**Security:**

A simplistic login system was implemented requiring a correct username and password to gain access to the rest of the system. upon distribution of the solution the client would receive their unique username and password. Ideally an option would be available that allows the user to create their own account but due to time constraints that was not implemented. Due to the client being the only person to access the program the login system is quite redundant, however was still added as a safety feature to protect against the chance of a malicious user gaining access to the clients device, and wished to view potential person information of student’s data stored within classes. If more time was allocated implementing an encryption system would benefit this, as it would add an extra layer of protection to the currently raw data files used by the solution

Backups:  
Throughout the development of the software updates were regularly backed up to a github repository once a day in both raw code and the solution zip file if required. With github allowing the user to view previous commits if a fatal bug was introduced it was simple to just revert the version to a previous one. Due to github databases also being stored online on the cloud unless the cloud is down it is possible to access the solution from any device through github, greatly reducing the risk of devices specific issues  
The solution was also backed up on the main device and a laptop meaning if by some minute chance both github and one of the devices stop being functional a third backup exists

**Data structures:**

The justifications of most data structures are done within the internal documentation of program. The main types of structures used were:

* Arrays and lists for sorting, deleting and manipulating simple index data values
* Associate arrays/dictionaries for their key, value pairs allowing to assign two related variables to one structure and binding them together. Uses included keeping track of the saved status’ of classes (class name being the key and their saved status being the value; allowed easy searching and validation of the two values without needed an extra array or string) and stat calculations – mainly finding mode of an array (value name being the key and the number of occurrences being the value; easiest method to store all names and occurrences within a single structure; also the easiest way to check for highest number of occurrences simply by iterating over the dictionary values).
* Classes – custom classes were created to store relevant data and functions. These classes were used to access said functions and data globally across the solution allowing a function to be called from another form that the original that the class is contained within providing higher readability (data or functions being called by classname.function; the user can immediately see the respective class the function or data belongs to
* Hashset – was only used once throughout the solution however due to its special property that only allows it to store unique value and removed any duplicates it was the most suitable option available to store the column values when creating a new class as all the columns were required to be unique

Within the solution there were minimal uses of data structures to store and manipulate data from class files. Data saved within an xml file could be imported into a datagrid that acted as a ‘class’ and modifications made to the datagrid would then be appended to the file by reading each column/row/cell and constructing the file element by element. Any sorting, deleting and manipulation of class data was done purely within the datagrids and then transferred to their respective xml file, or a new one would be created if it didn’t previously exist   
A singular text file was used to store all correct usernames and passwords, with the solution reading from the file and checking against user input to validate that they used the correct username and password