**Implementation in C++**

**Main Window / Dashboard**

The MainWindow class was designed as one large singleton class and oversees all runtime behaviour of the applications GUI. Our application uses a single global variable to make the object accessible and only ever has one sole instance of the class initialized.

Along with overseeing behaviour caused by user interaction with the GUI, the MainWindow class overrides some Qt methods to allow for the Drag and Drop of files and Keyboard shortcuts. Specifically, for Drag and Drop the MainWindow class implementes the “dragEnterEvent” and the “dropEvent” methods to get file path for a dropped file and calls the correct load file method depending on which category is selected (load\_teach, load\_pub etc.). For the keyboard shortcuts the MainWindow class implemented Qt’s“keyPressEvent” and “keyReleaseEvent” methods to keep track of which keys have been pressed and which action to trigger.

**Sort Orders**

The CustomSort class is responsible for the creation and editing of sort orders in the application. When the user presses the “New Sort” or “Edit Sort” button, the MainWindow class creates a brand-new instance of the CustomSort class and passes in a list of Fields that the user should be able to select from. The user can then select which Fields they want to sort by and when the save button is clicked the sort order passed back to the MainWindow and saved to a text file using an instance of the QSortListIO class and its “saveList” method.

QSortListIO simply takes a list of all the sort orders and uses QT’s built in QDataSteam to write the sort orders into a text file. Then on relaunch of the program QSortListIO’s “readList” method is called to read back the sort orders from the text file.

**Visualizing Data**

Three out of the four different types of charts in our program are implemented using the QCustomPlot Library. While the remaining chart, the pie chart is implemented using the PieChartWidget class.

When a user clicks on a datapoint in the MainWindow’s treeView, the corresponding on\_treeView\_clicked method is called. This method gets the data item that was clicked and its child data items and creates a vector of string/double pairs to be plotted on the graph. The on\_treeView\_clicked method then calls the correct setup method for each of the 4 different chart types.

For the each of the LineChart, BarChart and Stacked BarChart the setup method takes the vector of string/double pairs and graphs them using an instance of QCustomPlot. For the PieChart the setup method takes the vector of string/double pairs and graphs them using an instance of the PieChartWidget class.

**Loading Files**

To read in the CSV file the system MainWindow uses an instance of the CSV reader class. Firstly, the MainWindow creates a new instance of CSVReader and passes in the file path as parameter. The constructor for CSVReader then calls the loadCSV method to read in all the data from the file.

The MainWindow class then uses the CSVReaders getHeaders() and getData() methods to get the headers and data from the csv file and passes it into RecordManager instance we saves the data in a Tree to be displayed in the MainWindows TreeView.

**Save State**

In order to save the last file opened by our program, when the application closes the MainWindows “closeEvent” method is called. This method simply calls the MainWindow’s “saveProgramState” method and accepts the close event. The “saveProgramState” method uses an instance of Qt’s QDataStream to save the file path of the most recently opened file and the index of tab of the category that is currently open.

In order to load the state back into the program, the MainWindow classes constructor creates a different instance of Qt’s QDataStream and reads back in the file path and category tab index and calls the load method for that category to load in the csv file.