Developer's Documentation

Feedback Management System

Table of Contents:

Overview:

The program is designed to store and maintain data about feedbacks provided by students about courses offered by our university. The feedback consists of various questions about the course, its content, efficiency of teaching methods, and so on. The feedback once submitted can be viewed/managed by the administrator.

As a student, you can fill-out the feedback form and view your submitted feedbacks, while the administrator has much more flexibility. The abilities range from adding/removing students to adding/removing questions in feedback form. We achieve that by making use of file handling, and strings generally. But as we proceed further in this document, you will understand the full functionality of the program.

Scope:

The program is be capable of following functionalities:

Administrator:

- Log-in to the system using your Username/Password.
- Manage the Feedback Form
  - View the Feedback Form
  - Add a New Question
  - Remove an Existing Question
  - Search in the Feedback Form

- Manage Student
  - Check Total Number of Students
  - Search for a Student
  - Remove a Student
  - Add a New Student

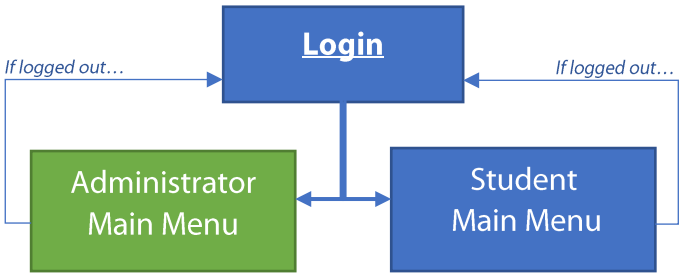
- Manage Feedbacks Submitted by Students
  - Check Total Number of Feedbacks Submitted
  - Print a Submitted Feedback
  - Remove a Submitted Feedback

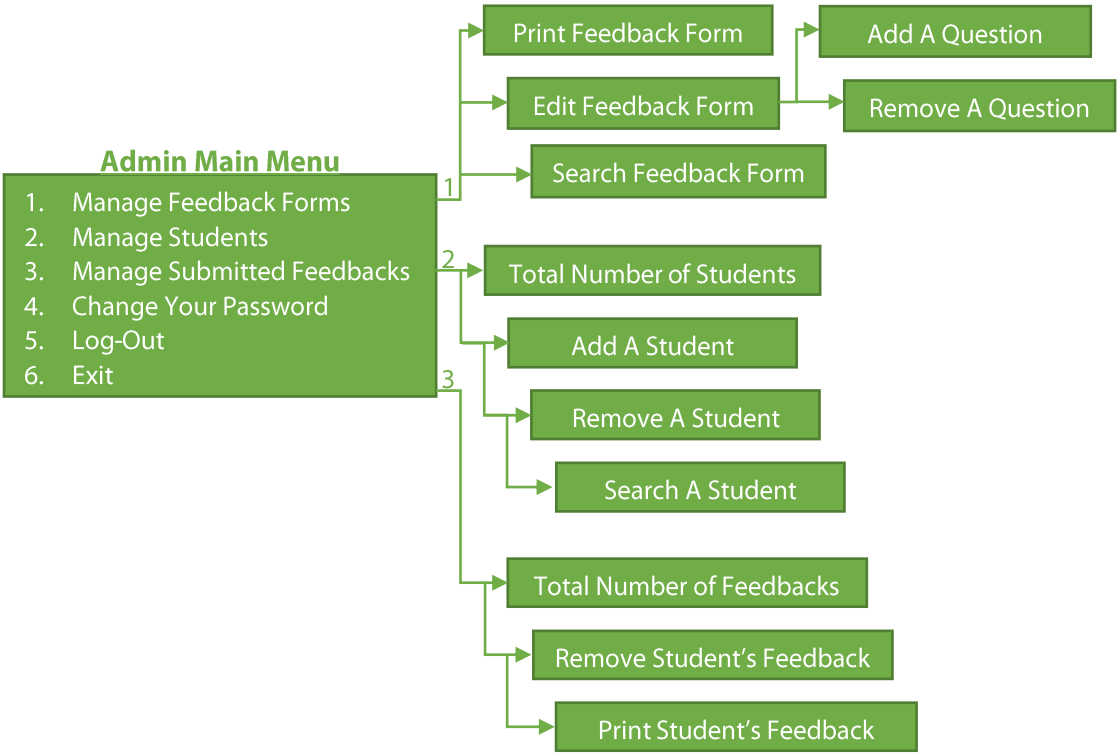
- Change your Password

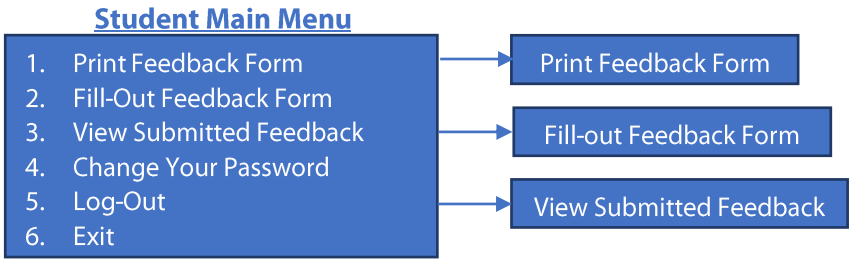
Student:

- Log-in to the system using your Username/Password.
- View the Feedback Form
- Fill-Out the Feedback Form
- View your Submitted Feedback
- Change your Password

Solution Design:







Included Libraries:

```
#include <iostream>
#include <string>
#include <fstream>
#include <windows.h>
#include <stdio.h>
#include <filesystem>
```

Following libraries have been included into the project. The filesystem library was an experimental library until it was fully incorporated in C++17. We will make use of it later in some Admin functions.
[consult: A.2.1 Check Total Number of Students**].**
[consult: A.3.1 Check Total Number of Submitted Feedbacks**].**

Files:

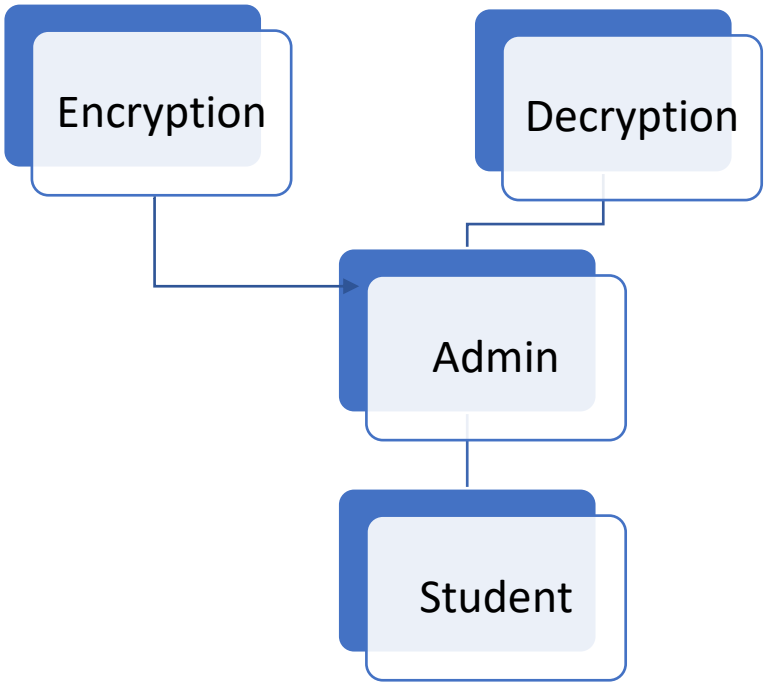
- CPP Files
  - **Feedback.cpp:** *This our main file, it controls the flow of the code, and handles the menus and the choices entered by the user.*
  - **Independent_Functions.cpp:** *This file contains all the functions that do not need to be inside any class.*
  - **Admin.cpp:** *This file contains the constructor, setters/getters, and further implementations for the Admin class.*
  - **Student.cpp:** *This file contains the constructor, and further implementations for the Student class.*

- Header Files
  - **Independent_Functions.h:** *Contains function declarations and includes all the libraries.*
  - **Caesar_Cipher.h:** *Contains pure virtual function declarations.*
  - **Admin.h:** *Contains function declarations for Admin class.*
  - **Student.h:** *Contains function declarations for Student class.*

- Pre-Existing Data Files
  - **Feedback Form & Submitted Feedbacks:** *Feedback Form is a text file which stores all the questions. Submitted Feedbacks folder contains all the feedbacks submitted by students.*
  - **Admin Folder:** *Contains Admin Username and Password (Encrypted).* ***[****consult: Admin Class Functions/Implementations: Encryption Function****]***
  - **Student Folder:** *Contains Students' Usernames and Passwords (Encrypted).* ***[****consult: Admin Class Functions/Implementations: Decryption Function****]***

Classes:

The program has 2 abstract classes (Encryption & Decryption) as base classes, and Admin class is derived from those classes. The student class is derived from Admin class. The program utilizes multiple inheritance.



Independent Functions:

- `void CLEAR_SCREEN ()`

This function clears the console output to avoid clutter.

- `void EXIT ()`

This function exits the program when user selects that option.

- `void TITLE ()`

This function is used to display the title of the program.

- `int MAIN_MENU_OR_EXIT (int choice)`

Asks the user whether he/she wants to exit or return to main menu. Returns User's Choice as integer.

Admin Class Functions/Implementations:

The Admin class has private variables for Username and Password; hence the class has setters and getters in place.

```
void ADMIN::SET_USERNAME (string SET_THIS_USERNAME)
string ADMIN::GET_USERNAME () //// Getter for USERNAME ////
void ADMIN::SET_PASSWORD (string SET_THIS_PASSWORD)
string ADMIN::GET_PASSWORD () //// Getter for PASSWORD ////
```

- **Login Function:**
  - `bool ADMIN::LOGIN_FUNCTION(bool ITS_ADMIN)`

    Checks the Username for Admin, transforms the string to lower alphabets to minimize the possibilities of bad login. Returns "TRUE" if Username is good.

- **Login Validation Function:**
  - `bool ADMIN::LOGIN_VALIDATION_FUNCTION ()`

    Checks the Password for Admin. Decrypts the Password and matches it to User's Input. Returns "TRUE" if Password is good. It also makes use of Exception Handling and throws Bad/Unsuccessful login attempts.

- **Get Username/Password Function:**
  - `string ADMIN::GET_LOGIN_DATA (const string& FILE_NAME)`

    Opens the file at given path (FILE_NAME), and returns the data read into a string. This function is being used to match Usernames and Passwords.

- **A.1.1) Print Feedback Form Function:**
  - `void ADMIN::PRINT_FEEDBACK_FORM ()`

    Reads the Feedback Form into a Dynamic Array, and then shows the output on screen. Each line comprises of one question; hence, each index of the array carries one question.

```
FEEDBACK_FORM = new string[LINE_COUNT];//...Dynamic String Array that has element number = lines in feedback form...//
                                       //... [each line has one question, hence each index will carry one question]..//

READ.open("./Feedback Form/Feedback Form.txt", ios::in);
{
    while (getline(READ, READ_LINE))
    {
        FEEDBACK_FORM[COUNTER] = READ_LINE;
        COUNTER++;
    }
}
READ.close();

for (int i = 0; i < LINE_COUNT; i++)
{
    cout << FEEDBACK_FORM[i] << endl;
}
delete[]FEEDBACK_FORM;
```

- **A.1.2.1) Add A New Question Function:**
  - `void ADMIN::ADD_QUESTION ()`

Reads the new questions into a string and appends it to the end of the feedback form.

```
QUESTION = to_string(LINE_COUNT + 1) + ") " + QUESTION;

READ.open("./Feedback Form/Feedback Form.txt", ios::app);
{
    READ << endl << QUESTION;
}
READ.close();
```

- **A.1.2.1) Remove A Question Function:**
  - `void ADMIN::REMOVE_QUESTION ()`

Asks the user for which question to remove. Reads the whole feedback form into an array of strings and over-writes the file while ignoring the question to be removed.

```
ofstream OVER_WRITE;
OVER_WRITE.open("./Feedback Form/Feedback Form.txt");
{
    for (int i = 0; i < LINE_COUNT; i++)
    {
        if (i != QUESTION - 1)
        {
            OVER_WRITE << FEEDBACK_FORM[i] << endl;
        }
    }
}
OVER_WRITE.close();
```

- **A.1.3) Search in Feedback Form Function:**
  - `bool ADMIN::SEARCH_IN_FEEDBACK_FORM ()`

Reads the whole file into dynamically allocated array of strings and searches each index for user's search query. If found, returns true.

```
for (int i = 0; i < LINE_COUNT; i++)
{
    if (FEEDBACK_FORM[i].find(QUERY) != string::npos)
    {
        return TRUE;
    }
}
return FALSE;
```

- **A.2.1) Check Total Number of Students Function:**
  - `void ADMIN::TOTAL_STUDENTS ()`

  Makes use of Directory Iterator to count all the files inside the Student folder. File System library is being used here.

```
int TOTAL_FILES = 0;
string FILE_PATH = "./Students/";
for (auto& p : filesystem::directory_iterator(FILE_PATH))
{
    TOTAL_FILES++;

}
```

- **A.2.2) Search A Student Function:**
  - `void ADMIN::SEARCH_STUDENT ()`

  Asks the user for NEPTUN code, and if the file exists, shows an output that the student was found in the system. Otherwise displays warning that the student doesn't exist in the system.

```
if (!CHECK_EXISTING_FILE(FILE_PATH.c_str()))
{
    cout << "\n\n\n";
    cout << "          _____________________________________________________\n";
    cout << "         |  No Student exists under this NEPTUN Code...        \n";
    cout << "         |_____________________________________________________\n";
}
```

- **A.2.3) Remove A Student Function:**
  - `void ADMIN::REMOVE_STUDENT ()`

  Asks the user for NEPTUN code, and if the file exists, deletes it using remove (). Otherwise displays warning that the student doesn't exist in the system.

- **A.2.4) Add A Student Function:**
  - `void ADMIN::ADD_STUDENT ()`

  Asks the user for NEPTUN code, and if the file exists, displays a warning that the student already exists in the system. Otherwise, makes a new file inside Students folder, and sets a temporary password for the student.

- **A.3.1) Check Total Number of Submitted Feedbacks Function:**
  - `void ADMIN::TOTAL_SUBMITTED_FEEDBACKS ()`

  Makes use of Directory Iterator to count all the files inside the Submitted Feedbacks folder. File System library is being used here.

- **A.3.3) Remove A Student's Feedback Function:**
  - `void ADMIN::REMOVE_SUBMITTED_FEEDBACK ()`

Asks the user for NEPTUN code, and if the file exists, deletes it using remove (). Otherwise displays warning that the student hasn't submitted the feedback yet.

```
    remove(FILE_PATH.c_str());

    cout << "\n\n\n";
    cout << "          _____________________________________________________\n";
    cout << "         |  Submitted Feedback File Deleted Successfully... \n";
    cout << "         |_____________________________________________________\n";
}
```

- **A.4) Change Password Function:**
  - `void ADMIN::CHANGE_PASSWORD ()`

Function for changing password. Overwrites the previous password and encrypts it.

```
string NEW_PASSWORD;
cout << "                >> Enter your NEW PASSWORD: ";
cin >> NEW_PASSWORD;

string FILE_PATH = "./Admin/ADMIN_PASSWORD.txt";
ofstream OVER_WRITE;
OVER_WRITE.open(FILE_PATH.c_str());
{
    OVER_WRITE << NEW_PASSWORD;
}
OVER_WRITE.close();
ENCRYPT(FILE_PATH.c_str());
```

- **Check Existing File Function:**
  - `bool ADMIN::CHECK_EXISTING_FILE (const string& FILE_NAME)`

Checks if a file exists in the system or not. Being used by various other functions like search, add, delete etc.

```
fstream READ;
READ.open(FILE_NAME.c_str(), ios::in);
{
    if (READ.is_open())
    {
        READ.close();
        return TRUE;
    }
    return false;
}
```

- **Encryption Function:**
  - `void ADMIN::ENCRYPT (const string& FILE_NAME)`

Takes a file path as parameter and uses Caesar cipher algorithm to encrypt the data. NOTE: The function is being over-ride, it belongs to ENCRYPTION class as a pure virtual function.

What it really does is that it takes all the characters and adds a specific number to it, hence the text seems like gibberish. You can make it as complex as you want. You can add, subtract, or perform any mathematical operation on it, as long as it can be reversed (for decryption purposes, obviously).

If you perform more than one mathematical operation, e.g. (x+3)-6, then in decryption, you must do the reverse of it, i.e. (x+6)-3.

```
string DATA;
fstream READ;
READ.open(FILE_NAME.c_str(), ios::in);
{
    getline(READ, DATA);
    for (unsigned int x = 0; x < DATA.length(); x++)
    {
        DATA[x] += 3; // using simple ceasar cipher...
                      //add a number to each character
                      //and garble the whole string character by character
    }
}
READ.close();

ofstream OVERWRITE;
OVERWRITE.open(FILE_NAME.c_str());
{
    OVERWRITE << DATA; // over-write the old data with new encrypted data...
}
OVERWRITE.close();
```

- **Decryption Function:**
  - `void ADMIN::DECRYPT (const string& FILE_NAME)`

  Takes a file path as parameter and uses Caesar cipher algorithm to decrypt the data. NOTE: The function is being over-ride, it belongs to DECRYPTION class as a pure virtual function.

```
string DATA;
fstream READ;
READ.open(FILE_NAME.c_str(), ios::in);
{
    getline(READ, DATA);
    for (unsigned int x = 0; x < DATA.length(); x++)
    {
        DATA[x] -= 3;// using simple ceasar cipher...
                     //add a number to each character
                     //and garble the whole string character by character
    }
}
READ.close();

ofstream OVERWRITE;
OVERWRITE.open(FILE_NAME.c_str());
{
    OVERWRITE << DATA;// over-write the old data with new encrypted data...
}
OVERWRITE.close();
```

Student Class Functions/Implementations:

- **Login Validation Function [Overloaded]:**
  - `bool STUDENT::LOGIN_VALIDATION_FUNCTION()`

  First, Checks the Username for Student. If a file under that username exists, proceeds to check the Password.

  Decrypts the Password and matches it to User's Input. Returns "TRUE" if Password is good. It also makes use of Exception Handling and throws Bad/Unsuccessful login attempts.

  If Username/Password is bad, the program goes back to Login Screen with a message telling to try again.

- **B.2) Fill-Out the Feedback Form Function:**
  - `bool STUDENT::STUDENT_FILL_OUT_THE_FEEDBACK_FORM_FUNCTION (int choice)`

First, it checks if the feedback has already been submitted or not. If it's not already been submitted, then displays the feedback questions one by one, and lets the user answer between 1-5:

```
___ 1.STRONGLY AGREE    2.AGREE    3.NEUTRAL    4.DISAGREE    5.STRONGLY DISAGREE __
```

Incase of wrong input, exception is throwed, and user is asked to try again. The answers are appended into the submitted feedback file.

```
if (choice == 1)
{
    WRITE_FEEDBACK_ANSWERS << QUESTION_LINE << endl << "    1. STRONGLY AGREE" << endl << endl;
}
if (choice == 2)
{
    WRITE_FEEDBACK_ANSWERS << QUESTION_LINE << endl << "    2. AGREE" << endl << endl;
}
if (choice == 3)
{
    WRITE_FEEDBACK_ANSWERS << QUESTION_LINE << endl << "    3. NEUTRAL" << endl << endl;
}
if (choice == 4)
{
    WRITE_FEEDBACK_ANSWERS << QUESTION_LINE << endl << "    4. DISAGREE" << endl << endl;
}
if (choice == 5)
{
    WRITE_FEEDBACK_ANSWERS << QUESTION_LINE << endl << "    5. STRONGLY DISAGREE" << endl << endl;
}
```

Changes in Project Application Form:

- **Techniques Used in this Project**
  - OOP Application
  - Dynamic Memory Management
  - File Management
  - Inheritance
  - Multiple Inheritance
  - Polymorphism
  - Data Encapsulation
  - Exception Management

Testing:

To test the application, I made a couple of test files for students, and made sure that the login function worked correctly for all of them. I tested changing their passwords and encrypting and decrypting them. I also tested adding and deleting students.

Besides that, I tested the exception management by inputting wrong choices in menus, and the program behaved exactly as it should have, i.e. I was prompted to enter the right choice. I also filled out the feedback form as a student, and edited the feedback form as an Administrator, and tested the search functions for students and submitted feedbacks.

I am glad to report that the program compiles without errors and is capable of all the functionalities mentioned in this document.