCtrls, SysUtils, Generics.Collections, uLibrary;

  const
    FILE_NAME = 'PageAccess.txt';

  var
    PageAccessDictionary : TDictionary<string, string>;

  procedure SetUpDictionary;
  procedure AccessPages(PageControl : TPageControl; JobTitle : string);

implementation
```

```
//-------------------------------------LOADS DATA INTO THE DICTIONARY
procedure SetUpDictionary;
var
  tFl : TextFile;
  iPos : byte;
  sLine, sJobName : string;
begin
  PageAccessDictionary := TDictionary<string, string>.Create;
  AssignFile(tFl, FILE_NAME);
  Reset(tFl);
  while NOT EoF(tFl) do
  begin
    ReadLn(tFl, sLine);
    iPos := POS(TEXT_FILE_DELIMETER, sLine);
    sJobName := Copy(sLine, 1, iPos - 1);
    Delete(sLine, 1, iPos);
    PageAccessDictionary.Add(sJobName, sLine);
  end;
  CloseFile(tFl);
end;
```

```
//-------------------------------------CHANGES PAGES TO VISIBLE FOR A
//-------------------------------------SPECIFIC STAFF TYPE
procedure AccessPages(PageControl : TPageControl; JobTitle : string);
var
  K : byte;
  sPagesToAccess : string;
begin
  PageAccessDictionary.TryGetValue(UPPERCASE(JobTitle), sPagesToAccess);
  for K := 1 to Length(sPagesToAccess) do
  begin
    PageControl.Pages[StrToInt(sPagesToAccess[K])].Visible := true;
  end;
end;
end.
```

## Staff Management Unit

```
unit uStaffManagement;
```

```
interface
  uses
    uLibrary, SysUtils, Dialogs;

  function JobTitles : string;
  procedure AddStaffMember(FirtsName, Surname, Email, JobTitle : string);
  procedure DeactiveStaffMember(Email : string);
  procedure ReactivateStaffMember(Email : string);

implementation
```

```
//-------------------------------------RETURNS THE JOB TITLE NAME
function JobTitles : string;
var
  sSQL : string;
begin
  sSQL := 'Select [Job Title Name] '
        + 'from JobTitles';
  Result := objDB.ToString(sSQL, '');
end;
```

```
//-----------------------------------ADDS A NEW STAFF MEMBER
procedure AddStaffMember(FirtsName, Surname, Email, JobTitle : string);
var
  sSQL, sJobTitleID, sPassword : string;
  K : byte;
begin
  sSQL := 'Select * '
        + 'from Staff '
        + 'where [Email] = "' + Email + '"';

  //-------------------------------------CHECKS IF THE EMAIL ALREADY EXISTS
  if objDB.RecordExist(sSQL) = false then
  begin
    sSQL := 'Select [JobTitleID] '
          + 'from JobTitles '
          + 'where [Job Title Name] = "' + JobTitle + '"';
    sJobTitleID := objDB.ToString(sSQL, '');

    //-------------------------------------CREATES A RANDOM PASSWORD
    sPassword := RandomPass;

    sSQL := 'Insert into Staff '
          + '([First Name], [Surname], [Email], [Password], [Job Title])'
          + 'Values("' + FirtsName + '", "' + Surname + '", "' + Email +
            '", "' + sPassword + '", ' + sJobTitleID + ')';
    objDB.DoSQL(sSQL);

    //-------------------------------------SENDS EMAIL TO NEW STAFF MEMBER
    //-------------------------------------WITH THEIR PASSWORD
```

```
    MessageDlg('Sending Email Please Wait...', mtInformation, mbOKCancel, 0);
    objEmails.SendEmail(FirtsName + ' ' + Surname, Email,
    'Welcome to Our Staff', 'Dear ' + FirtsName + ' ' + Surname + ',' +
    #13 + #13 + 'You have been added to our staff as a ' + JobTitle +
    '.' + #13 + 'Your password is "' + sPassword + '".' + #13 + #13 +
    'You can change your password under Account Details.' + #13 + #13 +
    'Regards,' + #13 + 'Birds Eye Security');

    MessageDlg('Staff Member Successfully Added', mtInformation, mbOKCancel,
    0);
  end
  else
    MessageDlg('The staff member you are trying to add already ' +
    'works for this company. If you wish to reactivate their ' +
    'account you can do so by selecting their account above and' +
    ' reactivating it.', mtError, mbOKCancel, 0);
end;
```

```
//-------------------------------------MAKES A STAFF'S STATUS INACTIVE
procedure DeactiveStaffMember(Email : string);
var
  sSQL : string;
begin
  sSQL := 'Update Staff '
        + 'Set [Inactive Staff] = true '
        + 'where [Email] = "' + Email + '"';
  objDB.DoSQL(sSQL);

  MessageDlg('Staff Member Deactivated Successfully', mtInformation,
  mbOKCancel, 0);
end;
```

```
//-------------------------------------MAKES A STAFF'S STATUS ACTIVE
procedure ReactivateStaffMember(Email : string);
var
  sSQL : string;
begin
  sSQL := 'Update Staff '
        + 'Set [Inactive Staff] = false '
        + 'where [Email] = "' + Email + '"';
  objDB.DoSQL(sSQL);

  MessageDlg('Staff Member Reactivated Successfully', mtInformation,
  mbOKCancel, 0);
end;
```

```
end.
```

## Staff Unit

```
unit uStaff;
```

```
interface
  uses
    uDB_Source, uLibrary, Dialogs;

  Type
    TStaff = class
      private
        fID, fName, fSurname, fEmail, fPassword, fJobTitle : string;

      public
        constructor Create(Email, Password : string);
        function GetStaffID : string;
        function GetJobTitle : string;
        function GetName : string;
        function GetSurname : string;
        function GetEmail : string;
        function GetPass : string;
        procedure SetPass(Password : string);
    end;

implementation

{ TStaff }
```

```
//-----------------------------------CREATES A STAFF OBJECT
constructor TStaff.Create(Email, Password: string);
var
  sSQL : string;
  arrAccountInfo : TDynamicArray;
begin
  fEmail := Email;
  fPassword := Password;
  sSQL := 'Select [StaffID], [First Name], [Surname], [Job Title Name]'
        + 'from Staff, JobTitles '
        + 'where [Email] = "' + fEmail + '" AND [Password] = "' + fPassword +
          '" AND [Job Title] = [JobTitleID]';
  arrAccountInfo := objDB.ToArray(sSQL);
  fID := arrAccountInfo[0];
  fName := arrAccountInfo[1];
  fSurname := arrAccountInfo[2];
  fJobTitle := arrAccountInfo[3];
end;
```

```
//-----------------------------------RETURNS STAFF EMAIL ADDRESS
function TStaff.GetEmail: string;
begin
  Result := fEmail;
end;
```

```
//-----------------------------------RETURNS JOB TITLE
function TStaff.GetJobTitle: string;
begin
  Result := fJobTitle;
end;
```

```
//-----------------------------------RETURNS STAFF FIRST NAME
function TStaff.GetName: string;
begin
  Result := fName;
end;
```

```
//-----------------------------------RETURNS STAFF PASSWORD
function TStaff.GetPass: string;
begin
  Result := fPassword;
end;
```

```
//-----------------------------------RETURNS STAFF ID
function TStaff.GetStaffID: string;
begin
  Result := fID;
end;
```

```
//-----------------------------------RETURNS STAFF SURNAME
function TStaff.GetSurname: string;
begin
  Result := fSurname;
end;
```

```
//-----------------------------------CHANGES STAFF PASSWORD
procedure TStaff.SetPass(Password: string);
begin
  fPassword := Password;
end;
```

```
end.
```

## Service Request Unit

```
unit uServiceRequest;

interface
  uses
    SysUtils, uDroneLaunch, uLibrary, Generics.Collections, Dialogs,
    DateUtils, uAutoDroneDeployment;

  Type
    TServiceRequest = class
      const
        EMERGENCY_SERVICE_NAME = 'Emergency Response';
        DETAILS_TXT = 'ServiceDetails.txt';
        DETAIL_DELIMETER = '#';

      private
        fStartDate, fEndDate : TDate;
        fServiceName : string;
        fPricePerDay : real;
        fDetailsDictionary : TDictionary<string, string>;


      public
        constructor Create;
        procedure SetValues(StartDate, EndDate : TDate;
        ServiceName : string; PricePerDay : real);
        function UpdateServiceSummary : string;
        function ConfirmServiceRequest : string;
        procedure RequestService(UserID, Time : string);
    end;

implementation

{ TServiceRequest }
```

```
//--------------------------------------RETURNS A DETAILED LIST OF THE DETAILS
//-------------------------------------OF A CLIENTS SEVICE REQUEST
function TServiceRequest.ConfirmServiceRequest: string;
begin
  Result := 'Are you sure you want to request this service:' + #13
          + 'Service Name:' + #13
          + fServiceName + #13 + #13
          + UpdateServiceSummary;
end;
```

```
constructor TServiceRequest.Create;
var
  tFl : TextFile;
  sLine, sService, sDetails : string;
  iPos : byte;
begin
  AssignFile(tFl, DETAILS_TXT);
  Reset(tFl);

  //-------------------------------------LOADS DATA INTO THE DICTIONARY
  fDetailsDictionary := TDictionary<string, string>.Create;
  while NOT EoF(tFl) do
  begin
    Readln(tFl, sLine);
    iPos := POS(DETAIL_DELIMETER, sLine);
    sService := UPPERCASE(Copy(sLine, 1, iPos - 1));
    Delete(sLine, 1, iPos);
    sDetails := sLine;
    fDetailsDictionary.Add(sService, sDetails);
  end;
  CloseFile(tFl);
end;
```

```
//-------------------------------------ADDS A CLIENTS SERVICE REQUEST TO
//-------------------------------------THE DATABASE
procedure TServiceRequest.RequestService(UserID, Time: string);
var
  sSQL, sServiceID, sOldDates, sUserServiceID, sSuburbID : string;
  K, iNumDays : byte;
  dDate : TDate;
begin
  sSQL := 'Select [ServiceID] '
        + 'from Services '
        + 'where [Service Name] = "' + fServiceName + '"';
  sServiceID := objDB.ToString(sSQL, '');

  sSQL := 'Select [Suburb] '
        + 'from Users '
        + 'where [UserID] = ' + UserID;
  sSuburbID := objDB.ToString(sSQL, '');

  sSQL := 'Select * '
        + 'from UserServices '
        + 'where [UserID] = ' + UserID + ' AND [ServiceID] = ' +
          sServiceID + ' AND (([Start Date] <= #' + DateToStr(fStartDate) +
          '# AND [End Date] >= #' + DateToStr(fStartDate) + '#) OR ' +
          '([Start Date] <= #' + DateToStr(fEndDate) + '# AND ' +
          '[End Date] >= #' + DateToStr(fEndDate) + '#))';

  //-------------------------------------CHECKS IF CLIENT HAS AN OVERLAPPING
  //-------------------------------------SERVICE REQUEST
  if objDB.RecordExist(sSQL) = false then
  begin
    sSQL := 'Select * '
          + 'from Drones '
```

```
        + 'where [DroneID] NOT IN (Select [DroneID] from ' +
          'DroneFlights where Format([Date and Time of Response], ' +
          '"Short Date") >= #' + DateToStr(fStartDate) + '# AND ' +
          'Format([Date and Time of Response], "Short Date") <= ' +
          '#' + DateToStr(fStartDate) + '#) AND [Drone Type] = ' +
          sServiceID + ' AND [Suburb] = ' + sSuburbID;
  if objDB.RecordExist(sSQL) = true then
  begin
    sSQL := 'Insert into UserServices '
          + '([UserID], [ServiceID], [Start Date], [End Date]) '
          + 'Values(' + UserID + ', ' + sServiceID + ', #' +
            DateToStr(fStartDate) + '#, #' + DateToStr(fEndDate) + '#)';
    objDB.DoSQL(sSQL);

    sSQL := 'Select [UserServiceID] '
          + 'from UserServices '
          + 'where [UserID] = ' + UserID + ' AND [ServiceID] = ' +
            sServiceID + ' AND [Start Date] = #' + DateToStr(fStartDate) +
            '# AND [End Date] = #' + DateToStr(fEndDate) + '#';
    sUserServiceID := objDB.ToString(sSQL, '');

    if UpperCase(fServiceName) = UpperCase(EMERGENCY_SERVICE_NAME) then
      ScheduleFlight(fServiceName, DateToStr(fStartDate),
      TimeToStr(Now()), UserID, sUserServiceID, sSuburbID)
    else
    begin
      //-------------------------------------SCHEDULES FLIGHTS FOR EACH
      //-------------------------------------DAY THE CLIENT REQUESTED
      iNumDays := Trunc(fEndDate - fStartDate) + 1;
      ScheduleFlight(fServiceName, DateToStr(fStartDate), Time,
      UserID, sUserServiceID, sSuburbID);
      dDate := fStartDate;
      for K := 2 to iNumDays do
      begin
        dDate := IncDay(dDate, 1);
        ScheduleFlight(fServiceName, DateToStr(dDate), Time,
        UserID, sUserServiceID, sSuburbID);
      end;
    end;
    DeployDrones;
    MessageDlg('Your Service Request was Submitted Successfully',
    mtInformation, mbOKCancel, 0);
  end
  else
    MessageDlg('All drones are currently busy between these ' +
    'dates. We are actively deploying more drones in order to' +
    ' meet demand. Please be patient until more drones are ' +
    'deployed in your area.', mtInformation, mbOKCancel, 0);
end
else
begin
  //-------------------------------------CREATES A MESSAGE TELLING THE
  //-------------------------------------CLIENT ABOUT THE OVERLAP AND
  //-------------------------------------ITS DETAILS
  sSQL := 'Select [Start Date] & " and " & [End Date] '
        + 'from UserServices '
```

```
          + 'where [UserID] = ' + UserID + ' AND [ServiceID] = ' +
          sServiceID + ' AND (([Start Date] <= #' + DateToStr(fStartDate) +
          '# AND [End Date] >= #' + DateToStr(fStartDate) + '#) OR ' +
          '([Start Date] <= #' + DateToStr(fEndDate) + '# AND ' +
          '[End Date] >= #' + DateToStr(fEndDate) + '#))';
    sOldDates := objDB.ToString(sSQL, '');

    MessageDlg('You already have requested this service ' +
    'between these dates.' + #13 + 'The service you requested ' +
    'are between these dates ' + sOldDates,
    mtInformation, mbOKCancel, 0);
  end;
end;
```

```
//-------------------------------------CHANGES ALL THE VALUES IN THE OBJECT
procedure TServiceRequest.SetValues(StartDate, EndDate: TDate;
  ServiceName: string; PricePerDay: real);
begin
  fStartDate := StartDate;
  fEndDate := EndDate;
  fServiceName := ServiceName;
  fPricePerDay := PricePerDay;
end;
```

```
//-------------------------------------RETURNS A SUMMARY OF THE SERVICE WHICH
//-------------------------------------IS CURRENTLY SELECTED
function TServiceRequest.UpdateServiceSummary: string;
var
  rTot : real;
  rDays : real;
  sDetails : string;
begin
  fDetailsDictionary.TryGetValue(UPPERCASE(fServiceName), sDetails);
  if UpperCase(fServiceName) <> UpperCase(EMERGENCY_SERVICE_NAME) then
  begin
    rDays := fEndDate - fStartDate + 1;
    rTot := fPricePerDay * rDays;
    Result := 'Details:' + #13 + sDetails + #13 + #13 +
    'Service Period:' + #13 + DateToStr(fStartDate) + ' - ' +
    DateToStr(fEndDate) + #13 + #13 + 'Total Cost:' + #13 +
    FloatToStr(rDays) + ' day(s) x ' + FloatToStrF(fPricePerDay, ffCurrency,
    10, 2) + ' = ' + FloatToStrF(rTot, ffCurrency, 10, 2);
  end
  else
  begin
    Result := 'Details:' + #13 + sDetails + #13 + #13 +
    'Service Period:' + #13 + 'Immediate Response' + #13 + #13 +
    'Total Cost:' + #13 + FloatToStrF(fPricePerDay, ffCurrency,10, 2);
  end;
end;
```

```
end.
```

## Login Form Manager Unit

```
unit uLoginFormManager;

interface
  uses
    Forms, Dialogs, SysUtils, uDB_Source, frmStaff, frmClient, uClient,
    uStaff, uLibrary, Controls, uAccountDetails, uHelp;

  var
    objAccountDetails : TAccountDetails;
    objHelp : THelp;

  procedure CreateObjs;

  //-----------------------------------------------SIGN UP
  function PopulateSuburbDropDown : string;
  function FieldsConfirmed(Password, RetypedPass, Email, RetypedEmail :
  string) : boolean;
  function EmailVerified(Email : string) : boolean;
  function RequiredFieldsFilled(FirstName, Surname, Email, RetypedEmail,
  Password, RetypedPass, AddressLine1, SuburbName, PostCode : string)
  : boolean;
  procedure RegisterNewUser(FirstName, Surname, Email, Password, AddressLine1,
  SuburbName, PostCode : string; LoginForm : TForm);
  procedure SignUpHelp;

  //----------------------------------------------RECOMMEND SUBURB
  function AlreadyInSuburbs(PostCode : string) : boolean;
  procedure RecommendSuburb(SuburbName, PostCode, Email : string);

  //----------------------------------------------LOGIN
  procedure Login(Email, Password : string; IsStaff : boolean; LoginForm :
  TForm);
  procedure ForgotPass(Email : string; Staff : boolean);
  procedure LoginHelp;

implementation
```

```
//-------------------------------------CHECKS IF THE SUBURB A IS ALREADY IN
//-------------------------------------THE DATABASE
function AlreadyInSuburbs(PostCode : string) : boolean;
var
  sSQL : string;
begin
  sSQL := 'Select * '
        + 'from Suburbs '
        + 'where [Postal Code] = "' + PostCode + '"';
  Result := objDB.RecordExist(sSQL);
end;
```

```
//-------------------------------------ADDS/UPDATES A RECOMMENDED SUBRUB
procedure RecommendSuburb(SuburbName, PostCode, Email : string);
var
  sSQL, sRecommendID : string;
begin
  sSQL := 'Select * '
        + 'from RecommendedSuburbs '
        + 'where [Suburb Name] = "' +
  SuburbName + '" AND [Postal Code] = "' + PostCode + '"';

  //-------------------------------------CHECKS IF ITS ALREADY BEEN RECOMMENDED
  if objDB.RecordExist(sSQL) = true then
  begin
    //-------------------------------------INCREASES NUMBER OF RECOMMENDATIONS
    sSQL := 'Update RecommendedSuburbs'
          + ' set [Number of Recommendations] = ' +
            '[Number of Recommendations] + 1'
          + ' where [Suburb Name] = "' + SuburbName + '" AND ' +
            '[Postal Code] = "' + PostCode + '"';
    objDB.DoSQL(sSQL);
  end
  else
  begin
    //-------------------------------------ADDS A NEW RECOMMENDED SUBURB
    sSQL := 'Insert into RecommendedSuburbs'
          + ' ([Suburb Name], [Postal Code], [Number of Recommendations])'
          + ' Values("' + SuburbName + '", "' + PostCode + '", 1)';
    objDB.DoSQL(sSQL);
  end;

  sSQL := 'Select [RecSuburbID] '
        + 'from RecommendedSuburbs '
        + 'where [Suburb Name] = "' + SuburbName + '" AND [Postal Code]' +
          ' = "' + PostCode + '"';
  sRecommendID := objDB.ToString(sSQL, '');

  //-------------------------------------ADDS EMAIL TO NOTIFICATIONS
  sSQL := 'Insert into NewSuburbEmailNotify'
        + ' ([RecSubID], [User Email])'
        + ' Values(' + sRecommendID + ', "' + Email + '")';
  objDB.DoSQL(sSQL);
end;
```

```
//-------------------------------------CREATES OBJECTS NEED ACROSS THE
//-------------------------------------WHOLE APPLICATION
procedure CreateObjs;
begin
  CreateGlobalObjs;
  objHelp := THelp.Create;
end;
```

```
//-------------------------------------RETURNS ALL THE SUBURBS' DETAILS
function PopulateSuburbDropDown: string;
var
  sSQL : string;
begin
  sSQL := 'Select [Suburb Name], [Postal Code] '
        + 'from Suburbs';
  Result := objDB.ToString(sSQL, SUBURB_DELIMETER);
end;

function FieldsConfirmed(Password, RetypedPass, Email, RetypedEmail :
  string) : boolean;
begin
  if (Password = RetypedPass) AND (Email = RetypedEmail) then
    Result := true
  else
  begin
    Result := false;
    MessageDlg('The email or passsword you typed doesn''t match the ' +
    'confirm password or email. Please ensure these field are the same.',
    mtInformation, mbOKCancel, 0);
  end;
end;
```

```
//-------------------------------------CHECKS IF THE EMAIL IS A LEGITIMATE
//-------------------------------------EMAIL ADDRESS
function EmailVerified(Email : string) : boolean;
var
  iPosFirstAtSign, iPos : byte;
begin
  Result := true;
  if POS(' ', Email) <> 0 then
    Result := false
  else
  begin
    iPosFirstAtSign := POS('@', Email);
    if (iPosFirstAtSign <> 0) AND (iPosFirstAtSign <> 1) then
    begin
      Delete(Email, 1, iPosFirstAtSign);
      iPos := POS('@', Email);

      if iPos <> 0 then
        Result := false
      else
      begin
        iPos := POS('.', Email);
        if (iPos = 0) then
          Result := false;
      end;
    end
    else
      Result := false;
  end;
```

```
  if Result = false then
    MessageDlg('The email you entered is not verified. Please enter ' +
    'a authentic email address.', mtInformation, mbOKCancel, 0);
end;
```

```
//-------------------------------------CHECKS IF ALL FIELDS ARE FILLED OUT
function RequiredFieldsFilled(FirstName, Surname, Email, RetypedEmail,
  Password, RetypedPass, AddressLine1, SuburbName, PostCode : string)
  : boolean;
begin
  if (FirstName <> '') AND (Surname <> '') AND (Email <> '') AND
  (Password <> '') AND (AddressLine1 <> '') AND (SuburbName <> '') AND
  (PostCode <> '') AND (RetypedEmail <> '') AND (RetypedEmail <> '') then
  begin
    Result := true;
  end
  else
  begin
    Result := false;
    MessageDlg('All field are required in order to register, please ' +
    'ensure all fields are filled out.', mtInformation, mbOKCancel, 0);
  end;
end;
```

```
//-------------------------------------ADDS A NEW CLIENT TO THE DATABASE
procedure RegisterNewUser(FirstName, Surname, Email,
  Password, AddressLine1, SuburbName, PostCode : string; LoginForm : TForm);
var
  sSQL, sSuburbID : string;
begin
  if MessageDlg('Please Confirm That The Information You Have Entered is ' +
  'Correct' + #13 + #13 + 'First Name:' + #13 + FirstName + #13 + #13 +
  'Surname: '  + #13 + Surname + #13 + #13 + 'Email:' + #13 +
  Email + #13 + #13 + 'Password:' + #13 + Password + #13 + #13 +
  'Address:' + #13 + AddressLine1 + #13 + SuburbName + #13 + PostCode,
  mtConfirmation, mbYesNo, 0) = mrYes then
  begin
    sSQL := 'Select * '
          + 'from Users '
          + 'where [Email] = "' + Email + '"';
    if objDB.RecordExist(sSQL) = false then
    begin
      sSQL := 'Select [SuburbID] '
            + 'from Suburbs '
            + 'where [Suburb Name] = "' + SuburbName + '" AND ' +
              '[Postal Code] = "' + PostCode + '"';

      sSuburbID := objDB.ToString(sSQL, '');

      sSQL := 'Insert into Users '
            + '([First Name], [Surname], [Email], [Password], ' +
              '[Address Line 1], [Suburb]) '
            + 'Values("' + FirstName + '", "' + Surname + '", "' + Email +
              '", "' + Password + '", "' + AddressLine1 + '", ' + sSuburbID +
              ')';
```

```
      objDB.DoSQL(sSQL);
      Login(Email, Password, false, LoginForm);
    end
    else
      MessageDlg('Your email is already registered within our database.' +
      ' Click Already Registered in order to sign into your account.',
      mtInformation, mbOKCancel, 0);
  end;
end;
```

```
//-------------------------------------LOADS HELP FOR A USER ON SIGN UP PAGE
procedure SignUpHelp;
begin
  objHelp.LoadHelp('Sign Up');
end;
```

```
//-------------------------------------LOGS A USER INTO THEIR ACCOUNT
procedure Login(Email, Password : string; IsStaff : boolean; LoginForm :
TForm);
var
  sSQL, sPass : string;
  bExists : boolean;
  objClient : TClient;
  objStaff : TStaff;
  objClientForm : TClientForm;
  objStaffForm : TStaffForm;
begin
  if IsStaff = true then
    sSQL := 'Select * '
          + 'from Staff '
          + 'where [Email] = "' + Email + '" AND [Inactive Staff] = False'
  else
    sSQL := 'Select * '
          + 'from Users '
          + 'where [Email] = "' + Email + '" AND [Inactive User] = False';

  bExists := objDB.RecordExist(sSQL);

  if bExists = true then
  begin
    if IsStaff = true then
      sSQL := 'Select [Password] '
            + 'from Staff '
            + 'where [Email] = "' + Email + '"'
    else
      sSQL := 'Select [Password] '
            + 'from Users '
            + 'where [Email] = "' + Email + '"';

    sPass := objDB.ToString(sSQL, '');

    //-------------------------------------CHECKS IF PASSWORD IS CORRECT
    //-------------------------------------INCLUDING CASE SENSITIVITY
    if (Password = sPass) AND (IsStaff = true) then
```

```
    begin
      objStaff := TStaff.Create(Email, Password);
      objStaffForm := TStaffForm.Create(Application);
      objAccountDetails := TAccountDetails.Create(LoginForm, false);
      objStaffForm.FormSetup(objStaff, objAccountDetails, objHelp);
      objStaffForm.Show;
      LoginForm.Hide;
    end
    else if (Password = sPass) AND (IsStaff = false) then
    begin
      objClient := TClient.Create(Email, Password);
      objClientForm :=  TClientForm.Create(Application);
      objAccountDetails := TAccountDetails.Create(LoginForm, true);
      objClientForm.FormSetup(objClient, objAccountDetails, objHelp);
      objClientForm.Show;
      LoginForm.Hide;
    end
    else
      MessageDlg('Your password is incorrect. If you are a member ' +
      'of staff please remember to check the staff member login box.',
      mtInformation, mbOKCancel, 0);
  end
  else
    MessageDlg('Your email is incorrect. If you are a member ' +
    'of staff please remember to check the staff member login box.',
    mtInformation, mbOKCancel, 0);
end;
```

```
//-------------------------------------ALLOWS A USER TO REQUEST A TEMPORARY
//-------------------------------------PASSWORD
procedure ForgotPass(Email : string; Staff : boolean);
var
  sSQL, sPassword : string;
begin
  if Staff = true then
    sSQL := 'Select * '
          + 'from Staff '
          + 'where [Email] = "' + Email + '"'
  else
    sSQL := 'Select * '
          + 'from Users '
          + 'where [Email] = "' + Email + '"';

  if objDB.RecordExist(sSQL) = true then
  begin
    sPassword := RandomPass;

    if Staff = true then
      sSQL := 'Update Staff '
            + 'Set [Password] = "' + sPassword + '" '
            + 'where [Email] = "' + Email + '"'
    else
      sSQL := 'Update Users '
            + 'Set [Password] = "' + sPassword + '" '
            + 'where [Email] = "' + Email + '"';
```

```
    objDB.DoSQL(sSQL);

    //-------------------------------------SENDS USER A RANDOM PASSWORD
    MessageDlg('Sending Email Please Wait...', mtInformation, mbOKCancel, 0);
    objEmails.SendEmail('', Email, 'Forgot Password Request', 'Here ' +
    'is your new password: ' + sPassword + #13 + #13 + 'You can ' +
    'change your password in Account Details once you have signed in.' +
    #13 + #13 + 'Regards,' + 'Birds Eye Security');
  end
  else
    MessageDlg('Your email does not exist within our database. ' +
    'Please sign up in order to create an account.', mtInformation,
    mbOKCancel, 0);
end;
```

```
//-------------------------------------LOADS HELP FOR A USER ON LOGIN PAGE
procedure LoginHelp;
begin
  objHelp.LoadHelp('Login');
end;
```

```
end.
```

## Library Unit

```
unit uLibrary;

interface
  uses
    uDB_Source, uEmails, Dialogs, SysUtils;

  const
    SECTION_BORDER = '-----------------------------------------------------' +
                     '-----------------------------------------------------';
    SUBURB_DELIMETER = ', ';
    TEXT_FILE_DELIMETER = '#';
    DB_FILE_NAME = 'BirdsEyeSecuritySurveillance_Data.mdb';

  var
    //-------------------------------------GLOBAL VARIABLES
    objDB : TDBClass;
    objEmails : TEmails;

  procedure CreateGlobalObjs;
  function PasswordCriteriaMet(Password: string): boolean;
  function RandomPass : string;
  function ValidatePostal(PostalCode : string) : boolean;

implementation
```

```
procedure CreateGlobalObjs;
begin
  objDB := TDBClass.Create(DB_FILE_NAME);
  objEmails := TEmails.Create;
end;
```

```
//-------------------------------------CHECKS THAT A PASSWORD MEETS PREDEFINED
//-------------------------------------CRITERIA
function PasswordCriteriaMet(Password: string): boolean;
var
  K, iLength, iCharCount, iNumCount: byte;
  cPass: char;
begin
  Result := true;
  iLength := Length(Password);
  if (iLength < 8) OR (iLength > 15) then
    Result := false
  else if POS(' ', Password) <> 0 then
    Result := false
  else
  begin
    //-------------------------------------ENSURES THE PASSWORD HAS ONE
    //-------------------------------------CHARACTER AND ONE NUMBER
    K := 1;
    iNumCount := 0;
    iCharCount := 0;
    while (K <= iLength) AND ((iCharCount = 0) OR (iNumCount = 0)) do
```

```
    begin
      cPass := UPCASE(Password[K]);
      if cPass IN ['A' .. 'Z'] then
        iCharCount := iCharCount + 1
      else if cPass IN ['0' .. '9'] then
        iNumCount := iNumCount + 1;
      K := K + 1;
    end;

    if (iCharCount = 0) OR (iNumCount = 0) then
      Result := false;
  end;

  if Result = false then
    MessageDlg('Your password does not meet the following criteria:' + #13 +
        '- Between 8 and 15 Characters' + #13 + '- No Spaces' + #13 +
        '- Atleast one character' + #13 + '- Atleast one number',
      mtInformation, mbOKCancel, 0);
end;
```

```
//-------------------------------------CREATES A RANDOM PASSWORD THAT MEETS
//-------------------------------------THE PREDEFINED CRITERIA
function RandomPass : string;
var
  K : byte;
begin
  Randomize;
  Result := Chr(Random(26) + 97);
  for K := 2 to 7 do
  begin
    if Random(2) + 1 = 1 then
    begin
      Result := Result + Chr(Random(26) + 97);
    end
    else
    begin
      Result := Result + IntToStr(Random(10));
    end;
  end;
  Result := Result + IntToStr(Random(10));
end;
```

```
//-----------------------------------CHECKS IF THE POSTAL CODE IS ONLY 4
//-----------------------------------CHARS LONG AND CONTAINS ONLY DIGITS
function ValidatePostal(PostalCode : string) : boolean;
var
  K, iLength : byte;
begin
  Result := true;
  K := 1;
  iLength := Length(PostalCode);
  if iLength <> 4 then
    Result := false
  else
  begin
    while (K <= iLength) AND (Result = true) do
      if NOT (PostalCode[K] IN ['0'..'9']) then
        Result := false
      else
        K := K + 1;
  end;

  if Result = false then
    MessageDlg('A postal code can only contain digits and be 4 ' +
    'characters long', mtError, mbOKCancel, 0);
end;
```

```
end.
```

## Help Unit

```
unit uHelp;

interface
  uses
    ShellAPI, Generics.Collections, uLibrary, Windows, SysUtils, DIalogs;

  Type
    THelp = class
      const
        FILE_NAME = 'HelpURLs.txt';

      private
        fHelpExtensionDictionary : TDictionary<string, string>;

      public
        constructor Create;
        procedure LoadHelp(PageName : string);
    end;

implementation

{ THelp }
//-------------------------------------LOADS DATA INTO THE DICTIONARY
constructor THelp.Create;
var
  tFl : TextFile;
  sLine, sPageName, sURL : string;
  iPos : byte;
begin
  fHelpExtensionDictionary := TDictionary<string, string>.Create;
  AssignFile(tFl, FILE_NAME);
  Reset(tFl);
  while NOT EoF(tFl) do
  begin
    ReadLn(tFl, sLine);
    iPos := POS(TEXT_FILE_DELIMETER, sLine);
    sPageName := Copy(sLine, 1, iPos - 1);
    Delete(sLine, 1, iPos);
    sURL := sLine;
    fHelpExtensionDictionary.Add(UPPERCASE(sPageName), sURL);
  end;
  CloseFile(tFl);
end;
```

```
//-------------------------------------DISPLAYS HELP TO A USER ACCORDING TO
//-------------------------------------A SPECIFIC PAGE
procedure THelp.LoadHelp(PageName: string);
var
  sURL : string;
begin
  fHelpExtensionDictionary.TryGetValue(UPPERCASE(PageName), sURL);
  ShellExecute(0, 'open', PChar(sURL), nil, nil, SW_SHOWNORMAL);
end;
```

```
end.
```

## Emails Unit

```
unit uEmails;

interface
  uses
    IdMessage, IdBaseComponent, IdComponent, IdTCPConnection, IdTCPClient,
    IdExplicitTLSClientServerBase, IdMessageClient, IdSMTPBase, IdSMTP,
    IdIOHandler, IdIOHandlerSocket, IdIOHandlerStack, IdSSL, IdSSLOpenSSL,
    Dialogs, SysUtils;

  Type
    TEmails = class
      const
        COMPANY_EMAIL = 'birdseyesecuity@gmail.com';
        COMPANY_EMAIL_PASS = 'BirdsEyeSecure01';

      private
        fSSL : TIdSSLIOHandlerSocketOpenSSL;
        fSTMP : TIdSMTP;

      public
        constructor Create;
        procedure SendEmail(RecipientName, EmailAddress, Subject, BodyMessage
        : string);
    end;

implementation

{ TEmails }

//-------------------------------------SETS UP ALL OBJECTS NEEDED TO
//-------------------------------------SEND EMAILS
constructor TEmails.Create;
begin
  fSSL := TIdSSLIOHandlerSocketOpenSSL.Create(nil);
  fSTMP := TIdSMTP.Create(nil);

  //----------------------------------------------SETTING UP SECURE SSL
  With fSSL do
  begin
    Destination := 'smtp.gmail.com:587';
    Host := 'smtp.gmail.com';
    Port := 587;
    SSLOptions.Method := sslvTLSv1;
    SSLOptions.Mode := sslmUnassigned;
    SSLOptions.VerifyMode := [];
    SSLOptions.VerifyDepth := 0;
  end;

  //----------------------------------------------SETTING SMTP DATA
  with fSTMP do
  begin
    Host := 'smtp.gmail.com';
```

```
    Port := 587;
    Username := COMPANY_EMAIL;
    Password := COMPANY_EMAIL_PASS;
    IOHandler := fSSL;
    AuthType := satDefault;
    UseTLS := utUseExplicitTLS;
  end;
end;
```

```
//-------------------------------------SENDS A USER AN EMAIL BASED ON
//-------------------------------------THE PARAMETERS
procedure TEmails.SendEmail(RecipientName, EmailAddress, Subject, BodyMessage
: string);
var
  mailMessage : TIdMessage;
begin
  mailMessage := TIdMessage.Create(nil);

  with mailMessage.Recipients.Add do
  begin
    Name := RecipientName;
    Address := EmailAddress;
  end;

  //-------------------------------------SETS UP THE EMAILS MESSAGE DETAILS
  mailMessage.From.Name := 'Birds Eye Security';
  mailMessage.From.Address :=  COMPANY_EMAIL;
  mailMessage.Subject := Subject;
  mailMessage.Body.Text := BodyMessage;
  mailMessage.Priority := mpHigh;

  //-------------------------------------CHECKS IF THE EMAILS HAS BEEN SENT
  TRY
    fSTMP.Connect();
    fSTMP.Send(mailMessage);
    MessageDlg('Email was Sent Succesfully', mtInformation, mbOKCancel, 0);
    fSTMP.Disconnect();
  except on e:Exception do
    begin
      MessageDlg('Email with email address ' + EmailAddress + ' was ' +
      'not sent, due to ' + e.Message, mtError, mbOKCancel, 0);
      fSTMP.Disconnect();
    end;
  end;
end;
```

```
end.
```

## Drone Management Unit

```
unit uDroneManagement;

interface
  uses
    uLibrary, SysUtils, Dialogs;

  function DroneOrderBy(OrderBy : string) : string;
  function ToStringSuburbs : string;
  function ToStringDroneTypes : string;
  function GenerateSerialNumber : string;
  function SerialNumExists(SerialNumber : string) : boolean;
  procedure AddNewDrone(SerialNumber, Suburb, DroneType : string);
  procedure EditDrone(OldSerialNumber, NewSerialNumber, Status, Suburb,
  DroneType : string);
  function ValidateSerialNumber(SerialNumber : string) : boolean;

implementation
```

```
//--------------------------------------RETURNS SQL FOR A RECORD ORDER
function DroneOrderBy(OrderBy : string) : string;
begin
  Result := 'Select [Serial Number], [Suburb Name], [Service Name] AS' +
            ' [Drone Type], [Status] '
          + 'from Drones, Suburbs, Services '
          + 'where [Suburb] = [SuburbID] AND [Drone Type] = [ServiceID] '
          + 'order by [' + Trim(OrderBy) + ']';
end;
```

```
//-------------------------------------RETURNS ALL SUBURBS' DETAILS
function ToStringSuburbs : string;
var
  sSQL : string;
begin
  sSQL := 'Select [Suburb Name], [Postal Code] '
        + 'from Suburbs ';
  Result := objDB.ToString(sSQL, SUBURB_DELIMETER);
end;
```

```
//-------------------------------------RETURNS ALL SERVICE NAMES
function ToStringDroneTypes : string;
var
  sSQL : string;
begin
  sSQL := 'Select [Service Name] '
        + 'from Services ';
  Result := objDB.ToString(sSQL, '');
end;
```

```
//-------------------------------------GENERATES A RANDOM SERIAL NUMBER
function GenerateSerialNumber : string;
var
  K : byte;
begin
  Result := '';
  for K := 1 to 7 do
    Result := Result + IntToStr(Random(10));
end;
```

```
//-------------------------------------CHECKS IF SERIAL NUMBER EXISTS
function SerialNumExists(SerialNumber : string) : boolean;
var
  sSQL : string;
begin
  sSQL := 'Select [Serial Number] '
        + 'from Drones '
        + 'where [Serial Number] = "' + SerialNumber + '"';
  Result := objDB.RecordExist(sSQL);

  if Result = true then
    MessageDlg('The drones new serial number already exists ' +
    'please ensure that the serial number is unique', mtError, mbOKCancel,
    0);
end;
```

```
//-------------------------------------ADDS A NEW DRONE
procedure AddNewDrone(SerialNumber, Suburb, DroneType : string);
var
  sSQL, sSuburbID, sDroneTypeID : string;
begin
  if SerialNumExists(SerialNumber) = false then
  begin
    sSQL := 'Select [SuburbID] '
          + 'from Suburbs '
          + 'where [Suburb Name] & ", " & [Postal Code] = "' + Suburb + '"';
    sSuburbID := objDB.ToString(sSQL, '');

    sSQL := 'Select [ServiceID] '
          + 'from Services '
          + 'where [Service Name] = "' + DroneType + '"';
    sDroneTypeID := objDB.ToString(sSQL, '');

    sSQL := 'Insert into Drones '
          + '([Serial Number], [Suburb], [Drone Type]) '
          + 'Values("' + SerialNumber + '", ' + sSuburbID + ', ' +
            sDroneTypeID + ')';
    objDB.DoSQL(sSQL);

    MessageDlg('Drone Added Successfully', mtInformation, mbOKCancel, 0);
  end
  else
    MessageDlg('The serial number already exists please ensure that ' +
    'the serial number is unique', mtError, mbOKCancel, 0);
end;
```

```
//-------------------------------------UPDATES DATA FOR A SPECIFIC DRONE
procedure EditDrone(OldSerialNumber, NewSerialNumber, Status, Suburb,
DroneType : string);
var
  sSQL, sSuburbID, sDroneTypeID, sDroneID : string;
begin
  sSQL := 'Select [SuburbID] '
        + 'from Suburbs '
        + 'where [Suburb Name] & ", " & [Postal Code] = "' + Suburb + '"';
  sSuburbID := objDB.ToString(sSQL, '');

  sSQL := 'Select [ServiceID] '
        + 'from Services '
        + 'where [Service Name] = "' + DroneType + '"';
  sDroneTypeID := objDB.ToString(sSQL, '');

  sSQL := 'Select [DroneID] '
        + 'from Drones '
        + 'where [Serial Number] = "' + OldSerialNumber + '"';
  sDroneID := objDB.ToString(sSQL, '');

  //-------------------------------------UPDATES DRONE DATA
  if (SerialNumExists(NewSerialNumber) = false) AND (ValidateSerialNumber(
  NewSerialNumber) = true) then
  begin
    sSQL := 'Update Drones '
          + 'Set [Status] = "' + Status + '" '
          + 'where [Serial Number] = "' + NewSerialNumber + '"';
    objDB.DoSQL(sSQL);

    sSQL := 'Update Drones '
          + 'Set [Suburb] = ' + sSuburbID + ' '
          + 'where [Serial Number] = "' + NewSerialNumber + '"';
    objDB.DoSQL(sSQL);

    sSQL := 'Update Drones '
          + 'Set [Drone Type] = ' + sDroneTypeID + ' '
          + 'where [Serial Number] = "' + NewSerialNumber + '"';
    objDB.DoSQL(sSQL);

    if OldSerialNumber <> NewSerialNumber then
    begin
      sSQL := 'Update Drones '
            + 'Set [Serial Number] = "' + NewSerialNumber + '" '
            + 'where [Serial Number] = "' + OldSerialNumber + '"';
      objDB.DoSQL(sSQL);
    end;

    MessageDlg('Drone Details Changed Successfully', mtInformation,
    mbOKCancel, 0);
  end;
end;
```

```
//-------------------------------------ENSURES THAT THE SERIAL NUMBER IS
//-------------------------------------ACCORDING TO PREDEFINED PARAMETERS
function ValidateSerialNumber(SerialNumber : string) : boolean;
var
  iLength, K : byte;
begin
  Result := true;
  K := 1;

  iLength := Length(SerialNumber);
  if iLength <> 6 then
    Result := false
  else
  begin
    //-------------------------------------CHECKS THAT THE SERIAL NUMBER
    //-------------------------------------CONTAINS ONLY DIGITS
    while (Result = true) AND (K <= iLength) do
    begin
      if NOT (SerialNumber[K] IN ['0'..'9']) then
        Result := false
      else
        K := K + 1;
    end;
  end;

  if Result = false then
    MessageDlg('The drones new serial number can only contain numbers ' +
    'and must be exactly 6 characters long.', mtError, mbOKCancel, 0);
end;
```

```
end.
```

## Drone Launch Unit

```
unit uDroneLaunch;
```

```
interface
  uses
    uDB_Source, uLibrary, SysUtils;

  procedure AssignStaffToService(DroneFlightID : string);
  procedure ScheduleFlight(Service, Date, Time, UserID, UserServiceID,
  SuburbID : string);
  procedure AssignUserServiceToFlight(UserServiceID, DroneFlightID: string);

implementation

{ TDroneLaunch }
```

```
//-------------------------------------ASSIGNS A STAFF MEMBER TO A FLIGHT
procedure AssignStaffToService(DroneFlightID : string);
var
  sSQL : string;
  arrStaffMembers : TDynamicArray;
  iRandom : word;
begin
  sSQL := 'Select [StaffID] '
        + 'from Staff, JobTitles '
        + 'where [Job Title] = [JobTitleID] AND [Job Title Name] = ' +
          '"Surveillance Analyst"';
  arrStaffMembers := objDB.ToArray(sSQL);

  Randomize;
  iRandom := Random(Length(arrStaffMembers));
  sSQL := 'Insert into SurveillanceReports '
        + '([DroneFlightID], [StaffID]) '
        + 'Values(' + DroneFlightID + ', ' + arrStaffMembers[iRandom] + ')';
  objDB.DoSQL(sSQL);
end;
```

```
//-------------------------------------ASSIGNS A USERS SERVICE REQUEST TO
//-------------------------------------A SPECIFIC FLIGHT
procedure AssignUserServiceToFlight(UserServiceID, DroneFlightID: string);
var
  sSQL : string;
begin
  sSQL := 'Insert into FlightToServiceAssignment '
        + '([UserServiceID], [DroneFlightID]) '
        + 'Values(' + UserServiceID + ', ' + DroneFlightID + ')';
  objDB.DoSQL(sSQL);
end;
```

```
//-------------------------------------SCHEDULES ALL THE FLIGHTS FOR
//-------------------------------------A SERVICE REQUEST
procedure ScheduleFlight(Service, Date, Time, UserID, UserServiceID,
  SuburbID : string);
var
  sSQL, sServiceID, sDroneFlightID : string;
  arrAvailableDrones : TDynamicArray;
  iRandom : byte;
begin
  sSQL := 'Select [ServiceID] '
        + 'from Services '
        + 'where [Service Name] = "' + Service + '"';
  sServiceID := objDB.ToString(sSQL, '');

  //-------------------------------------FINDS ALL AVAILABLE DRONES FOR THE
  //-------------------------------------SPECIFIC DATE
  sSQL := 'Select [DroneID] '
        + 'from Drones '
        + 'where [Drone Type] = ' + sServiceID + 'AND [Suburb] = ' +
          SuburbID + ' AND [DroneID] NOT IN (Select [DroneID] ' +
          'from DroneFlights where Format(' +
          '[Date and Time of Response], "Short Date") = #' +
          Date + '# AND [Status] <> "Completed")';
  arrAvailableDrones := objDB.ToArray(sSQL);
  if Length(arrAvailableDrones) <> 0 then
  begin
    //-------------------------------------ASSIGNS A RANDOM DRONE TO
    //-------------------------------------A SERVICE REQUEST
    Randomize;
    iRandom := Random(Length(arrAvailableDrones));

    sSQL := 'Insert into DroneFlights '
          + '([DroneID], [Date and Time of Response]) '
          + 'Values(' + arrAvailableDrones[iRandom] + ', Format("' + Date +
            ' ' + Time + '", "General Date"))';
    objDB.DoSQL(sSQL);

    sSQL := 'Select [DroneFlightID] '
          + 'from DroneFlights '
          + 'where [DroneID] = ' + arrAvailableDrones[iRandom] +
            ' AND [Status] = "Scheduled" AND [DroneFlightID] NOT ' +
            'IN (Select [DroneFlightID] from FlightToServiceAssignment)';
    sDroneFlightID := objDB.ToString(sSQL, '');

    AssignUserServiceToFlight(UserServiceID, sDroneFlightID);
    AssignStaffToService(sDroneFlightID);
  end;
end;

end.
```

## Database Source Unit

```
unit uDB_Source;

interface
  uses
    DB, AdoDB, Forms, SysUtils, Dialogs, Controls;

  Type
    TDynamicArray = array of string;
    TDBClass = class
      private
        fDbTbl : TADOQuery;

      public
        constructor Create(NameOfDB : string);
        procedure DoSQL(TheSql : string);
        function RecordExist (TheSQL : string) : boolean;
        function ToStringHeadings(TheSQL, Delimiter : string) : string;
        function ToString(TheSQL, Delimiter : string) : string;
        function ToArray(TheSQL : string) : TDynamicArray;
    end;

IMPLEMENTATION

{ TdbClass }
```

```
//-------------------------------------CREATES A DB OBJECT
constructor TDBClass.Create(NameOfDB: string);
begin
  fDbTbl := TADOQuery.Create(Application);
  fDbTbl.ConnectionString := 'Provider=Microsoft.Jet.OLEDB.4.0;'
                             //'Provider= Microsoft.ACE.OLEDB.12.0; '
                             + 'Data Source=' + NameOfDB
                             + ';Persist Security Info=False';
end;
```

```
//-------------------------------------RETURNS ALL DATA OF A SQL STATEMENT
//-------------------------------------IN A ARRAY
function TDBClass.ToArray(TheSQL: string): TDynamicArray;
var
  K, iCount : byte;
begin
  DoSQL(TheSQL);
  fDbTbl.Open;
  fDbTbl.First;
  SetLength(Result, 0);
  iCount := 0;
  while NOT fDbTbl.Eof do
  begin
    for K := 0 to fDbTbl.FieldCount - 1 do
    begin
      iCount := iCount + 1;
      SetLength(Result, iCount);
      Result[iCount - 1] := fDbTbl.Fields[K].AsString;
    end;
    fDbTbl.Next;
  end;
  fDbTbl.Close;
end;
```

```
//-------------------------------------RETURNS ALL DATA OF A SQL STATEMENT
//-------------------------------------IN STRING FORMAT
function TDBClass.ToString(TheSQL, Delimiter : string)
: string;
var
  K : byte;
begin
  DoSQL(TheSQL);
  fDbTbl.Open;
  fDbTbl.First;
  Result := '';
  while NOT fDbTbl.Eof do
  begin
    for K := 0 to fDbTbl.FieldCount - 1 do
      Result := Result + fDbTbl.Fields[K].AsString + Delimiter;
    Delete(Result, Length(Result) - Length(Delimiter) + 1, Length(Delimiter));
    Result := Result + #13;
    fDbTbl.Next;
  end;
  Delete(Result, Length(Result), 1);
  fDbTbl.Close;
end;
```

```
//-------------------------------------PERFORMS THE SQL STATEMENT
procedure TDBClass.DoSQL(TheSQL : string);
begin
  //ShowMessage(TheSQL);
  fDbTbl.SQL.Text := TheSQL;
  fDbTbl.ExecSQL;
end;
```

```
//-------------------------------------RETURNS ALL THE FIELD HEADINGS OF
//-------------------------------------A SQL STATEMENT IN STRING FORMAT
function TDBClass.ToStringHeadings(TheSQL, Delimiter : string): string;
var
  K : byte;
begin
  DoSQL(TheSQL);
  fDbTbl.Open;
  fDbTbl.First;
  Result := '';
  for K := 0 to fDbTbl.FieldCount - 1 do
      Result := Result + fDbTbl.Fields[K].FieldName + Delimiter;
end;
```

```
//-------------------------------------CHECKS IF A RECORD EXISTS IN
//-------------------------------------THE DATABASE
function TDBClass.RecordExist(TheSQL: string): boolean;
begin
  DoSQL(TheSQL);
  fDbTbl.Open;
  fDbTbl.First;
  Result := NOT fDbTbl.Eof;
end;
```

```
end.
```

## Dashboard Unit

```
unit uDashboard;
```

```
interface
  uses
    uDB_Source, SysUtils, uLibrary;

  procedure DroneStatus(var TotalDrones, ActiveDrones : word; var Demand
  : string);


implementation
```

```
//-------------------------------------RETURNS DATA ABOUT IN FLIGHT DRONES
procedure DroneStatus(var TotalDrones, ActiveDrones : word; var Demand
: string);
var
  sSQL : string;
  rPercActive : real;
begin
  sSQL := 'Select Count(*) '
        + 'from Drones';
  TotalDrones := StrToInt(objDB.ToArray(sSQL)[0]);

  sSQL := 'Select Count(*) '
        + 'from Drones '
        + 'where [DroneID] IN (Select [DroneID] from DroneFlights where' +
          ' [Status] <> "Completed")';
  ActiveDrones := StrToInt(objDB.ToArray(sSQL)[0]);

  rPercActive := ActiveDrones/TotalDrones * 100;
  if rPercActive <= 40 then
    Demand := 'LOW DEMAND'
  else if rPercActive >= 80 then
    Demand := 'HIGH DEMAND'
  else
    Demand := 'AVG DEMAND';
end;
```

```
end.
```

## Client Reports and Logs Unit

```
unit uClientReportsAndLogs;

interface
  uses
    uLibrary, uDB_Source, SysUtils, DateUtils, frmFootage, Forms;

  const
     //-------------------------------------ARRAY OF STATUS SQL
    STATUS : array[0..3] of string =
    (
        '',
        ' AND [DroneFlights.Status] = "Scheduled"',
        ' AND [DroneFlights.Status] = "Departed"',
        ' AND [DroneFlights.Status] = "Completed"'
    );

     //-------------------------------------ARRAY OF ORDER BY SQL
    ORDER_BY : array[0..1] of string =
    (
        ' order by [Service Name], [Date and Time of Response]',
        ' order by [Date and Time of Response]'
    );

  function UserFlightReports(UserID : string; OrderByIndex, StatusIndex
  : byte) : string;
  function SurveillanceSummary(DroneFlightID : string) : string;
  procedure ShowSurvFootage(DroneFlightID : string);

  function CostOfServicesReport(UserID : string) : string;
  function SuburbReport(UserID : string) : string;

implementation
```

```
//-------------------------------------RETURNS A DETAILED RECIEPT FOR ALL
//-------------------------------------DRONE REQUEST SERVICES FOR THE
//-------------------------------------CURRENT MONTH
function CostOfServicesReport(UserID: string): string;
var
  sSQL : string;
begin
  Result := 'Cost of Service Report for ' +
            LongMonthNames[MonthOf(Date())] + ':' + #13 + #13;

  //-------------------------------------GETS DETAILS OF EACH SERVICE REQUEST
  sSQL := 'Select [Service Name], [Date and Time of Response], ' +
          'Format([Price Per Day], "Currency") AS [Price] '
        + 'from DroneFlights, Drones, Services, FlightToServiceAssignment' +
          ', UserServices '
        + 'where [DroneFlights.DroneID] = [Drones.DroneID] AND ' +
          '[Drone Type] = [Services.ServiceID] AND ' +
          '[DroneFlights.DroneFlightID] = ' +
          '[FlightToServiceAssignment.DroneFlightID] AND ' +
```

```
          '[FlightToServiceAssignment.UserServiceID] = ' +
          '[UserServices.UserServiceID] AND [UserServices.UserID] = ' +
           UserID + ' AND Month([Date and Time of Response]) = ' +
           'Month(Date())';
  Result := Result + objDB.ToStringHeadings(sSQL, #9) + #13;
  Result := Result + SECTION_BORDER + #13;
  Result := Result + objDB.ToString(sSQL, #9) + #13;
  Result := Result + SECTION_BORDER + #13;

  //-------------------------------------GETS THE TOTAL AMOUNT OWING
  sSQL := 'Select Format(Sum([Price Per Day]), "Currency") AS [Total] '
        + 'from DroneFlights, Drones, Services, FlightToServiceAssignment' +
          ', UserServices '
        + 'where [DroneFlights.DroneID] = [Drones.DroneID] AND ' +
          '[Drone Type] = [Services.ServiceID] AND ' +
          '[DroneFlights.DroneFlightID] = ' +
          '[FlightToServiceAssignment.DroneFlightID] AND ' +
          '[FlightToServiceAssignment.UserServiceID] = ' +
          '[UserServices.UserServiceID] AND [UserServices.UserID] = ' +
           UserID + ' AND Month([Date and Time of Response]) = ' +
           'Month(Date())';
  Result := Result + objDB.ToStringHeadings(sSQL, '') + #9 + #9 +
            objDB.ToString(sSQL, '');
end;
```

```
//-------------------------------------RETURNS A LIST OF ALL SERVICES
//-------------------------------------REQUESTED IN THE USERS SUBURB
function SuburbReport(UserID: string): string;
var
  sSQL, sSuburbID : string;
begin
  Result := 'Suburb Report for ' + LongMonthNames[MonthOf(Date())] +
            ':' + #13 + #13;

  sSQL := 'Select [Suburb] '
        + 'from Users '
        + 'where [UserID] = ' + UserID;
  sSuburbID := objDB.ToString(sSQL, '');

  sSQL := 'Select [Service Name], Count(*) AS [Number of Responses] '
        + 'from DroneFlights, Drones, Services '
        + 'where [DroneFlights.DroneID] = [Drones.DroneID] AND ' +
          '[Suburb] = ' + sSuburbID + ' AND [Drone Type] = ' +
          '[Services.ServiceID] AND Month(' +
          '[Date and Time of Response]) = Month(Date()) '
        + 'group by [Service Name]';
  Result := Result + objDB.ToStringHeadings(sSQL, #9) + #13;
  Result := Result + SECTION_BORDER + #13;
  Result := Result + objDB.ToString(sSQL, #9);
end;
```

```
//-------------------------------------RETRUNS A SURVEILLANCE SUMMARY OF A
//-------------------------------------SPECIFIC FLIGHT
function SurveillanceSummary(DroneFlightID: string)
: string;
var
  sSQL : string;
begin
  sSQL := 'Select [Surveillance Summary] '
        + 'from SurveillanceReports '
        + 'where [DroneFlightID] = ' + DroneFlightID;

  Result := objDB.ToString(sSQL, '');
end;
```

```
//-------------------------------------OPENS THE FOOTAGE FILE IN A
//-------------------------------------DEDICATED FORM
procedure ShowSurvFootage(DroneFlightID : string);
var
  objFootageForm : TFootageForm;
  sSQL, sFilePath : string;
begin
  sSQL := 'Select [Surveillance Footage] '
        + 'from SurveillanceReports '
        + 'where [DroneFlightID] = ' + DroneFlightID;
  sFilePath := objDB.ToString(sSQL, '');

  objFootageForm := TFootageForm.Create(Application);
  objFootageForm.ShowFootage(sFilePath);
  objFootageForm.Show;
end;
```

```
//-------------------------------------RETURNS SQL FOR ALL THE USERS
//-------------------------------------SERVICE REQUEST
function UserFlightReports(UserID : string; OrderByIndex, StatusIndex
  : byte) : string;
begin
  Result := 'Select [DroneFlights.DroneFlightID] AS [Flight No], ' +
            '[Service Name], [Serial Number] AS [Drone Serial Number]' +
            ', [Date and Time of Response], [DroneFlights.Status] ' +
            'AS [Status] '
          + 'from DroneFlights, Drones, Services, ' +
            'FlightToServiceAssignment, UserServices '
          + 'where [DroneFlights.DroneID] = [Drones.DroneID] AND ' +
            '[Drone Type] = [Services.ServiceID] AND ' +
            '[DroneFlights.DroneFlightID] = ' +
            '[FlightToServiceAssignment.DroneFlightID] AND ' +
            '[FlightToServiceAssignment.UserServiceID] = ' +
            '[UserServices.UserServiceID] AND [UserServices.UserID] = ' +
            UserID + STATUS[StatusIndex] + ORDER_BY[OrderByIndex];
end;
```

```
end.
```

## Auto Deployment Class

```
unit uAutoDroneDeployment;

interface
  uses
    uLibrary, uDB_Source;

  const
    arrFootage : array[1..2] of string =
    (
      'SurvFootage1.jpg', 'SurvFootage2.jpg', 'SurvFootage3.jpg',
      'SurvFootage4.jpg', 'SurvFootage5.jpg'
    );

  procedure DeployDrones;
  procedure CompleteFlights;

implementation

//-------------------------------------DEPLOYS ALL DRONES IF THEY HAVE A
//-------------------------------------SCHEDULED FLIGHT
procedure DeployDrones;
var
  sSQL : string;
  K, iLength : byte;
  arrDroneFlightIDs : TDynamicArray;
begin
  //-------------------------------------GETS ALL FLIGHTS WHICH ARE MEANT
  //-------------------------------------TO BE UNDERWAY
  sSQL := 'Select [DroneFlightID] '
        + 'from DroneFlights '
        + 'where [Date and Time of Response] <= Now() AND [Status] ' +
          '<> "Completed"';
  arrDroneFlightIDs := objDB.ToArray(sSQL);

  //-------------------------------------DEPART EACH DRONE
  iLength := Length(arrDroneFlightIDs);
  if iLength <> 0 then
  begin
    sSQL := 'Update DroneFlights '
        + 'Set [Status] = "Departed" '
        + 'where [Date and Time of Response] <= Now() AND [Status] ' +
          '<> "Completed"';
    objDB.DoSQL(sSQL);

    Randomize;
    for K := 0 to Length(arrDroneFlightIDs) - 1 do
    begin
      sSQL := 'Update SurveillanceReports '
            + 'Set [Surveillance Summary] = "Retrieving ' +
              'Surveillance Footage..." '
            + 'where [DroneFlightID] = ' + arrDroneFlightIDs[K];
      objDB.DoSQL(sSQL);
```

```
    end;
  end;
end;
```

```
//-------------------------------------CHANGE FLIGHT STATUS TO COMPLETE IF
//-------------------------------------SURVEILLANCE IS OVER
procedure CompleteFlights;
var
  sSQL : string;
  K, iRandom, iLength : byte;
  arrDroneFlightIDs : TDynamicArray;
begin
  //-------------------------------------GETS ALL FLIGHTS WHICH ARE MEANT
  //-------------------------------------TO HAVE BEEN COMPLETED
  sSQL := 'Select [DroneFlightID] '
        + 'from DroneFlights, Drones, Services '
        + 'where [DroneFlights.DroneID] = [Drones.DroneID] AND ' +
          '[Drones.Drone Type] = [ServiceID] AND ' +
          'IIF([Service Name] = "Emergency Response", DateAdd("h", 1, ' +
          '[Date and Time of Response]) <= Now(), Format(Format(' +
          '[Date and Time of Response], "Short Date") & " " & ' +
          'Format([Surveillance End Time], "Long Time"), ' +
          '"General Date") <= Now()) AND [DroneFlights.Status] ' +
          '<> "Completed"';
  arrDroneFlightIDs := objDB.ToArray(sSQL);

  //-------------------------------------COMPLETES EACH FLIGHT
  iLength := Length(arrDroneFlightIDs);
  if iLength <> 0 then
  begin
    sSQL := 'Update DroneFlights '
          + 'Set [Status] = "Completed" '
          + 'where [DroneFlightID] IN (Select [DroneFlightID]' +
            ' from DroneFlights, Drones, Services where ' +
            '[DroneFlights.DroneID] = [Drones.DroneID] AND ' +
            '[Drones.Drone Type] = [ServiceID] AND ' +
            'IIF([Service Name] = "Emergency Response", DateAdd("h", 1, ' +
            '[Date and Time of Response]) <= Now(), Format(Format(' +
            '[Date and Time of Response], "Short Date") & " " & ' +
            'Format([Surveillance End Time], "Long Time"), ' +
            '"General Date") <= Now()) AND [DroneFlights.Status] ' +
            '<> "Completed")';
    objDB.DoSQL(sSQL);

    //-------------------------------------ASSIGNS FOOTAGE TO A FLIGHT
    Randomize;
    for K := 0 to iLength - 1 do
    begin
      iRandom := Random(Length(arrFootage)) + 1;
      sSQL := 'Update SurveillanceReports '
            + 'Set [Surveillance Footage] = "' + arrFootage[iRandom] + '" '
            + 'where [DroneFlightID] = ' + arrDroneFlightIDs[K];
      objDB.DoSQL(sSQL);

      sSQL := 'Update SurveillanceReports '
```

```
            + 'Set [Surveillance Summary] = "Waiting for ' +
              'Incident Report..." '
            + 'where [DroneFlightID] = ' + arrDroneFlightIDs[K];
      objDB.DoSQL(sSQL);
    end;
  end;
end;
```

```
end.
```

## Analyse Surveillance Unit

```
unit uAnalyseSurveillance;

interface
  uses
    uLibrary, Dialogs;

  function SurveillanceToBeAnalysed(StaffID : string) : string;
  function GetSurveillanceFootage(FlightID : string) : string;
  procedure SubmitSurveillanceAnalyses(FlightID, Summary : string);

implementation
```

```
//-------------------------------------RETURNS SQL OF ALL SURVEILLANCE WHICH
//-------------------------------------NEEDS TO BE ANALYSED BY A SPECIFC
//-------------------------------------STAFF MEMBER
function SurveillanceToBeAnalysed(StaffID : string) : string;
begin
  Result := 'Select [DroneFlights.DroneFlightID] AS [Flight No],' +
            ' [Service Name], [Serial Number] AS [Drone Serial Number]' +
            ', [Date and Time of Response] '
          + 'from DroneFlights, Drones, Services, SurveillanceReports '
          + 'where [DroneFlights.DroneID] = [Drones.DroneID] AND ' +
            '[Drone Type] = [Services.ServiceID] AND ' +
            '[DroneFlights.DroneFlightID] = ' +
            '[SurveillanceReports.DroneFlightID] AND ' +
            '[StaffID] = ' + StaffID + 'AND [Time Analysed] IS NULL AND ' +
            '[DroneFlights.Status] = "Completed"';
end;
```

```
//-------------------------------------RETURNS THE SURVEILLANCE FOOTAGE OF
//-------------------------------------A SPECIFIC FLIGHT
function GetSurveillanceFootage(FlightID : string) : string;
var
  sSQL : string;
begin
  sSQL := 'Select [Surveillance Footage] '
        + 'from SurveillanceReports '
        + 'where [DroneFlightID] = ' + FlightID;
  Result := objDB.ToString(sSQL, '');
end;
```

```
//-------------------------------------SUBMITS THE STAFF MEMBERS SURVEILLANCE
//-------------------------------------REPORT
procedure SubmitSurveillanceAnalyses(FlightID, Summary : string);
var
  sSQL : string;
begin
  sSQL := 'Update SurveillanceReports '
        + 'set [Surveillance Summary] = "' + Summary + '"  '
        + 'where [DroneFlightID] = ' + FlightID;
  objDB.DoSQL(sSQL);

  sSQL := 'Update SurveillanceReports '
        + 'set [Time Analysed] = Now() '
        + 'where [DroneFlightID] = ' + FlightID;
  objDB.DoSQL(sSQL);

  MessageDlg('Surveillance Analysis Submitted Successfully', mtInformation,
  mbOKCancel, 0);
end;
```

```
end.
```

## Account Details Unit

```
unit uAccountDetails;

interface

uses
  uLibrary, Dialogs, Forms, SysUtils;

Type
  TAccountDetails = class
  private
    fLoginForm: TForm;
    fIsClient: boolean;

  public
    constructor Create(LoginForm: TForm; IsClient: boolean);
    procedure ChangePassword(OldPass, NewPass, RetypedNewPass, PersonsPass,
    ID, Name, Email: string; var Completed : boolean);
    procedure DeactivateAcc(ID: string; Form: TForm);
    procedure LogOut(Form: TForm);
  end;

implementation
```

```
//-------------------------------------CHANGES A USERS PASSWORD
procedure TAccountDetails.ChangePassword(OldPass, NewPass, RetypedNewPass,
  PersonsPass, ID, Name, Email: string; var Completed : boolean);
var
  sSQL: string;
begin
  Completed := false;

  //-------------------------------------CHECKS THAT ALL FIELDS HAVE
  //-------------------------------------BEEN ENTERED
  if (OldPass = '') OR (NewPass = '') OR (RetypedNewPass = '') then
    MessageDlg('None of the field for changing your password may be empty',
      mtError, mbOKCancel, 0)
  else if OldPass = PersonsPass then
  begin
    if PasswordCriteriaMet(NewPass) then
    begin
      if NewPass = RetypedNewPass then
      begin
        Completed := true;

        if fIsClient = true then
          sSQL := 'Update Users '
                + 'Set [Password] = "' + NewPass + '" '
                + 'where [UserID] = ' + ID
        else
          sSQL := 'Update Staff '
                + 'Set [Password] = "' + NewPass + '" '
                + 'where [StaffID] = ' + ID;
```

```
        objDB.DoSQL(sSQL);

        MessageDlg('Email Sending Please Wait...', mtInformation, mbOKCancel,
        0);

        //-------------------------------------SENDS EMAIL CONFIRMING
        //-------------------------------------PASSWORD CHANGE
        objEmails.SendEmail(Name, Email, 'Your Password Has Been Changed',
        'Dear ' + Name + ',' + #13 + #13 + 'Your password has ' +
        'been changed to "' + NewPass + '", this change happened at ' +
        TimeToStr(Now()) + ' on the ' + DateToStr(Date()) + #13 +
        'If this was NOT you please log in and ensure that you change' +
        ' your password back and secure your account!' + #13 + #13 +
        'Regards,' + #13 + 'Birds Eye Security');

        MessageDlg('You password has been changed', mtInformation, mbOKCancel,
          0);
      end
      else
        MessageDlg('The new password that you entered if different to' +
            ' the retyped new password. Please ensure you entered your new ' +
            'desired password correctly', mtError, mbOKCancel, 0);
    end;
  end
  else
    MessageDlg('The current password that you entered if different to' +
    ' the password that is connected to this account. Please ensure you ' +
    'entered the password correctly', mtError, mbOKCancel, 0);
end;
```

```
//-------------------------------------CREATES ACCOUNT DETAILS OBJECT
constructor TAccountDetails.Create(LoginForm: TForm; IsClient: boolean);
begin
  fLoginForm := LoginForm;
  fIsClient := IsClient;
end;
```

```
//-------------------------------------DEACTIVATES A USERS ACCOUNT
procedure TAccountDetails.DeactivateAcc(ID: string; Form: TForm);
var
  sSQL: string;
begin
  if fIsClient = true then
    sSQL := 'Update Users ' + 'Set [Inactive User] = True ' +
      'where [UserID] = ' + ID
  else
    sSQL := 'Update Staff ' + 'Set [Inactive Staff] = True ' +
      'where [StaffID] = ' + ID;

  objDB.DoSQL(sSQL);

  MessageDlg('Your account has been successfully deactivated!', mtConfirmation,
    mbOKCancel, 0);

  LogOut(Form);
end;

//-------------------------------------LOGS A USER OUT OF THEIR ACCOUNT
procedure TAccountDetails.LogOut(Form: TForm);
begin
  MessageDlg('You have been Logged Out', mtInformation, mbOKCancel, 0);
  Pointer((@Application.MainForm)^) := fLoginForm;
  Form.Destroy;
  fLoginForm.Show;
end;
```

```
end.
```

# Application Setup File

```
#define MyAppName "Birds Eye Security Surveillance"
#define MyAppVersion "1.0.5"
#define MyAppPublisher "Birds Eye Security"
#define MyAppURL "https://birdseyesecuity.wixsite.com/application"
#define MyAppExeName "BirdsEyeSecuritySurveillance.exe"
```

```
[Setup]
; NOTE: The value of AppId uniquely identifies this application. Do not use
the same AppId value in installers for other applications.
; (To generate a new GUID, click Tools | Generate GUID inside the IDE.)
AppId={{437AC36C-F341-4507-83ED-849856088A9D}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
;AppVerName={#MyAppName} {#MyAppVersion}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={autopf}\{#MyAppName}
DisableProgramGroupPage=yes
; Uncomment the following line to run in non administrative install mode
(install for current user only.)
;PrivilegesRequired=lowest
OutputDir=F:\School\Information Technology\IEB PAT
OutputBaseFilename=BirdsEyeSecuritySetup
SetupIconFile=C:\Users\Ryan\Downloads\icon.ico
Compression=lzma
SolidCompression=yes
WizardStyle=modern


[Languages]
```

```
Name: "english"; MessagesFile: "compiler:Default.isl"


[Tasks]

Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}";
GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked


[Files]

Source: "F:\School\Information Technology\IEB
PAT\BirdsEyeSecuritySurveillance.exe"; DestDir: "{app}"; Flags:
ignoreversion

Source: "F:\School\Information Technology\IEB
PAT\BirdsEyeSecuritySurveillance_Data.mdb"; DestDir: "{app}"; Flags:
ignoreversion

Source: "F:\School\Information Technology\IEB PAT\HelpURLs.txt"; DestDir:
"{app}"; Flags: ignoreversion

Source: "F:\School\Information Technology\IEB PAT\libeay32.dll"; DestDir:
"{app}"; Flags: ignoreversion

Source: "F:\School\Information Technology\IEB PAT\PageAccess.txt"; DestDir:
"{app}"; Flags: ignoreversion

Source: "F:\School\Information Technology\IEB PAT\ServiceDetails.txt";
DestDir: "{app}"; Flags: ignoreversion

Source: "F:\School\Information Technology\IEB PAT\ssleay32.dll"; DestDir:
"{app}"; Flags: ignoreversion

Source: "F:\School\Information Technology\IEB PAT\SurvFootage1.jpg";
DestDir: "{app}"; Flags: ignoreversion

Source: "F:\School\Information Technology\IEB PAT\SurvFootage2.jpg";
DestDir: "{app}"; Flags: ignoreversion

Source: "F:\School\Information Technology\IEB PAT\SurvFootage3.jpg";
DestDir: "{app}"; Flags: ignoreversion

Source: "F:\School\Information Technology\IEB PAT\SurvFootage4.jpg";
DestDir: "{app}"; Flags: ignoreversion

Source: "F:\School\Information Technology\IEB PAT\SurvFootage5.jpg";
DestDir: "{app}"; Flags: ignoreversion

; NOTE: Don't use "Flags: ignoreversion" on any shared system files


[Icons]

Name: "{autoprograms}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"
```

```
Name: "{autodesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}";
Tasks: desktopicon


[Run]

Filename: "{app}\{#MyAppExeName}"; Description:
"{cm:LaunchProgram,{#StringChange(MyAppName, '&', '&&')}}"; Flags: nowait
postinstall skipifsilent
```


# BIRDS EYE SECURITY SURVEILLANCE

## Testing and Output

Ryan Trickett
Gr12 Reddam House Bedfordview
Exam Number: 201112020858

# TABLE OF CONTENTS

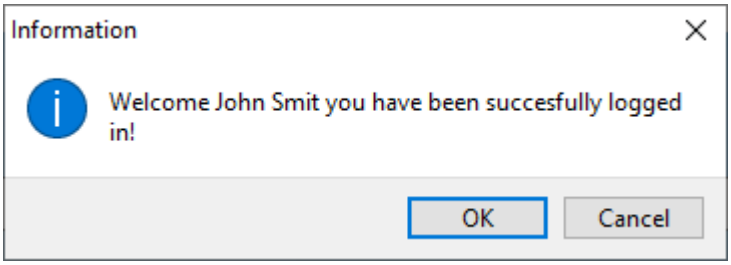
# TESTING

## Sign Up

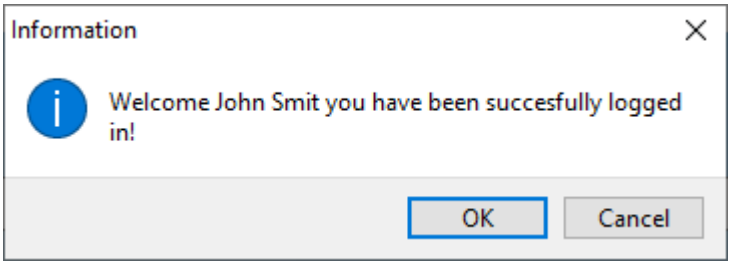
| Test Number | Input Field | Normal | Extreme | Erroneous |
|---|---|---|---|---|
| 1 | Email | john.smit@gmail.com | JohnnnnnnnnsmIThs12@yahoo.com | @SA453 |
| 2 | Password | Pass2029! | PasswordsAreTheBesThingsInTheWorld | #234:A |
| 3 | Recommended Postal Code | 2194 | PR920 | 😀 |

## Test 1

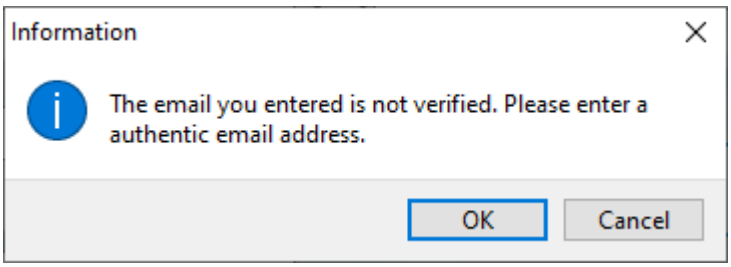
### Normal




### Extreme

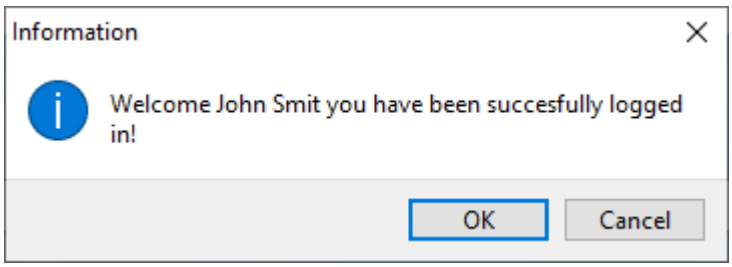



### Erroneous

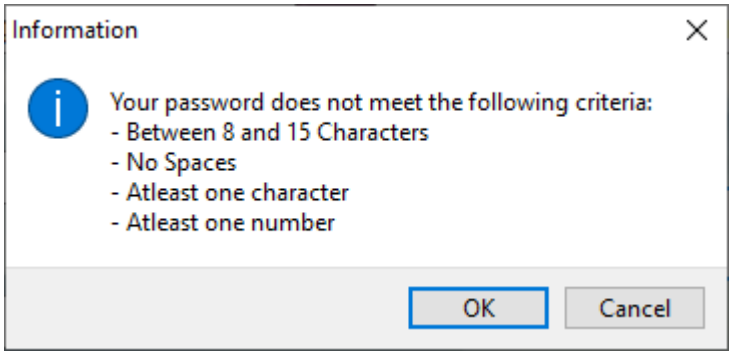


## Test 2

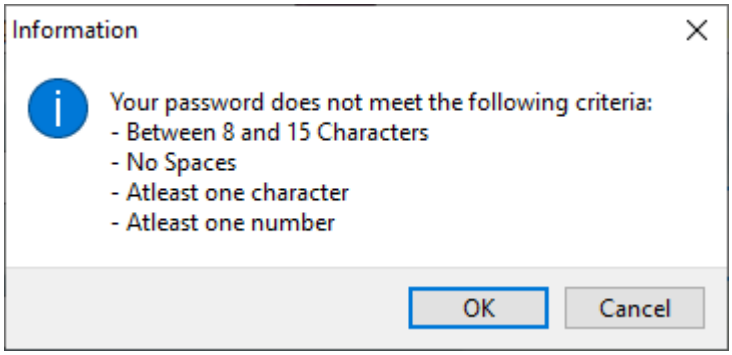
### Normal




### Extreme

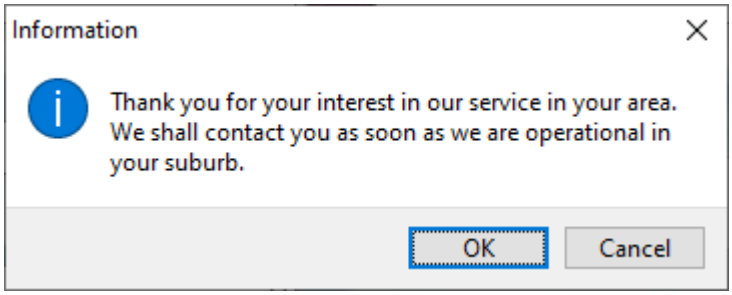



### Erroneous

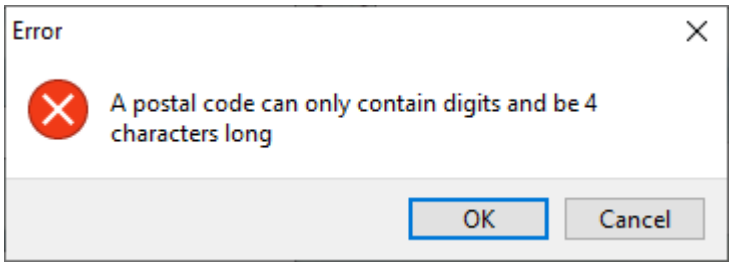


# Test 3

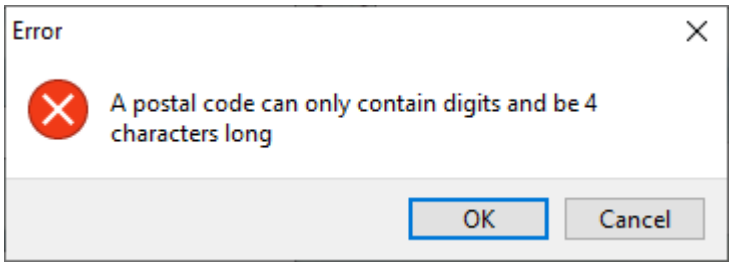
## Normal




## Extreme




## Erroneous

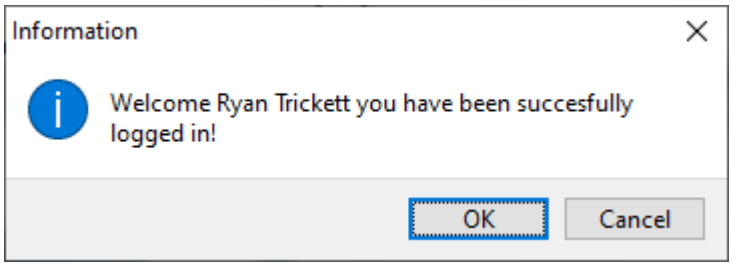


# Login

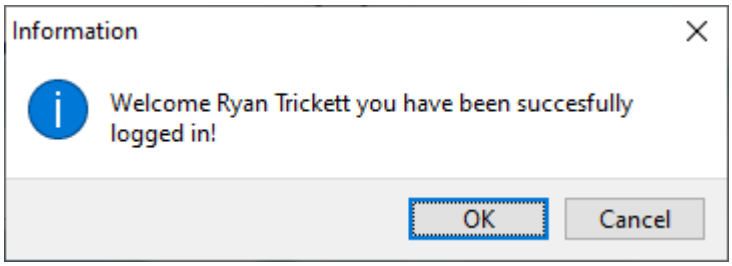
| Test Number | Input Field | Normal | Extreme | Erroneous |
|---|---|---|---|---|
| 1 | Email | ryanbasiltrickett@gmail.com | ryansasdwdasd @yahoo.com | @SA453 |
| 2 | Password | PassWor9 | PasswordsAreTheBesThingsInTheWorld | #234:A |
| 3 | Forgot Password Email | ryanbasiltrickett@gmail.com | ryansasdwdasd@yahoo.com | $400 |

## Test 1

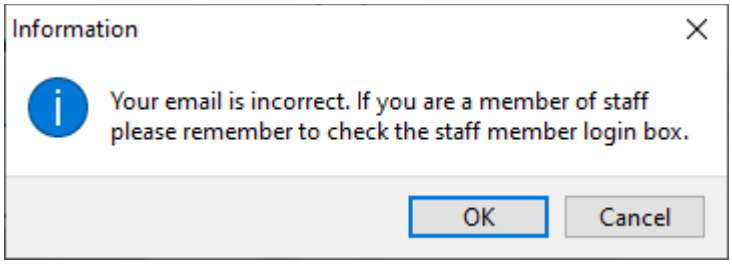
### Normal




### Extreme

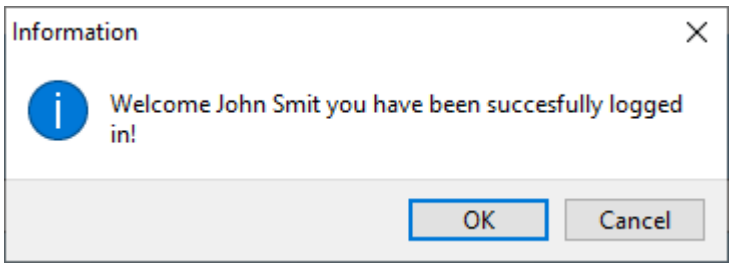



### Erroneous

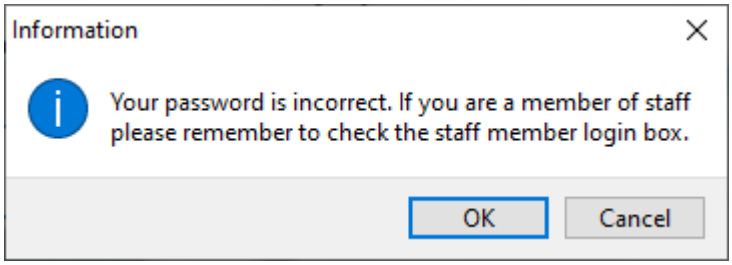


## Test 2

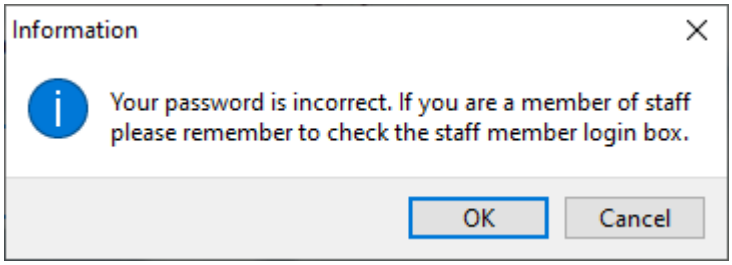
### Normal




### Extreme

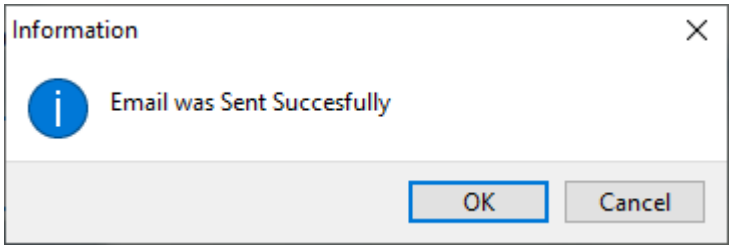



### Erroneous

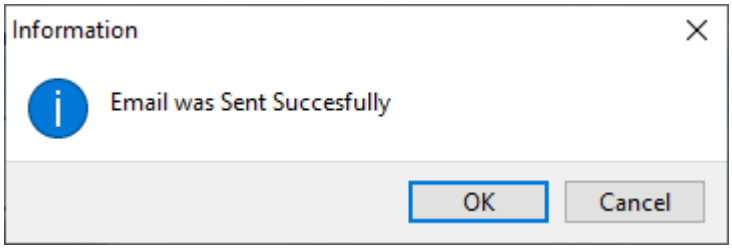


# Test 3

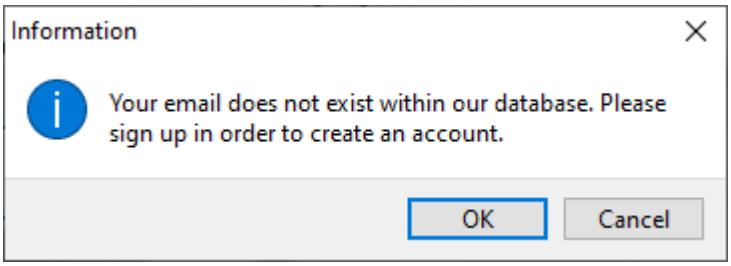
## Normal




## Extreme




## Erroneous

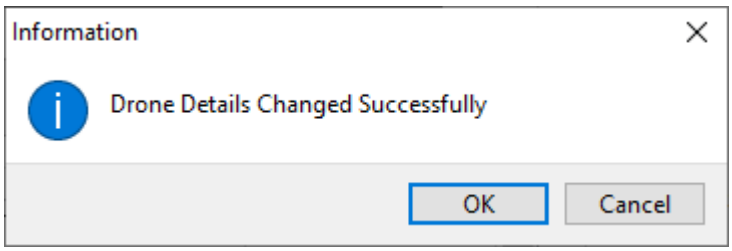


# Drone Management

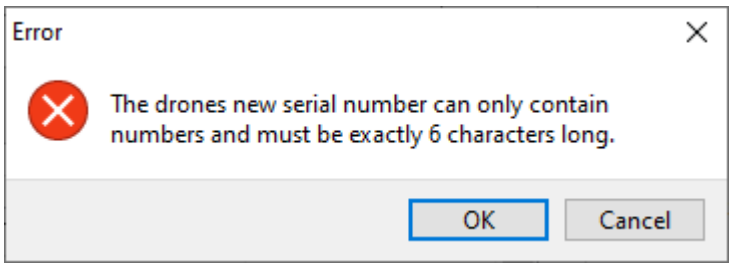
| Test Number | Input Field | Normal | Extreme | Erroneous |
|---|---|---|---|---|
| 1 | Serial Number | 123145 | RICH4RD | :20L2! |

## Test 1

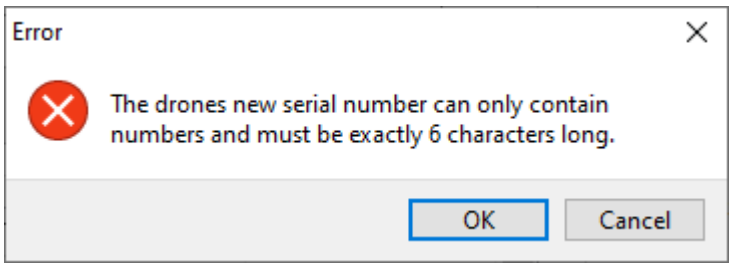
### Normal




### Extreme

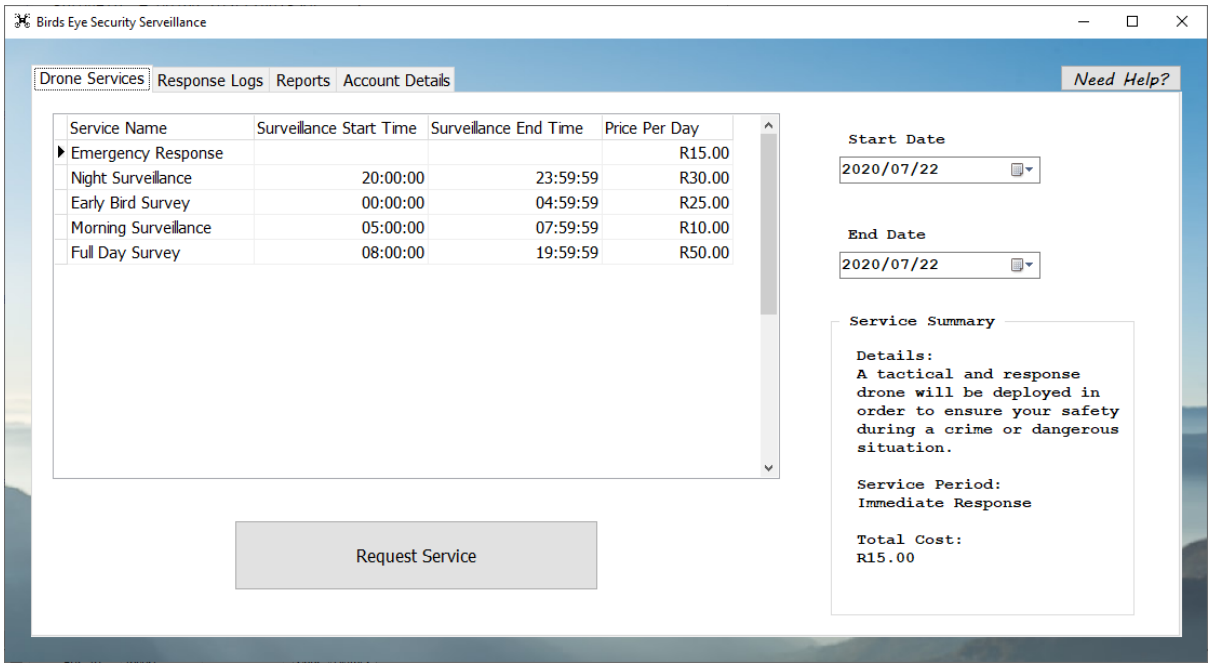



### Erroneous

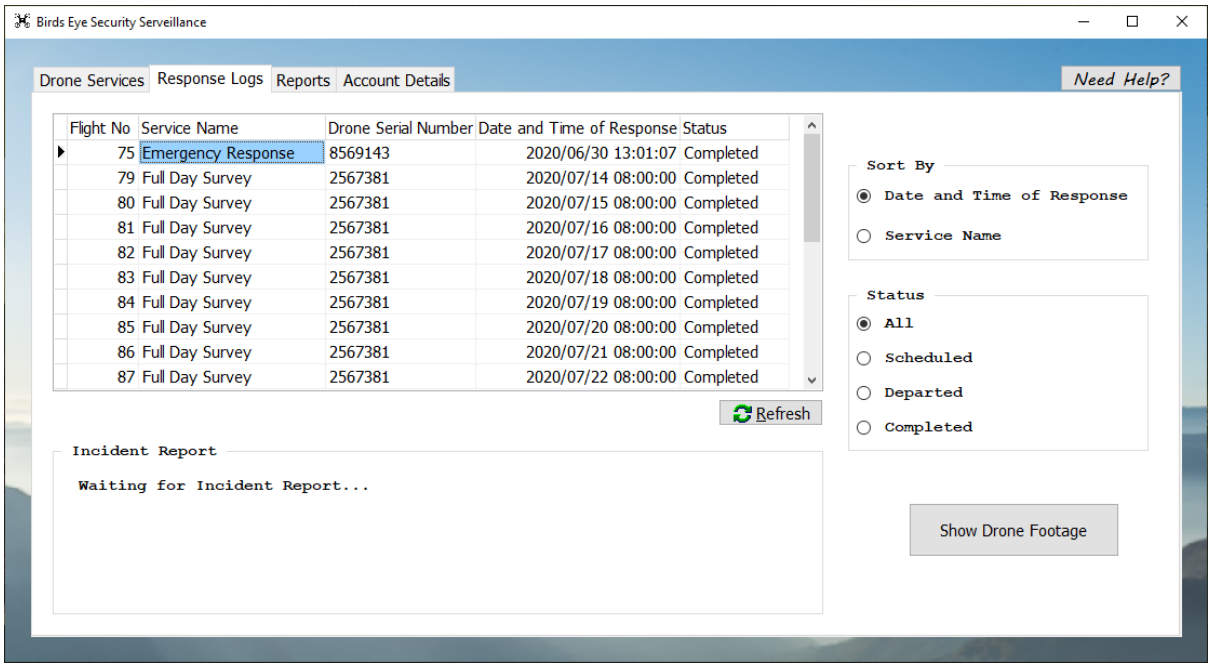


# APPLICATION OUTPUT

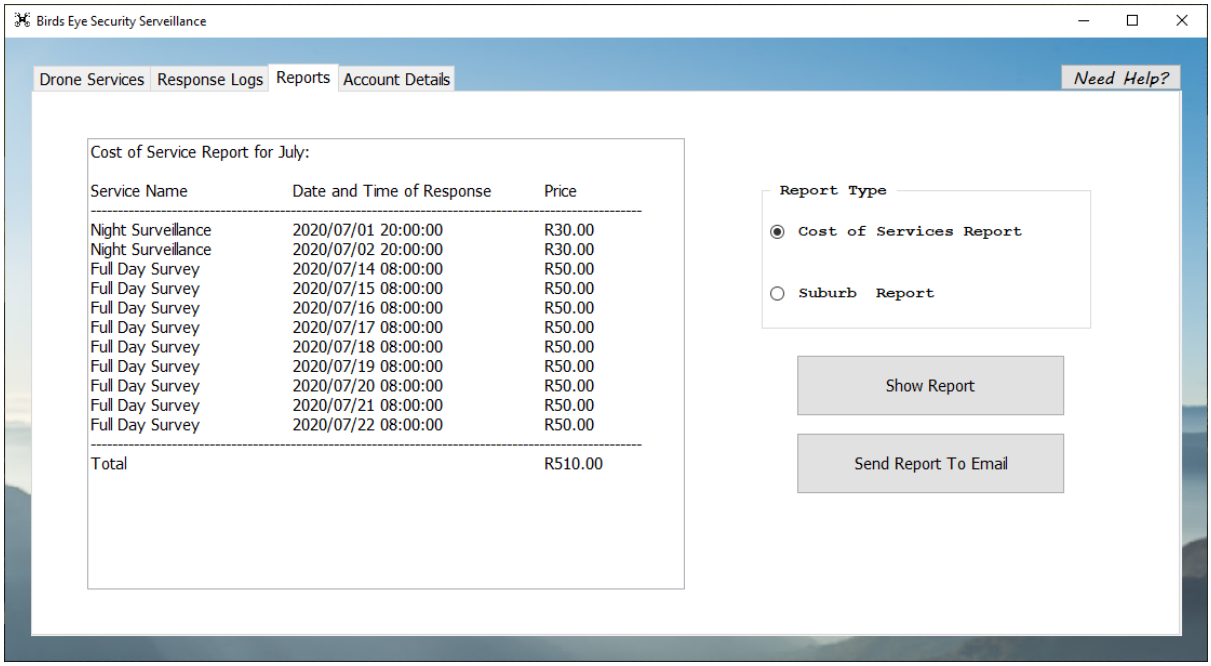
## Drone Service Request



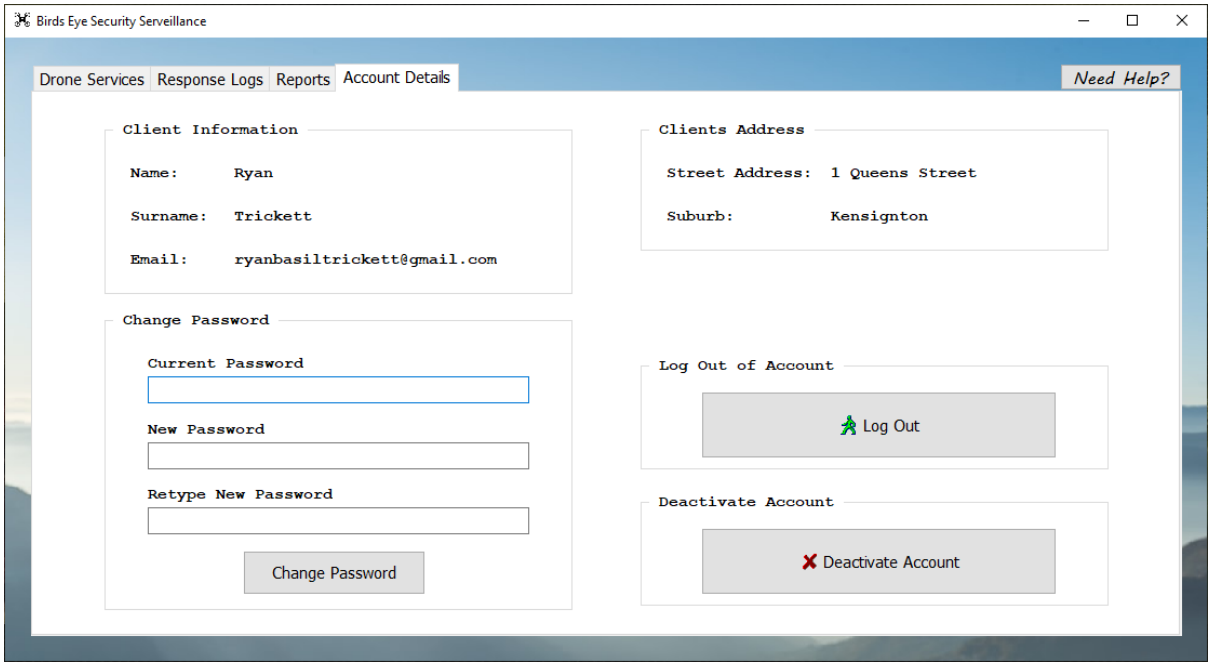

## Response Logs



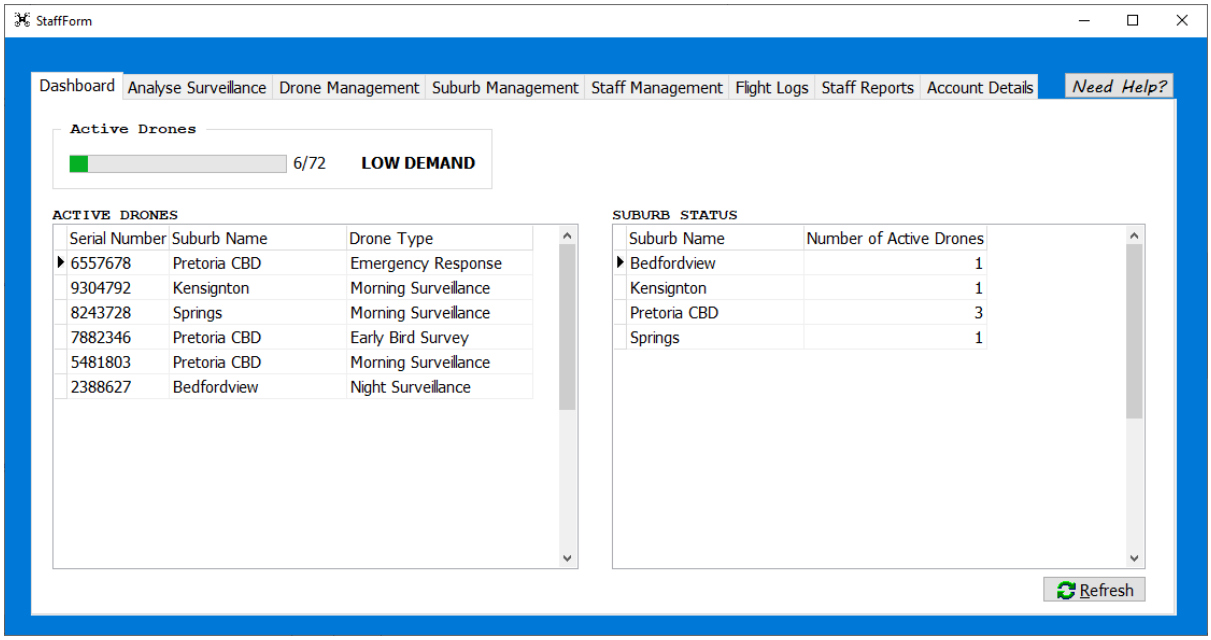
# Reports



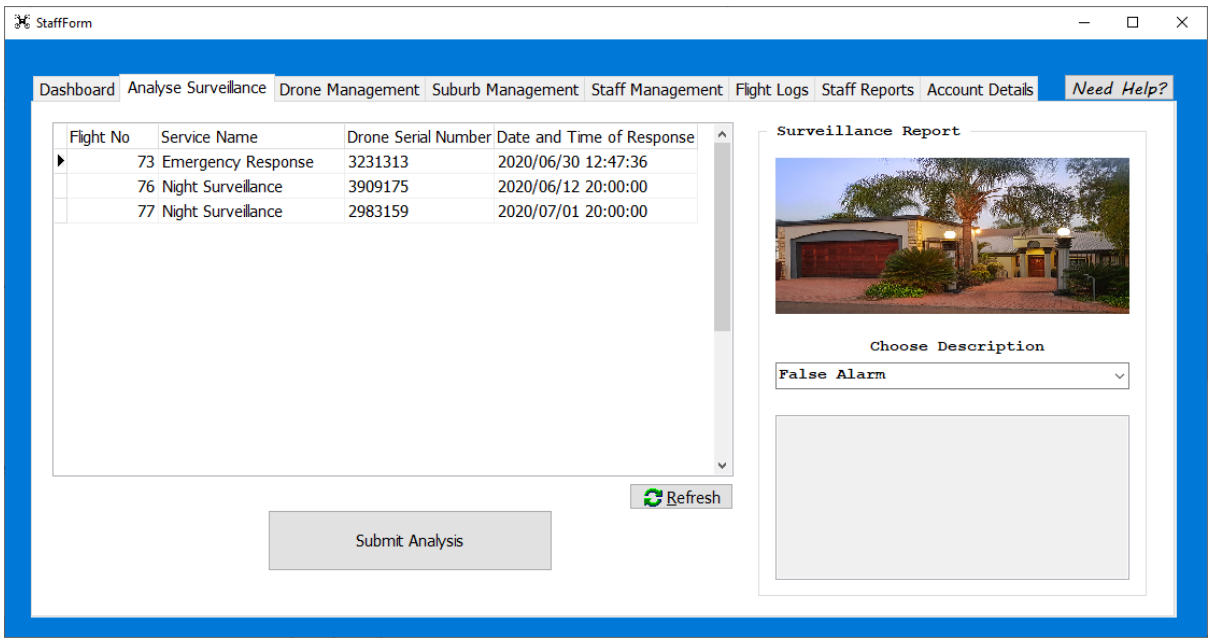

# Account Details



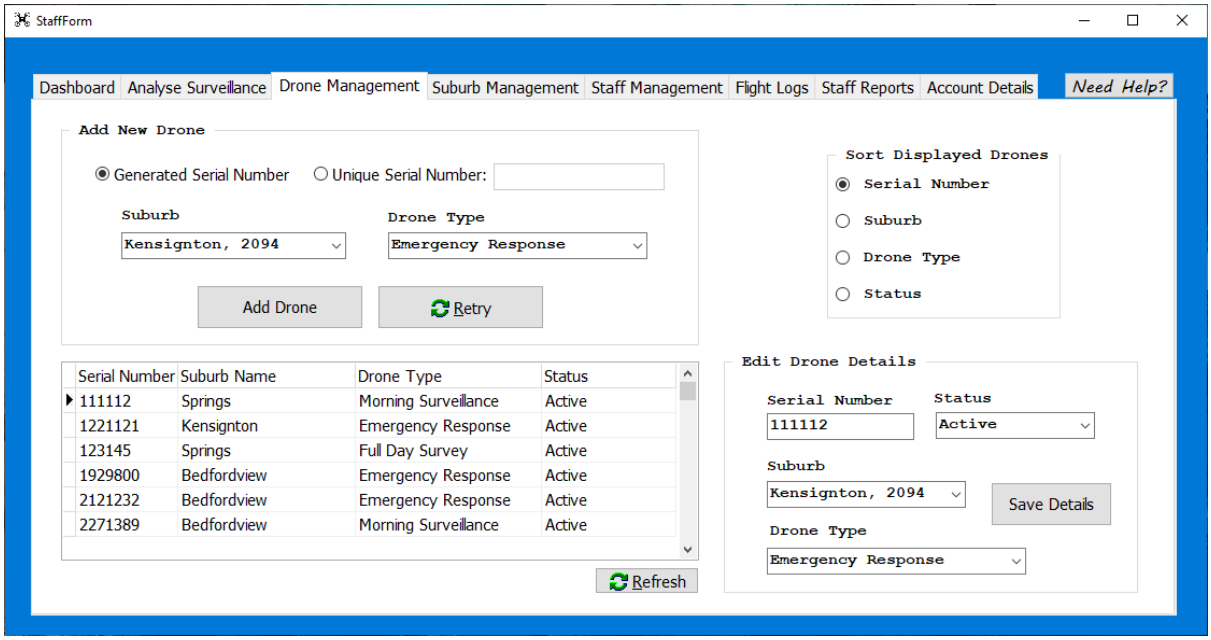
# Dashboard



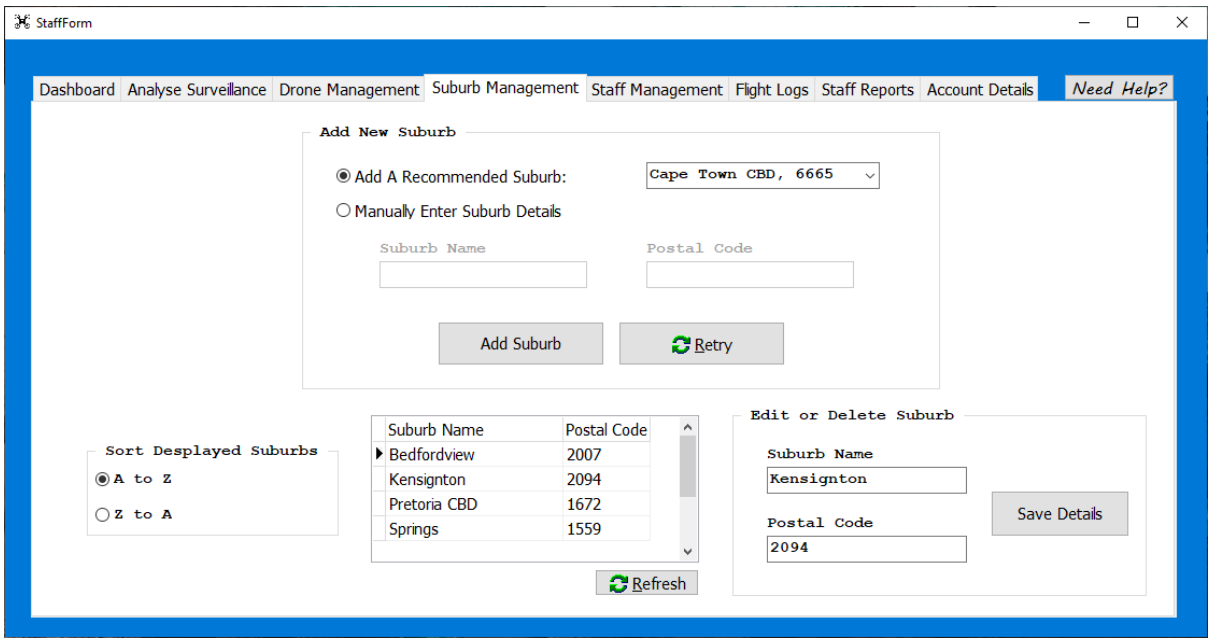

# Analyse Surveillance



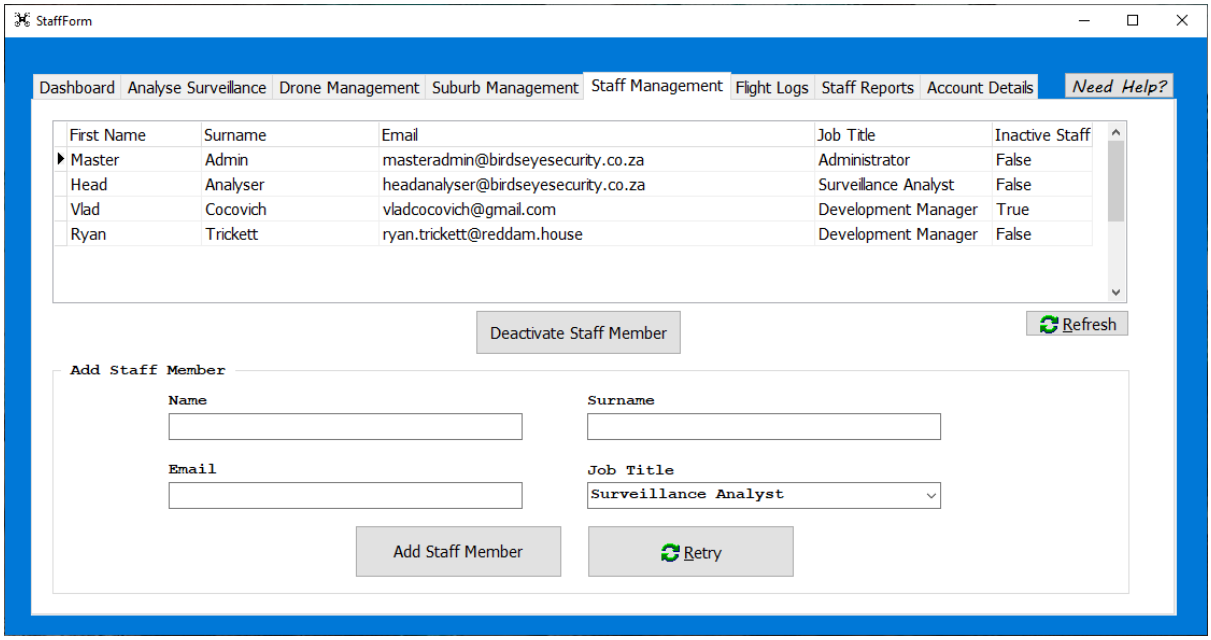
# Drone Management



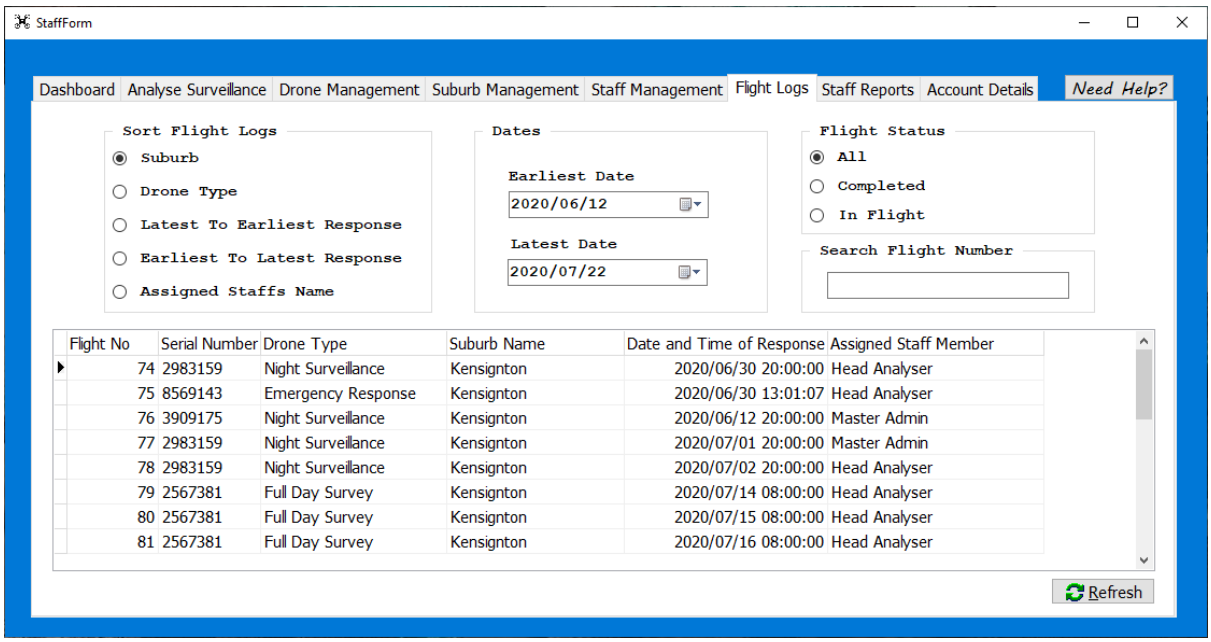

# Suburb Management



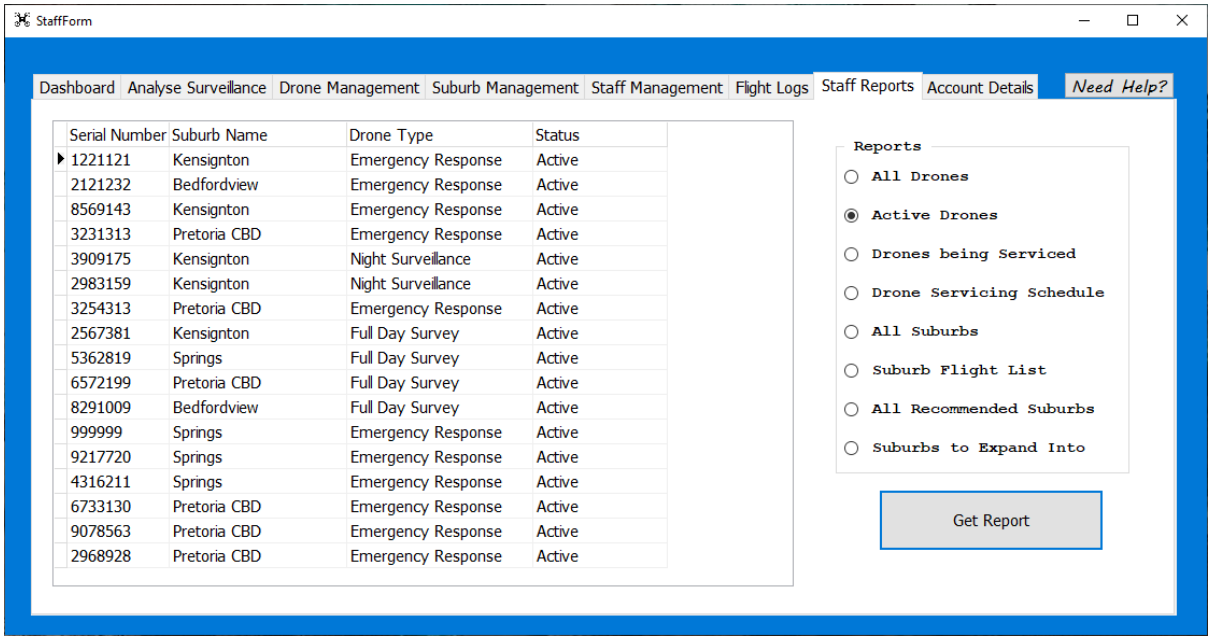
# Staff Management




# Flight Logs



# Staff Reports




# Account Details (Staff)