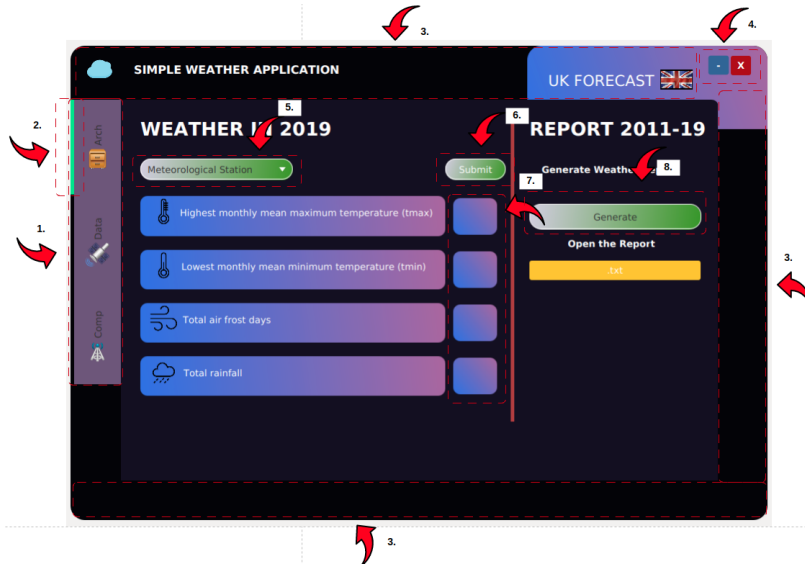


Program design, implementation and basic algorithmic description

Additional features are highlighted in yellow.

Tab: "Weather in 2019"

In order to navigate over the GUI. The tab-pane with 3 tabs has been placed on the left side: appendix 1. (Point 1). For element alignment purpose, every single element in the tab has been placed in the grid-pane. The selected tab highlights in green, to indicate user position in the application (Point 2).



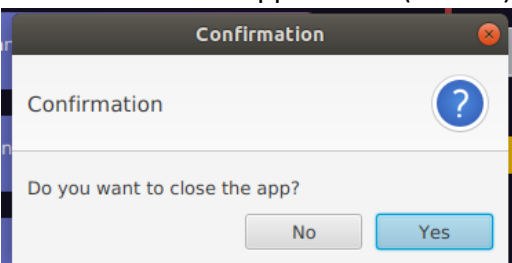
The simple weather application GUI has been designed with the fixed window dimensions. Browser frame has been made transparent, root set and drag on mouse areas indicated: appendix 1. (Point 3).

```
//grab root
root.setOnMousePressed(event -> {
    xOffset = event.getSceneX();
    yOffset = event.getSceneY();
});

//move around the application
root.setOnMouseDragged(event -> {
    primaryStage.setX(event.getScreenX() - xOffset);
    primaryStage.setY(event.getScreenY() - yOffset);
});
Scene scene = new Scene(root);
scene.setFill(Color.TRANSPARENT);
```

Appendix 1. Weather in 2019 tab.

Window buttons: appendix 1. (Point 4) exist a on the GUI for the entire operating time. The minimize button “-” shrinks the application window and places it on the taskbar while leaving the program running. By clicking the close button “x”, the confirmation dialog box becomes displayed and acts as safety function. User confirmation is required to close the application.



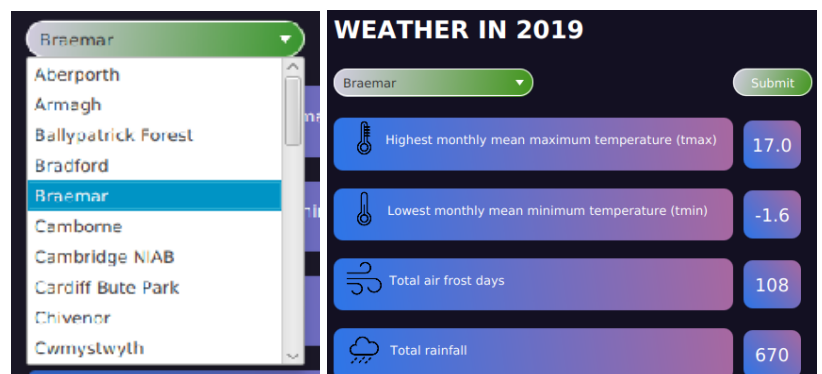
```
@FXML // private method to close the application
private void closeApp() {
    Alert alert = new Alert(Alert.AlertType.CONFIRMATION, "Do you want to close the app?",
        ButtonType.YES, ButtonType.NO);
    alert.showAndWait();
}
```

Basic requirement 1: ability to obtain statistics from the “last year (2019)” meteorological stations. The user selects meteorological station combo-box: appendix 1. (Point 5) the list of stations becomes displayed 1. Secondly, the user clicks “onAction” linked submit button appendix 1. (Point 6). “StantionsUtil” class, loads requested data into strings (tempMini, tempMaxi, airFrost, rainfall), method loops through the double arrays and displays selected data in linked GUI labels: appendix 1. (Point 7). For a purpose of clarity: the data of Braemar station displayed.

```
try {
    StationData stationData = StationsUtil.load(name, year);

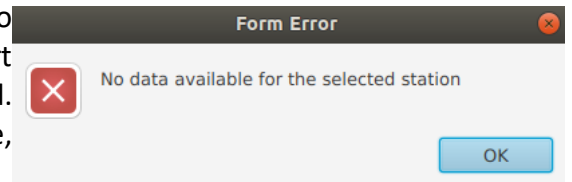
    List<String> inputtempMini = stationData.getInputtempMini();
    List<String> inputtempMaxi = stationData.getInputtempMaxi();
    List<String> inputairFrost = stationData.getInputairFrost();
    List<String> inputrainfall = stationData.getInputrainfall();

    double[] toNumberMini = new double[inputtempMini.size()];
    for (int i = 0; i < inputtempMini.size(); ++i) {
        toNumberMini[i] = Double.parseDouble(inputtempMini.get(i));
    }
}
```

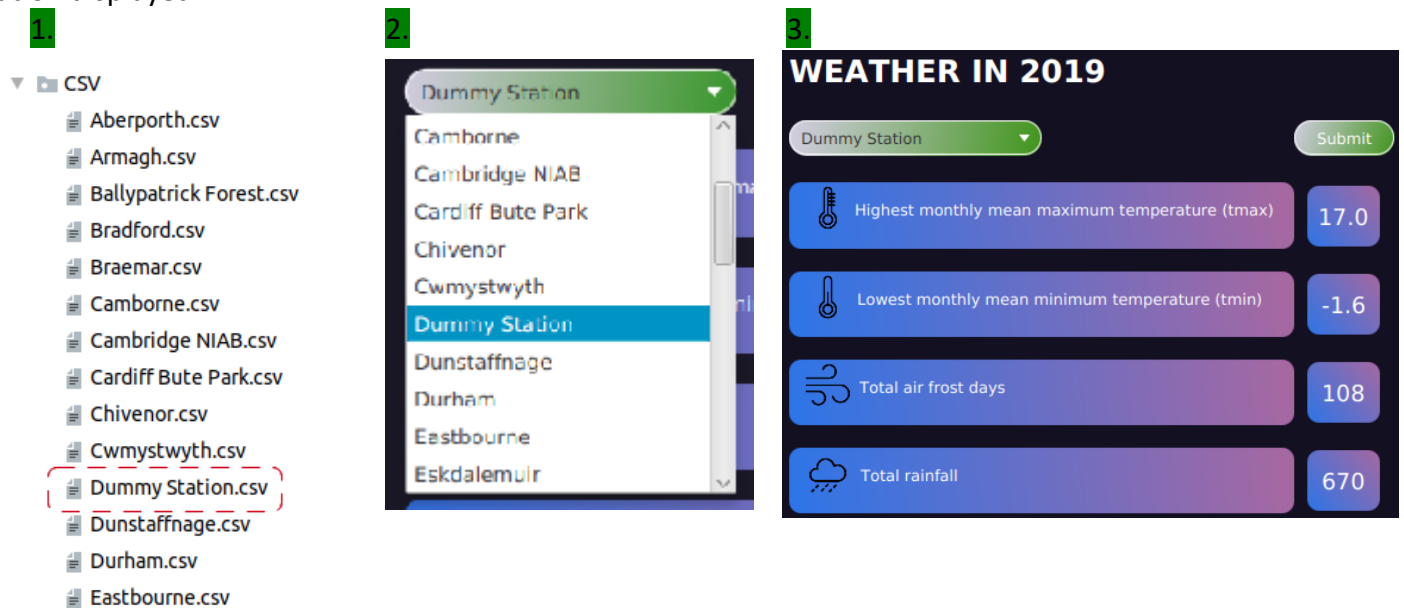


Additional feature 1: if a .CSV file does not contain data to display for selected period. Invokes the method and the alert dialog box (applies for the whole application) becomes displayed. Taken from AlertHelper class, e.g. if metStation, cannot getValue, for selected “year”, catch – “AlertHelper”.

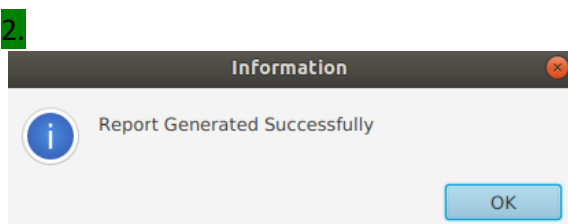
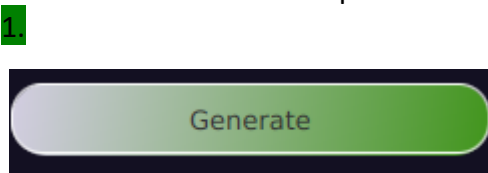
```
try {
    archiveChart(metStation.getValue(), year: "2019");
} catch (Exception e) {
    Window owner = metStation.getScene().getWindow();
    AlertHelper.showAlert(Alert.AlertType.ERROR, owner, title: "Form Error",
        message: "No data available for the selected station");
}
```



Additional feature 2: The application can dynamically update the analysed station’s data “Station Util class” and can add additional stations in the selection menu, because its functionality allows to drag and drop any .csv file that has been compiled in the same format as the other example .csv files used in this project. The below demonstrates this with file “Dummy Station.csv”. For a purpose of clarity: the data copied of Braemar station displayed.



Basic requirement 2: the user is able to generate a weather report, containing the key facts for all the meteorological stations in a single .txt file. Click the - Generate button: appendix 1. (Point 8). The information dialog box becomes displayed 2. Indicates: “Report Generated Successfully”. The generated .txt file contains all the requested information 3:



```
Number: <1>
Name: <Lerwick>
Highest: <7/2011 with the highest tmax 15.6°C>
Lowest: <12/2012 with the lowest tmin -1.2°C>
Average annual af: <29 days>
Average annual rainfall: <1238.720753424659 mm>
```

```
Number: <2>
Name: <Stornoway Airport>
Highest: <7/2016 with the highest tmax 18.2°C>
Lowest: <12/2012 with the lowest tmin -0.7°C>
Average annual af: <23 days>
Average annual rainfall: <1209.4006849314976 mm>
```

Tab: "Weather Data"



The "Weather Data" tab positioned on the GUI, is indicated in the appendix 2. (Point 1).

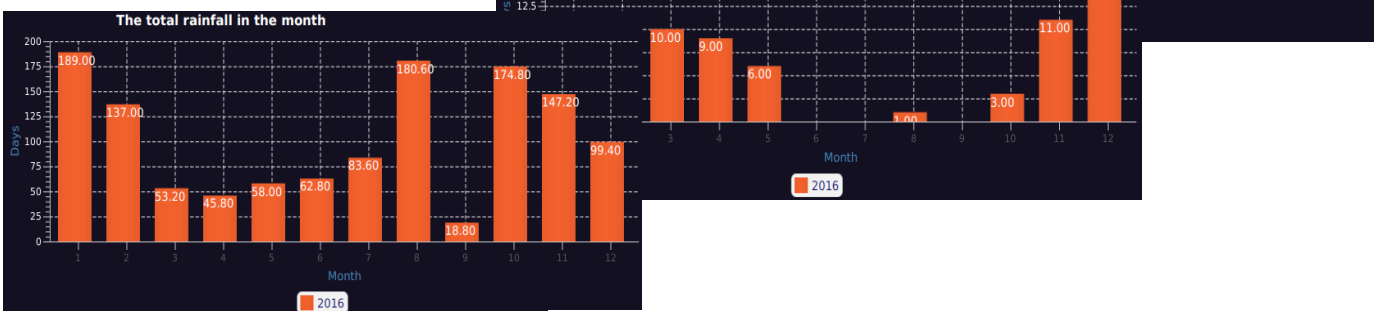
Scroll pane: on the right side appendix 2. (Point 2) implemented. Allows to fit all the necessary charts to display the weather data in the fixed size GUI.

Basic requirement 3: the user is able to select an individual meteorological station and year appendix 2. (Point 3) he wishes to be displayed in charts. Then, press the "Submit" button and the selected datasets are populated over 3 charts.

Appendix 2. Weather in 2019 tab.

"The mean minimum/ maximum temperature in the month" – in one line chart, creates ability to make clear visual correlations.

"The number of air frost in the month" and "The total rainfall in the month" are in separate bar charts:



The above values are stored in the following series:

```
private XYChart.Series<String, Number> tempMin1 = new XYChart.Series<>();
private XYChart.Series<String, Number> tempMax1 = new XYChart.Series<>();
private XYChart.Series<String, Number> frost1 = new XYChart.Series<>();
private XYChart.Series<String, Number> rain1 = new XYChart.Series<>();
```

Additional feature 3: ability to compare years against other years for the same station appendix 2. (Point 4). User clicks the "onAction" - "Add Year" menu button 1. The list of stored years appears 2. Simply select a year user wants to compare and the dataset populates over 3 (above discussed) weather charts.

The above series ending with integer "1" are compared with the below series ending names with integer "2".

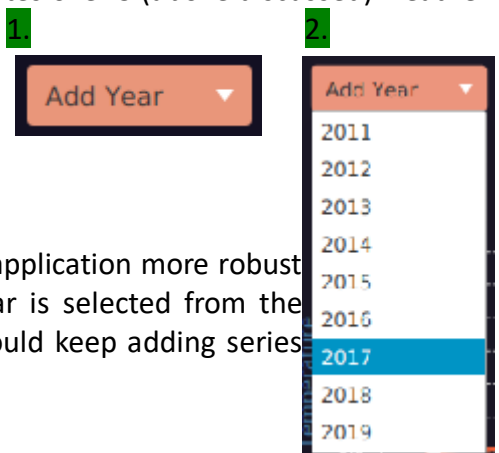
```
private XYChart.Series<String, Number> tempMin2 = new XYChart.Series<>();
private XYChart.Series<String, Number> tempMax2 = new XYChart.Series<>();
private XYChart.Series<String, Number> frost2 = new XYChart.Series<>();
private XYChart.Series<String, Number> rain2 = new XYChart.Series<>();
```

Additional UX error handling has been implement by making the application more robust and preventing from adding additional years when a certain year is selected from the menu button. Without the code displayed on the below, users could keep adding series and clutter the visualisation.

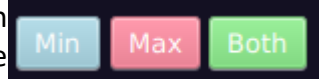
@FXML//avoids chart clutter and compares only two sets of year at once

```
private void addYear() {
    if (!toggleData.getItems().isEmpty()) {
        if (metStationData.getValue() != null) {
            MinMaxDataChart.getData().remove(tempMax2);
            MinMaxDataChart.getData().remove(tempMin2);
            barFrostDataChart.getData().remove(frost2);
            barRainfallDataChart.getData().remove(rain2);

            stationDataCharts(metStationData.getValue(), toggleData.getValue(), tempMax2, tempMin2, frost2, rain2);
        }
    }
}
```

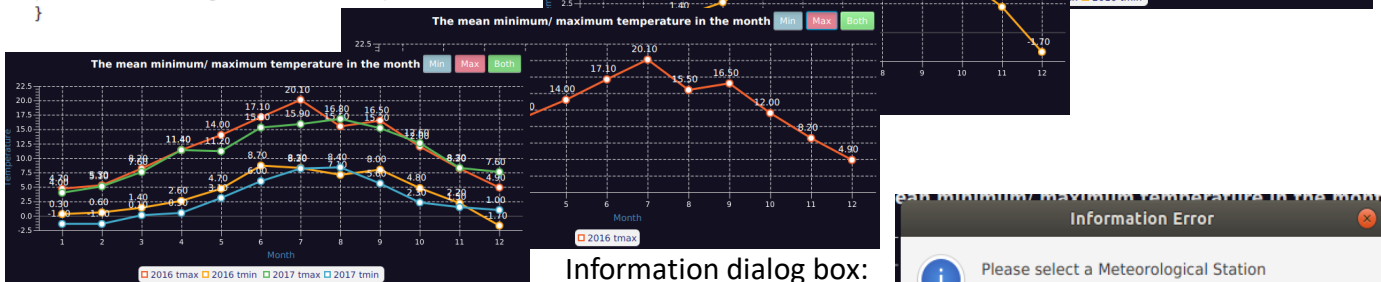


Additional feature 4: to manipulate displayed temperature datasets over “The mean minimum/ maximum temperature” line chart. By default “Both” temperatures have been set to display. Click on the implemented toggle buttons appendix 2. (Point 5). The user, is able to filter datasets as illustrated in the code example on the below. By clicking on the “Min” button to view the mean lowest, “Max” – mean highest, or click on the “Both”, to display – Min & Max. If no data selected and any toggle button is clicked, the information dialog box becomes displayed. **Note:** the same feature applies in years against other years mode and in *add station mode “weather comparison” tab3.

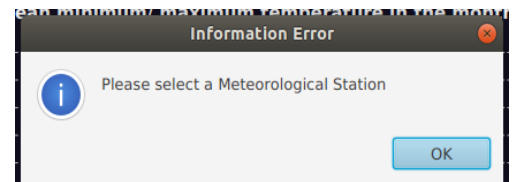


```
@FXML
private void tbMinClicked() {
    viewChart();

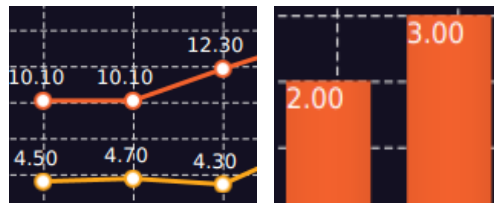
    if (toggleData.getValue() != null) {
        addSeries();
        MinMaxDataChart.getData().remove(tempMax2);
    }
    MinMaxDataChart.getData().remove(tempMax1);
}
```



Information dialog box:



Additional feature 5: every single node in charts has a numerical label, with set values:

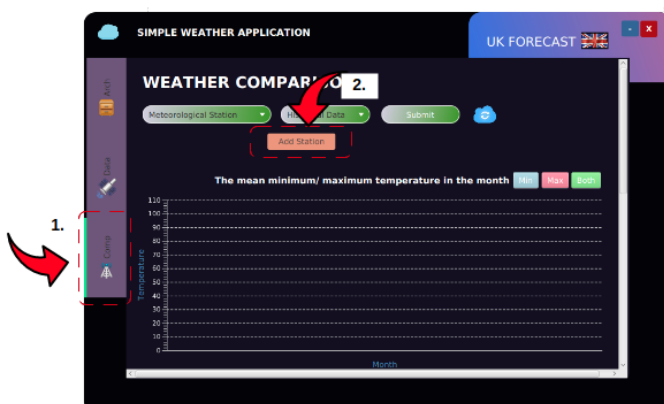


Additional feature 6: the data displayed in charts can be cleared. By clicking the reset button appendix 2. (Point 6). e.g. “onAction” button. On click getData().clear **Note:** clears only charts and keeps information in the combo- boxes.



```
@FXML
private void clearData2() {
    minMaxComparisonChart.getData().clear();
    airFrostComparisonChart.getData().clear();
    rainFallComparisonChart.getData().clear();
}
```

Tab: "Weather Comparison"

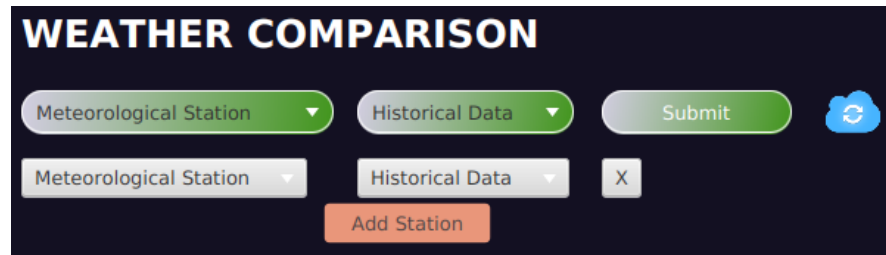


The “Weather Data” tab positioned on the GUI, indicated in the appendix 3 . (Point 1).

Appendix 3. Weather comparison tab.

Additional feature 7: users are able to add additional meteorological stations and historical data to compare, by clicking on the “Add Station” button appendix 3. (Point 2). 2 combo-boxes indicating meteorological station and historical data, plus, 1 button with implemented “x” function to remove selected station appears.

Note: user should select only 1 comparison station to avoid cluttering the line chart.



The code on the below: sets “comparison StationBox”, “yearBox” and “button” (x) locations in the grid. (Node child int rowIndex, int colspan, int rowspan). Will automatically expand the grid to accommodate the content.

```
stationsComparisonGrid.add(stationsBox, i: 1, comparisonGridRowIndex, i2: 2, i3: 1);
stationsComparisonGrid.add(yearsBox, i: 3, comparisonGridRowIndex, i2: 1, i3: 1);
stationsComparisonGrid.add(button, i: 4, comparisonGridRowIndex, i2: 1, i3: 1);
```

Another piece of code for ensuring code robustness: To prevent the system from displaying series on the graph, unless the station’s name and the year dropdown menus have buttons selected.

```
//Iterates over stations and years
for (int i = 0; i < stationBoxes.size(); i++) {
    ComboBox stationBox = stationBoxes.get(i);
    ComboBox yearBox = yearBoxes.get(i);

    String selectedStation = (String) stationBox.getSelectionModel().getSelectedItem();
    String selectedYear = (String) yearBox.getSelectionModel().getSelectedItem();

    // Adds only those stations are selected and have selected year
    if (selectedStation != null && selectedYear != null) {
        StationData stationData = StationsUtil.load(selectedStation, selectedYear);

        comparisonStations.add(stationData);
    }
}
```