Student Planner

Student id: c18307371

name: glen oneill

# Contents Page

[Contents Page 1](#_Toc103907892)

[Figures Page 2](#_Toc103907893)

[Chapter 1 - Introduction 3](#_Toc103907894)

[Motivation for project 3](#_Toc103907895)

[High-level aims of the project 3](#_Toc103907896)

[Main objectives of this system are: 4](#_Toc103907897)

[E/C/S/E (Ethical/ Commercial/ /Safety/Environmental) 5](#_Toc103907898)

[Chapter 2 - Background Research (Literature Review) 6](#_Toc103907899)

[Choice of programming language & supporting libraries 6](#_Toc103907900)

[Accessing web content programmatically 6](#_Toc103907901)

[GUI development 7](#_Toc103907902)

[Integrating with external APIs 8](#_Toc103907903)

[Work Plan 8](#_Toc103907904)

[Chapter 3 - Design 10](#_Toc103907905)

[Design Heuristics 10](#_Toc103907906)

[Matching system and real world 10](#_Toc103907907)

[Aesthetic and minimalist design 10](#_Toc103907908)

[Help users recognize, diagnose, and recover from errors 10](#_Toc103907909)

[Help users recognise, diagnose, and recover from errors 11](#_Toc103907910)

[Visibility of system status 11](#_Toc103907911)

[User control and freedom 11](#_Toc103907912)

[Recognition rather than recall 11](#_Toc103907913)

[Application layout design 12](#_Toc103907914)

[Working with external APIs 13](#_Toc103907915)

[Weather API 13](#_Toc103907916)

[Google classroom API 14](#_Toc103907917)

[Requirements for working with APIs 17](#_Toc103907918)

[OAuth Authorization 18](#_Toc103907919)

[Chapter 4 - Implementation 19](#_Toc103907920)

[Topic 1: Initial Research 19](#_Toc103907921)

[Topic 2: Starting With GUI 19](#_Toc103907922)

[Topic 3: Weather System (Non-API Version) 20](#_Toc103907923)

[Topic 4: Initial Use Of APIs 20](#_Toc103907924)

[Topic 5: OAuth API 21](#_Toc103907925)

[Topic 6: Designing a built-in text-editor 22](#_Toc103907926)

[Topic 7: Re-designing GUI 23](#_Toc103907927)

[Chapter 5 - Evaluation 24](#_Toc103907928)

[Software testing 24](#_Toc103907929)

[Functional and non-functional 24](#_Toc103907930)

[Types of testing (White/Black/Grey box) 25](#_Toc103907931)

[Testing for Inbox.py 26](#_Toc103907932)

[Testing for Appointments.py 27](#_Toc103907933)

[Testing for RoomLocations.py 28](#_Toc103907934)

[Testing for weatherAPI.py 29](#_Toc103907935)

[Testing for googleClassroom.py 30](#_Toc103907936)

[Testing for Grades.py 31](#_Toc103907937)

[Testing for textedit.py 32](#_Toc103907938)

[GitHub 33](#_Toc103907939)

[Chapter 6 – Conclusions & Future Work 34](#_Toc103907940)

[References 36](#_Toc103907941)

# 

# Figures Page

[Figure 1: win32com commands 7](#_Toc103868650)

[Figure 2: Error codes 8](#_Toc103868651)

[Figure 3: Work plan (weeks 1-4) 9](#_Toc103868652)

[Figure 4: Work plan (weeks 5-8) 9](#_Toc103868653)

[Figure 5: Sitemap 12](#_Toc103868654)

[Figure 6: Creating application on AccuWeather 13](#_Toc103868655)

[Figure 7: Inserting API key to code 14](#_Toc103868656)

[Figure 8: Console page 14](#_Toc103868657)

[Figure 9: Creating new project 15](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868658)

[Figure 10: Naming new project 15](#_Toc103868659)

[Figure 11: Filling in OAuth consent information 16](#_Toc103868660)

[Figure 12: Detailing Scopes 16](#_Toc103868661)

[Figure 13: Downloading credentials json 17](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868662)

[Figure 14: Saving json under name “credentials” 17](#_Toc103868663)

[Figure 15: Initial code to read emails 19](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868664)

[Figure 16: Initial code to read emails (displayed) 19](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868665)

[Figure 17: Inbox messages 19](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868666)

[Figure 18: Displaying inbox messages in GUI format 19](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868667)

[Figure 21: Pie-chart of grades generated 20](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868668)

[Figure 20: User input for inbox/calendar 20](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868669)

[Figure 19: Web-scraping weather info 20](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868670)

[Figure 22: Getting weather through API 20](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868671)

[Figure 23: Logging into Brightspace 20](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868672)

[Figure 24: OAuth on Brightspace 21](#_Toc103868673)

[Figure 25: Announcement Page 22](#_Toc103868674)

[Figure 26: Text-editor Page 22](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868675)

[Figure 27: Old design for grades page 23](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868676)

[Figure 28: Updated design for grades page 23](file:///C:\Users\gleno\Downloads\C18307371_REPORT%20(2).docx#_Toc103868677)

[Figure 29: Types of software testing 24](#_Toc103868678)

# Chapter 1 - Introduction

## Motivation for project

As a current college student, I know first-hand the inconvenience of having to check multiple applications and websites to gather the required information on a daily basis. If I wanted to check the weather so I would know whether or not I needed to prepare for the conditions, I would need to check a website or an application on my phone. If I wanted to check my current assignments, due dates, and grades, I would need to access Brightspace through either an application or the official website. If I wanted to check my emails or check my timetable/calendar activities to plan out my day and avoid my personal activities and academic colliding I would need to check my outlook app or use the website. Taking all of this into account I always wished there could be one central hub where all this information could be easily accessible as besides the tediousness of checking each individually, I was also prone to forgetting to check one due to the number of them which unfortunately led to some rainy commutes unprepared. Due to this I decided to take it upon myself to create this application and design it with all of the knowledge I have on what a student may need to know on a daily or weekly basis. I also wanted to create the app to be useable for any student and not just tailor made for me.

## High-level aims of the project

The overall requirement for my system is to provide a student with a central hub which will provide them with all the information they could possibly require in order to plan out their day/week or to check their grades etc.

### Main objectives of this system are:

* Display the students calendar appointments in a clear and concise method. The student will be greeted with a well-designed GUI page, that will contain their most recent calendar appointments. The reason for this is to allow the student to see what classes they have for that particular day/week, what classroom/location it’s taking place in, and at what time said event will occur.
* Allow the student to check the weather in their desired county. The user will be asked which county they would like to check the weather for, once input it will display the temperature for said county. The reasoning behind this feature is that as a student myself I understand how knowing the weather can make a big difference to your day, for example if it’s heavily raining one day you will of course want to wear a well-protected jacket with a hood or bring an umbrella, if you commute to college this information can be very useful, as from personal experience I haven’t been aware of the weather for a day and paid the price for not preparing correctly!
* Create multiple GUI pages which the user can switch between effortlessly. Instead of fitting everything onto one page I opted for a multiple page approach, however I wanted this transition from page to page to be as smooth as possible, the way in which I did this was through closing of the current page and opening of the new one. I also made the new page open in full screen (sizes the GUI page to fit the entire screen) straight away giving the smooth transition effect of changing pages, as opposed to making the user rescale the page themselves.
* Add entries to their outlook online calendar. The reason for this is to keep consistent with the main goal of creating a one stop shop for the student and not requiring them to open the outlook application separately to add a new Calendar entry. By adding new entries, the student may be alerted if a clash in time with their new entry and an existing one has occurred, this will allow the student to plan out their personal and academic life without overlapping times, which is very important to achieve such balance.
* Display the users’ grades in pie chart form. The reasoning behind this feature is to provide the student with a very easy to read image format of their grades. Once the user has input their grades reading the pie chart is simple and you can see from a quick glance which modules the student is excelling in, and which require more attention.
* Show a correlation between sleep and grades. The student will be asked for both their grades and their amount of sleep-in hours. The program will then use this information to try show a correlation between higher/lower grades with more or less sleep time on average. Showing this information to a student may get them to sleep more if they notice their grades are better when they are well rested, which would be a good benefit to their health.
* Ask the student what part of the outlook application they would like displayed. Whilst certain pages of the outlook application I want hard coded into a page such as calendar and inbox, there are also many more parts of outlook such as drafts, deleted messages, sent messages, etc. Having a page that asks the user which one they would like displayed would continue in my pursuit to create a one stop shop application and prevent the user from needing to open anything else.
* Send an email from the application. The student should be able to email a lecturer from the application, perhaps to confirm due dates etc. once again the student shouldn’t have to use a browser or separate app to complete this task, hence why it will be added in.

## E/C/S/E (Ethical/ Commercial/ /Safety/Environmental)

There are two main areas of concern with regards to commercialisation are with my weather application and also my Google classroom section. The reason these two are a concern in relation to any potential commercialisation of my application is due to the fact that my weather system and my classroom section both use an API which is owned by a separate company AccuWeather and then Google for my classroom section, both of which would need licensing and/or permission for any fair usage of their platforms in my application.

From a GDPR point of view it holds the same level of security as Outlook and also Google. The reasoning for this is due to the fact that in order to retrieve your emails for example from Outlook, you must first be signed into the Outlook application, meaning that the security is the essentially derived from Outlook as without being signed in the program will refuse to display or even retrieve any personal information. The same applies to the Google Classroom API as the user must sign-in to their google account, however there is also another layer of security involved which is the token that is generated on the Google Cloud console, which the user must sign in to retrieve.

# Chapter 2 - Background Research (Literature Review)

## Choice of programming language & supporting libraries

Due to my program using python as its coding language I had access to a wide variety of sources of information to take inspiration from. The reason I decided to use python for this particular project was due to my previous knowledge of the language combined with an interest to learn more such as working with GUI and APIs for the first time. Python also has a substantial number of libraries which can be easily accessed to use even more features than are readily available, [9] an example of a library I used in this project was regex or regular expression, which is a sequence of characters that forms a search pattern.

## Accessing web content programmatically

The first thing I researched with regards to my project was how to access outlook emails. My initial thought process was to use web-scraping, however with some research I found a library named “win32com”. [1] Win32 allows a user to interact with COM objects and to automate Windows applications with python, however the one drawback of it is that there is no open-source version readily available, after much research the only solution I could find for retrieving email information from outlook on a Linux machine or other OS (Operating Systems) that is not windows would be to use web scraping. [10] The COM objects mentioned earlier are Microsoft Component Object Model (COM) which is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. Continuing on with my research into win32 I found the documentation on the official Microsoft websites to be very informative and useful in furthering my progress in achieving my end goals. [2] The way in which I manipulated which part of outlook I wanted to display was mainly through the calendar = outlook.GetDefaultFolder(9) command. The number in the bracket decides the section, in my example I’m using 9 which correlates to the calendar, other numbers can be used however as shown in figure 1.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 1: win32com commands

## GUI development

The next major form of research I conducted was integral to my project as it involved

implementing a graphical user interface (GUI). [3] For my initial trialling of applying a GUI I researched a basic program in which the user would enter a folder location and be able to select and display an image of their choice, once I understood the basics of how the GUI window was being displayed and taking user input, I decided to further my research into

creating multiple pages which could be assigned to appropriately named ‘next page’ and ‘previous page’ buttons. [4] The documentation on ‘pythonguides’ heavily inspired my code and allowed me to achieve an even greater understanding of how the GUI process operated, from there I was able to implement my own ideas such as displaying images and displaying information from the outlook application. The next two important pieces of research I conducted both interlink to providing the same functionality which is allowing the user to see the current weather for a specified location. [5] The first method I researched involved web scraping which is a method of gathering information from websites such as HTML (Hypertext Markup Language) code which can then be filtered down using regex to match the sequence specified for the search pattern. An example of this code that is being extracted with web scraping can be shown in the below image, which can then as previously stated be filtered to only display what is needed for your python application whether that be headings, images, text from a website etc.

## Integrating with external APIs

[6] The other method I researched involved using an API to obtain the weather, other methods of APIs used in this program will be further discussed later in this document. APIs was an area I had not delved into before and required more research as opposed to web scraping which I had used before. I learned much from my research such as firstly what an API actually is, how to use it, and different applications of it. First and foremost an API is an abbreviation for Application Programming Interface, and it is a server that can be used in order to retrieve and send data. APIs are most commonly used to retrieve data, and in my application as specified I used it in order to retrieve data for weather information. Status codes can be displayed when sending requests to a web server, these allow the user to gain an insight to what happened with their request:

Graphical user interface, text, application

Description automatically generated

Figure 2: Error codes

## Work Plan

[21] For this project it was decided to adapt an agile development approach as this is a commonly used software development methodology in the software industry today and facilitates dividing the project into small iterations which are then integrated together for the final product. Each iteration involved design, implementation, and testing phases before integrating it with the overall project code.

Chart

Description automatically generated

Figure 3: Work plan (weeks 1-4)

Chart

Description automatically generated

Figure 4: Work plan (weeks 5-8)

# Chapter 3 - Design

## Design Heuristics

As previously discussed, there is plenty of functionality in the program, and each of them were carefully implemented and designed to fit the system and make the users experience as easy and user friendly as it could be whilst also providing them with important details such as error messages. The student planner application has been carefully designed to implement multiple Usability Heuristics to help make the application as accessible and usable to a wide range of users. [20] The design heuristics I shall be discussing are the heuristics that were originally created by Jakob Nielsen which are essentially a broad rule of thumb when it comes to designing an application.

### Matching system and real world

[20] The first heuristic that will be discussed regarding this system is the match between system and the real world. This heuristic deals with creating the system to speak the user’s language. This is achieved through using words, phrases, and a logical order of information that the user will understand. The way in which this was implemented to the program was by using clear concise phrasing. The system doesn’t use overly complicated syntax and is created in a logical tabbed order that the user can follow and will not get lost in or question.

### Aesthetic and minimalist design

[20] The next heuristic we implemented was Aesthetic and minimalist design. A system should be created in such a way that it isn’t blank but also doesn’t look cluttered with unnecessary information whilst also drawing attention to the main sections of the page. In this program I made the GUI look very neatly spaced and sized with a combination of different coloured buttons for each page to help reinforce to the user that they have changed pages. Everything is very visible and as previously mentioned suitably spaced to help the user see the contents of each page as easily as possible without any confusion of what goes where or where a certain section is located.

### Help users recognize, diagnose, and recover from errors

[20] The next Heuristic That will be discussed is “Help users recognize, diagnose, and recover from errors”. This heuristic deals with not overloading the user with error messages or with error messages that contain complicated terminology. The error messages should be displayed in a clear manner that allows the user to know in simple terms what went wrong and how they could remedy it. The way in which this was incorporated into my program was through feedback text, an example of this is on the announcements page, if the user does not select a module and presses the button, they will be asked to select a module from the drop-down list. There is no use of any complicated syntax which could cause any potential confusion towards the user.

### Help users recognise, diagnose, and recover from errors

[20] The next heuristic ties in with the previously discussed “help users recognise, diagnose, and recover from errors”, as it is the heuristic of error prevention. The reason I say that this heuristic ties in so well is as mentioned we created text feedback which instruct the user on their error preventing them from making the same mistake twice. Alongside this, every textbox/field that is to be filled in by the user has a name above it clearly outlining what the field is looking for, therefore preventing the user from entering the wrong details and in turn preventing errors.

### Visibility of system status

[20] The heuristic of visibility of system status deals with always keeping the user updated with what is currently going on with the system. In this program we keep the user updated through a series of feedback text boxes, for example on the grades page when the user correctly enters their grade for each module, they will be told that a pie-chart has been generated showing their grades in a visual format.

### User control and freedom

[20] User control and freedom deals with user error and how the system should give them a clearly marked “emergency exit” option. In this program the user always has access to a clearly marked exit button.

### Recognition rather than recall

[20] Another heuristic that shall be discussed is the heuristic of “recognition rather than recall”. This heuristic deals with minimising the user memory load, as in details they may need to remember from one page to the next. The planner application is simplistic in design in terms of the user will not have anything to remember between tabs, as each page is independent from the next.

## Application layout design

Diagram

Description automatically generatedTo facilitate easy navigation across the application, each page was interlinked through buttons which allowed the user to navigate to whichever page they desired from any page they are currently on. Below is a sitemap (figure 5.0) giving a visual representation of how each page is linked.

Figure 5: Sitemap

## Working with external APIs

As discussed briefly in previous sections of this report an integral part of some of the program’s functionality relies on APIs. In this segment I shall be discussing the three different types of APIs which were used throughout the program, and how each differed in terms of how they were applied/prerequisites required to make them operable. The first ‘genre’ of API I shall discuss is the APIs which require no prerequisites such as tokens. The outlook API named Win32com was used to gather information from the user’s outlook account such as emails received, events in calendar, room locations for classroom meetings, etc. As mentioned, there are no tokens required for authentication with this API, with Win32com the only requirement is that said user has outlook installed on their device and are currently logged in. The lack of requirements made this API very intuitive to both use and set up. The next type of API I used in this project had only one requirement, a token. The weather system API required an API key in order to authenticate the application. API keys are commonly used to control the utilization of the API’s interface and track how it is being used. This is often as a precaution to prevent abuse or malicious use.

### Weather API

The weather API I used for this project was the AccuWeather API. The steps involved in setting up this API for use in my project was as follows:

1. Create an account on the AccuWeather API.
2. Create an application to generate a key

Graphical user interface, text, application

Description automatically generated

Figure 6: Creating application on AccuWeather

1. Use the key generated in the previous step in your code to authenticate your program and allow for the full weather functionality of the API to work.

Graphical user interface, text, application, email

Description automatically generated

Figure 7: Inserting API key to code

### Google classroom API

The final API I shall discuss which I incorporated in this project was one which required multiple layers of perquisites to operate. The google classroom API was used in this project due to time limitations with my first option. The original idea for this application was to allow the user to see their assignments due and classroom announcements from Brightspace which is the current website used by TUD, however for this to work I would need access to an admin account on Brightspace and this task would not be realistic given the timeframe, hence why google classroom was used in order to mimic how the application would work had access to Brightspace worked.

The steps involved in setting up the google classroom API were as follows:

1. Before we even use the google classroom website, we must first go to google cloud and create an account.
2. Once the account is created, we navigate to the console page (figure 8)

Graphical user interface, text, application, email

Description automatically generated

Figure 8: Console page

1. Once on this page, select the three bars in the top right corner to access the tabs. From here select ‘APIs and services.’
2. On the top of the APIs and services page the user will have the option to create a new project (Green circle), after this select new project (Red circle).

Text

Description automatically generated with low confidence

Figure 9: Creating new project

1. Name your project and create.

Graphical user interface, text, application, email

Description automatically generated

Figure 10: Naming new project

1. Select the OAuth consent screen on the side and fill in the required information.

Graphical user interface, text, application, email

Description automatically generated

Figure 11: Filling in OAuth consent information

1. Certain parts of the API require scopes which as detailed in figure 12 below is used to allow the program to access specific types of private user data. Graphical user interface, text, application, email

   Description automatically generated

Figure 12: Detailing Scopes

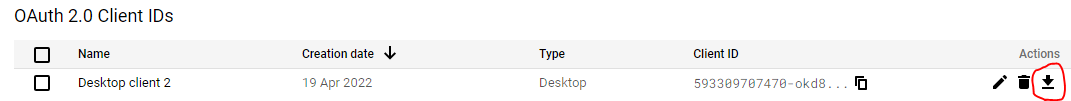
1. Navigate to the credentials page and select the download action next to the project that has just been created.

Figure 13: Downloading credentials json

1. Select “Download to JSON” and name the file “Credentials”

Table

Description automatically generated

Figure 14: Saving json under name “credentials”

1. This file can then be placed in the same folder as your python code
2. It is very important to specify the scopes before generating the JSON credentials, as if any changes are made the JSON (JavaScript Object Notation) must be re-downloaded for the effects to take change.
3. Once the JSON file has been correctly placed in with the python files the program can be run without specifying a token key like with the previous weather API, however the credentials file must be opened in the program.

### Requirements for working with APIs

As shown with the three APIs that have been discussed there are many different types in terms of requirements, some require keys, others require an account set-up and scopes, whilst others require an application on the device with the user logged in. In my opinion the google classroom API was the most difficult to setup however this was due to my lack of experience not only with APIs but their scopes as well. An error I made which hindered my progress for a short period of time was briefly touched on in one of the setup steps, and it was not specifying the correct scopes before downloading the credentials file. The lack of scopes meant that the program was attempting to access data which it was not authorised to do so. This one error allowed me to learn more about scopes and what exactly they were and why they were required.

### OAuth Authorization

[17] As shown in step six above, OAuth was used in the process of setting up the Google classroom API, but what is OAuth? OAuth is not an API nor is it a service, it an open standard used for authorization. [18] It is the industry-standard protocol used for authorization. OAuth 2.0 centres on client developer simplicity all while providing specific authorization flows for web applications, desktop applications, mobile phones, and more. [19] With regards to this program it is used to identify the application and allow end users to authenticate the application with Google. It allows the application to gain access to Google Cloud Classroom API on behalf of the end user.

# Chapter 4 - Implementation

## Topic 1: Initial Research

*Graphical user interface, text, application

Description automatically generated*For the first stage of the project, a small application to read and send emails was created through the use of “win32” which was discussed previously in the research section.

Figure 15: Initial code to read emails

*Text

Description automatically generated*

Figure 16: Initial code to read emails (displayed)

## Topic 2: Starting With GUI

Graphical user interface, text

Description automatically generatedA picture containing schematic

Description automatically generatedFor the second stage of this project, I started to work with GUI for the first time and interlinked my previous weeks work with this week by displaying the emails obtained from using “win32” in a GUI format. Once the above tasks had been completed, I attempted designing the GUI pages to look more appealing and to also add next and previous page interactive buttons. Both tasks discussed can be shown below:

Figure 17: Inbox messages

Figure 18: Displaying inbox messages in GUI format

## Topic 3: Weather System (Non-API Version)

[7] For the third section of this project, web scraping was used to implement a weather system, the weather information was obtained from the RTE website (Figure 19). Once the weather application was running, I worked on allowing the user to specify which folder they would like to receive information from on the outlook application, as opposed to having it hardcoded in (Figure 20). The final piece of work for this week was to allow the users to input their grades into a GUI window and then have a pie chart automatically generate giving a visual representation of their grades (Figure 21):

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generatedChart, pie chart

Description automatically generated

Figure : Pie-chart of grades generated

Figure : User input for inbox/calendar

Figure : Web-scraping weather info

## Topic 4: Initial Use Of APIs

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generatedFor this week’s work I firstly revisited my weather application and instead of using web scraping [8] I decided to use an API from AccuWeather, this new method of gathering information led to much more concise code and a more detailed weather description (Figure 22). The next idea I implemented was allowing the user to login to Brightspace through a GUI page, this idea will hopefully be built upon more in future weeks (Figure 23):

Figure 22: Getting weather through API

Figure 23: Logging into Brightspace

## Topic 5: OAuth API

The initial plan for this week is to automatically extract information from Brightspace to allow the user to see upcoming deadlines and finally to better format my GUI pages.

The initial idea for using Brightspace had to be altered slightly, this was due to time constraint with the project. To retrieve the information from Brightspace the user is required to register their application, however upon attempting this I discovered that admin privileges are necessary. After contact with the TUD helpdesk it was made clear that obtaining the correct privileges would not fit my timeframe.

Graphical user interface, text, application

Description automatically generated

Figure 24: OAuth on Brightspace

To compensate for this set-back I decided to use Google classroom as an alternative VLE (Virtual learning environment). Google classroom was the perfect substitute for Brightspace as I was able to add modules, make announcements, and assign tasks for the students which included deadlines. Once google classroom was setup, I was able to access the information through use of scopes which will be discussed in greater detail in the API section of this document. However, to briefly explain, the scopes act as permissions/authentication which tell the program which parts of the classroom can be accessed. The way I designed the GUI page for this section was with a drop-down menu as opposed to a textbox, my reasoning for doing so was so the user would not have to memorise their module title or prevent any incorrect spelling mistakes. Once the user selects their specified module from the list, they press the button which will then return text displaying the announcement for said module alongside the date it was posted. My design idea behind this page, and other pages in this program was to make it as simplistic as possible and not overload the user with pointless information, as can be shown in figure 25.

Text

Description automatically generated

Figure 25: Announcement Page

## Topic 6: Designing a built-in text-editor

This week I made a small addition to the program, which was a text editor page. The idea for this page was to provide the user with a way in which they can make notes for their day-to-day activities or notes for their modules. The user can open previously made text documents or create new ones and save them to a folder of their choosing. As with all my pages in this program the original idea of creating an application which would contain everything the student would need in one place was still the goal and creating notes for study is something a student makes daily speaking from experience.

A picture containing shape

Description automatically generated

Figure 26: Text-editor Page

## Topic 7: Re-designing GUI

Graphical user interface, application

Description automatically generatedOnce I had all the previous pages completed from preceding weeks the final stage was to clean up the code and merge all the pages together. Some of the initial designs for pages such as the calculating of grades was overly simplistic and had no connection to the other pages. Changes to the design were made were needed as shown below with the original grades page versus the updated one, as exhibited we can see the introduction of a unique colour scheme as well as images to the page to make it distinct from the others, buttons linking to the other pages in the program, and most notably multiple text boxes to fill in the grades for each module as opposed to a new window opening for each module resulting in a cleaner design.

A picture containing timeline

Description automatically generated

Figure 27: Old design for grades page

Figure 28: Updated design for grades page

As shown with the grades page, each page now has several buttons linking to all pages in this program, this made the program more accessible and useable as the user will no longer have to restart the application to access other areas of the program.

# Chapter 5 - Evaluation

## Software testing

[12] Software testing is the process of identifying the correctness of the software in question by considering all its attributes (Reliability, scalability, portability, re-usability, usability) and also evaluating the execution of different software components to find any software errors. Software testing involves testing each component under the required services in order to confirm whether or not it satisfies the specified requirements. The testing aspect of coding is of upmost importance when attempting to create a useable error free program. Testing is a group of techniques comprised to determine the correctness of the application. The main intention of testing is to identify any failures of the application so that these failures can be discovered and corrected. It does not determine whether or not a product functions properly under all conditions but only that it is not working in some specific conditions. [12] There are multiple types of testing available as displayed below (Figure 29):

Text

Description automatically generated with medium confidence

Figure 29: Types of software testing

## Functional and non-functional

As shown in figure 29, black box testing can be either functional or non-functional.

[16] Functional testing are comprised of short tests, and test whether each function of the software meets the specified requirements. Non-functional testing tests all of the non-functional requirements of a system, which comprise of every test that does not fall under the functional ones.

For this program manual testing was used. Manual testing is the process of checking the functionality of the program without the use of any automation tools. With this method of testing no knowledge of how to use testing tools is required, rather an understanding of the program and how it should or not should not operate is required. There are three types of manual testing currently available, these are white-box, black-box, and grey-box testing.

## Types of testing (White/Black/Grey box)

[13] White Box Testing is a software testing technique in which the internal structure, design, and also the coding of said software is tested in order to verify flow of input-output and to improve design, usability, and security. In white box testing, the code is visible to testers to see, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.

[11] Black box testing is often referred to as functional or behavioural testing and its purpose is to demonstrate functions are operational. The black box testing technique used was boundary value analysis. It is referred to as black box as the code is not visible for testers to see whilst conducting tests.

[14] Finally, grey box testing is a combination of both white and black box testing. The purpose of grey box testing is to identify the defects due to improper code structure or due to improper use of applications.

Having discussed the three types of testing the method of testing that was used for this program was black box testing.

[15] The black box testing technique that is demonstrated in the below tables is known as Cause and effect’. This technique highlights the relationship between a given result and all the factors that affect the result. It is used to write dynamic test cases. The dynamic test cases are used when code works dynamically based on the users input. For example, while entering grades, on entering valid numerical grades within correct range, the system accepts it but, when you enter a non-numerical input or do not fill in all fields, it throws an error message. In this technique, the input conditions are assigned with causes and the result of these input conditions with effects. The main aim of this technique is to reduce the number of test cases while still covering all necessary test cases with maximum coverage to achieve the desired application quality.

## Testing for Inbox.py

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** |
| **1** | User presses “Appointments” button | The Inbox page will be closed, and the Appointments page will be opened in full screen. | The Appointments page was opened in full screen whilst the Inbox page was automatically closed. |
| **2** | User presses “Room Locations” button | The Inbox page will be closed, and the Room Locations page will be opened in full screen. | The Room Locations page was opened in full screen whilst the Inbox page was automatically closed. |
| **3** | User presses “Inbox” button | The Inbox page will be closed, and the Inbox page will be re-opened in full screen. | The Inbox page was re-opened in full screen whilst closing the original Inbox page. |
| **4** | User presses “Weather” button | The Inbox page will be closed, and the Weather page will be opened in full screen. | The Weather page was opened in full screen whilst the Inbox page was automatically closed. |
| **5** | User presses “Announcements” button | The Inbox page will be closed, and the Announcements page will be opened in full screen. | The Announcements page was opened in full screen whilst the Inbox page was automatically closed. |
| **6** | User presses “Grades” button | The Inbox page will be closed, and the Grades page will be opened in full screen. | The Grades page was opened in full screen whilst the Inbox page was automatically closed. |
| **7** | User presses “Text editor” button | The Inbox page will be closed, and the Text editor page will be opened in full screen. | The Text editor page was opened in full screen whilst the Inbox page was automatically closed. |
| **8** | User presses “EXIT” button | The Inbox page will be closed, and the program will be shutdown | The Inbox page was closed, and the program shutdown |
| **9** | User moves the scrollbar up/down | The user will be able to scroll through a list containing their subjects of each email in their inbox | The scrollbar allowed the user to navigate through the list of email subjects in their inbox. |

## Testing for Appointments.py

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** |
| **1** | User presses “Appointments” button | The Appointments page will be closed, and the Appointments page will be re-opened in full screen. | The Appointments page was re-opened in full screen whilst closing the original Appointments page. |
| **2** | User presses “Room Locations” button | The Appointments page will be closed, and the Room Locations page will be opened in full screen. | The Room Locations page was opened in full screen whilst the Appointments page was automatically closed. |
| **3** | User presses “Inbox” button | The Appointments page will be closed, and the Inbox page will be opened in full screen. | The Inbox page was opened in full screen whilst the Appointments page was automatically closed. |
| **4** | User presses “Weather” button | The Appointments page will be closed, and the Weather page will be opened in full screen. | The Weather page was opened in full screen whilst the Appointments page was automatically closed. |
| **5** | User presses “Announcements” button | The Appointments page will be closed, and the Announcements page will be opened in full screen. | The Announcements page was opened in full screen whilst the Appointments page was automatically closed. |
| **6** | User presses “Grades” button | The Appointments page will be closed, and the Grades page will be opened in full screen. | The Grades page was opened in full screen whilst the Appointments page was automatically closed. |
| **7** | User presses “Text editor” button | The Appointments page will be closed, and the Text editor page will be opened in full screen. | The Text editor page was opened in full screen whilst the Appointments page was automatically closed. |
| **8** | User presses “EXIT” button | The Appointments page will be closed, and the program will be shutdown | The Appointments page was closed, and the program shutdown |
| **9** | User moves the scrollbar up/down | The user will be able to scroll through a list containing their appointments | The scrollbar allowed the user to navigate through the list of appointments in their calendar |

## Testing for RoomLocations.py

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** |
| **1** | User presses “Appointments” button | The Room Locations page will be closed, and the Inbox page will be opened in full screen. | The Appointments page was opened in full screen whilst the Room Locations page was automatically closed. |
| **2** | User presses “Room Locations” button | The Room Locations page will be closed, and the Appointments page will be re-opened in full screen. | The Room Locations page was re-opened in full screen whilst closing the original Room Locations page. |
| **3** | User presses “Inbox” button | The Room Locations page will be closed, and the Inbox page will be opened in full screen. | The Inbox page was opened in full screen whilst the Room Locations page was automatically closed. |
| **4** | User presses “Weather” button | The Room Locations page will be closed, and the Weather page will be opened in full screen. | The Weather page was opened in full screen whilst the Room Locations page was automatically closed. |
| **5** | User presses “Announcements” button | The Room Locations page will be closed, and the Announcements page will be opened in full screen. | The Announcements page was opened in full screen whilst the Room Locations page was automatically closed. |
| **6** | User presses “Grades” button | The Room Locations page will be closed, and the Grades page will be opened in full screen. | The Grades page was opened in full screen whilst the Room Locations page was automatically closed. |
| **7** | User presses “Text editor” button | The Room Locations page will be closed, and the Text editor page will be opened in full screen. | The Text editor page was opened in full screen whilst the Room Locations page was automatically closed. |
| **8** | User presses “EXIT” button | The Room Locations page will be closed, and the program will be shutdown | The Room Locations page was closed, and the program shutdown |
| **9** | User moves the scrollbar up/down | The user will be able to scroll through a list containing their Room Locations for classes | The scrollbar allowed the user to navigate through the list of Room Locations in their calendar |
| **11** | The user is signed in but has no internet connection | No information will be displayed | The information previously loaded whilst an internet connection was present are displayed |

## Testing for weatherAPI.py

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** |
| **1** | User presses “Appointments” button | The Weather page will be closed, and the Appointments page will be opened in full screen. | The Appointments page was opened in full screen whilst the Weather page was automatically closed. |
| **2** | User presses “Room Locations” button | The Weather page will be closed, and the Room Locations page will be opened in full screen. | The Room Locations page was opened in full screen whilst the Weather page was automatically closed. |
| **3** | User presses “Inbox” button | The Weather page will be closed, and the Inbox page will be opened in full screen. | The Inbox page was opened in full screen whilst the Weather page was automatically closed. |
| **4** | User presses “Weather” button | The Weather page will be closed, and the Weather page will be re-opened in full screen. | The Weather page was re-opened in full screen whilst closing the original Weather page. |
| **5** | User presses “Announcements” button | The Weather page will be closed, and the Announcements page will be opened in full screen. | The Announcements page was opened in full screen whilst the Weather page was automatically closed. |
| **6** | User presses “Grades” button | The Weather page will be closed, and the Grades page will be opened in full screen. | The Grades page was opened in full screen whilst the Weather page was automatically closed. |
| **7** | User presses “Text editor” button | The Weather page will be closed, and the Text editor page will be opened in full screen. | The Text editor page was opened in full screen whilst the Weather page was automatically closed. |
| **8** | User presses “EXIT” button | The Weather page will be closed, and the program will be shutdown | The Weather page was closed, and the program shutdown |
| **9** | User presses “Get the temp” button with no city entered | An error message will be displayed | An error message stating the following is displayed “City entered was not found, try again” |
| **10** | The user enters a valid city | The weather information for said city will be displayed | The city entered will have its weather information shown |
| **11** | The API doesn’t work | A backup weather system will be used to display information | A backup weather system is opened for the user |

## Testing for googleClassroom.py

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** |
| **1** | User presses “Appointments” button | The Announcements page will be closed, and the Appointments page will be opened in full screen. | The Appointments page was opened in full screen whilst the Announcements page was automatically closed. |
| **2** | User presses “Room Locations” button | The Announcements page will be closed, and the Room Locations page will be opened in full screen. | The Room Locations page was opened in full screen whilst the Announcements page was automatically closed. |
| **3** | User presses “Inbox” button | The Announcements page will be closed, and the Inbox page will be opened in full screen. | The Inbox page was opened in full screen whilst the Announcements page was automatically closed. |
| **4** | User presses “Weather” button | The Announcements page will be closed, and the Weather page will be opened in full screen. | The Weather page was opened in full screen whilst the Announcements page was automatically closed. |
| **5** | User presses “Announcements” button | The Announcements page will be closed, and the Announcements page will be re-opened in full screen. | The Announcements page was re-opened in full screen whilst closing the original Announcements page. |
| **6** | User presses “Grades” button | The Announcements page will be closed, and the Grades page will be opened in full screen. | The Grades page was opened in full screen whilst the Announcements page was automatically closed. |
| **7** | User presses “Text editor” button | The Announcements page will be closed, and the Text editor page will be opened in full screen. | The Text editor page was opened in full screen whilst the Announcements page was automatically closed. |
| **8** | User presses “EXIT” button | The Announcements page will be closed, and the program will be shutdown | The Announcements page was closed, and the program shutdown |
| **9** | User presses “Get announcements” button with no module selected from the drop-down list | An error message will be displayed | An error message stating the following is displayed “Please select a module from the drop-down menu” |
| **10** | The user selects a module | The announcement for said module will be displayed | The announcement made for the selected module is shown |

## Testing for Grades.py

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** |
| **1** | User presses “Appointments” button | The Grades page will be closed, and the Appointments page will be opened in full screen. | The Appointments page was opened in full screen whilst the Grades page was automatically closed. |
| **2** | User presses “Room Locations” button | The Grades page will be closed, and the Room Locations page will be opened in full screen. | The Room Locations page was opened in full screen whilst the Grades page was automatically closed. |
| **3** | User presses “Inbox” button | The Grades page will be closed, and the Inbox page will be opened in full screen. | The Inbox page was opened in full screen whilst the Grades page was automatically closed. |
| **4** | User presses “Weather” button | The Grades page will be closed, and the Weather page will be opened in full screen. | The Weather page was opened in full screen whilst the Grades page was automatically closed. |
| **5** | User presses “Announcements” button | The Grades page will be closed, and the Announcements page will be opened in full screen. | The Announcements page was opened in full screen whilst the Grades page was automatically closed. |
| **6** | User presses “Grades” button | The Grades page will be closed, and the Grades page will be re-opened in full screen. | The Grades page was re-opened in full screen whilst closing the original Grades page. |
| **7** | User presses “Text editor” button | The Grades page will be closed, and the Text editor page will be opened in full screen. | The Text editor page was opened in full screen whilst the Grades page was automatically closed. |
| **8** | User presses “EXIT” button | The Grades page will be closed, and the program will be shutdown | The Grades page was closed, and the program shutdown |
| **9** | User presses “Get the grade” button without entering numeric values into each field or does not enter grades for all fields | An error message will be displayed | An error message stating the following is displayed “Please ensure all fields have been filled in and are numerical values” |
| **10** | The user enters a number NOT between the range of 0-100 for each field | An error message will be displayed | An error message stating the following is displayed “Please ensure all grades are between 0-100” |
| **11** | The user enters a number between 0-100 for each field | A pie-chart will be generated | A pie-chart showing a visual representation of the grades is automatically generated and opened for the user |

## Testing for textedit.py

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST ID** | **DESCRIPTION** | **EXPECTED RESULT** | **ACTUAL RESULT** |
| **1** | User presses “Appointments” button | The Text editor page will be closed, and the Appointments page will be opened in full screen. | The Appointments page was opened in full screen whilst the Text editor page was automatically closed. |
| **2** | User presses “Room Locations” button | The Text editor page will be closed, and the Room Locations page will be opened in full screen. | The Room Locations page was opened in full screen whilst the Text editor page was automatically closed. |
| **3** | User presses “Inbox” button | The Text editor page will be closed, and the Inbox page will be opened in full screen. | The Inbox page was opened in full screen whilst the Text editor page was automatically closed. |
| **4** | User presses “Weather” button | The Text editor page will be closed, and the Weather page will be opened in full screen. | The Weather page was opened in full screen whilst the Text editor page was automatically closed. |
| **5** | User presses “Announcements” button | The Text editor page will be closed, and the Announcements page will be opened in full screen. | The Announcements page was opened in full screen whilst the Text editor page was automatically closed. |
| **6** | User presses “Grades” button | The Text editor page will be closed, and the Announcements page will be opened in full screen. | The Announcements page was opened in full screen whilst the Text editor page was automatically closed. |
| **7** | User presses “Text editor” button | The Text editor page will be closed, and the Text editor page will be re-opened in full screen. | The Text editor page was re-opened in full screen whilst closing the original Text editor page. |
| **8** | User presses “EXIT” button | The Text editor page will be closed, and the program will be shutdown | The Text editor page was closed, and the program shutdown |
| **9** | User presses “Saves as” button | The user will be asked where they want to save their file | The user can name and save the file to their specifications |
| **10** | User presses “Open” button | The user will be able to open text files | The user can select a text file and it will open in the editor |

## GitHub

All of the code displayed and mentioned in this report have been uploaded to a GitHub repository. The link to the GitHub repository can be found [here](https://github.com/Glen2000/StudentPlanner)

# Chapter 6 – Conclusions & Future Work

In conclusion the program worked as initially intended with slight modifications having to made to either suit the timeframe of the project, or to improve on the initial idea. The open aspect of this project allowed for addition and removal of ideas to fine tune the finished product, which was something I really enjoyed as each time I worked on the program I found myself thinking of new ideas to incorporate. There were many aspects to this program which I had no previous experience with such as using APIs and also building a GUI in python, which both turned out to be the topics I found most interesting and intriguing to work on. The Google Classroom API in particular piqued my interest, accessing information from a website which could not be web scraped like I had previously done for other projects required me to explore alternatives which resulted in using the Google classroom API. However, the topic I found most enticing was the GUI through use of the Tkinter library. Previously to date, all programs I had created over the past four years had been displayed in a command line with little to no interaction, perhaps the odd user input. But with Tkinter I was finally able to create a program which was interactive and designed in a cohesive way which always allowed the user to know what page they were on and what functions were available to them. There was a lot to be learned from creating this program and it really opened my eyes to how powerful python can be and that there seems to always be another method to achieving your goal even if it was not the intended one. However, there are some things that I would do differently if I had the chance to create this application a second time. First of which is to leave the GUI until the end, to a certain extent. What I mean by this is not that no GUI should be made or tested, as since it was my first-time using GUI in python, I needed to experiment with it and see what was possible, however I feel as though I became engulfed in trying to create each page as perfectly crafted and well designed as I could as opposed to focusing on the code first, as without the code in working order the GUI is in essence an empty shell. So, a second time around I would only do what I needed within the GUI to make sure the actual code was working, and then towards the end of completion of the project revisit the GUI. This approach is actually something I took on board towards the end of my project, an example of this was with the grades page which didn’t receive an update to its GUI until I had the code fully working.

The second thing I would do differently is actually a change to the program itself and it would be to create a homepage which would give the user a landing page, as opposed to starting the program on their inbox and navigating from there. The third thing I would do differently a second time around would be to follow a more structured approach to my coding, what I mean by this is I found myself jumping from one topic to another when with a little bit of planning I could have stayed on the same general topic. An example of this comes with the APIs, after finishing the weather API I started working on the outlook pages again, when in retrospect it would have been a more logical approach to continue onto the Google classroom API whilst the newfound knowledge of APIs was fresh in my mind.

The final alteration I would make to my approach if I had to start this project again, would be to make the program more accessible to other system. Currently the program obtains its emails, classroom locations and other information through use of the Win32 library as discussed, however this is a Microsoft windows only library, meaning that other software such as Macintosh or Linux will not be able to use it, which is a huge limitation to this program that I would like to fix.

# References

[1] - Moffitt, C., 2022. Automating Windows Applications Using COM - Practical Business Python. [online] Pbpython.com. Available at: <https://pbpython.com/windows-com.html> [Accessed 20 March 2022].

[2] - Docs.microsoft.com. 2022. OlDefaultFolders enumeration (Outlook). [online] Available at: <https://docs.microsoft.com/en-us/office/vba/api/outlook.oldefaultfolders> [Accessed 20 March 2022].

[3] - Python, R., 2022. PySimpleGUI: The Simple Way to Create a GUI With Python – Real Python. [online] Realpython.com. Available at: <https://realpython.com/pysimplegui-python/> [Accessed 20 March 2022].

[4] - Kumar, B., 2022. How To Go To Next Page In Python Tkinter Program - Python Guides. [online] Python Guides. Available at: <https://pythonguides.com/go-to-next-page-in-python-tkinter/> [Accessed 20 March 2022].

[5] - Python, R., 2022. Beautiful Soup: Build a Web Scraper With Python – Real Python. [online] Realpython.com. Available at: <https://realpython.com/beautiful-soup-web-scraper-python/> [Accessed 20 March 2022].

[6] - Dataquest.io. 2022. [online] Available at: <https://www.dataquest.io/blog/python-api-tutorial/> [Accessed 20 March 2022].

[7] – RTE.ie. 2022. RTÉ Weather - Forecast for Dublin (Co. Dublin). [online] Available at: <https://www.rte.ie/weather/22259-dublin/> [Accessed 20 March 2022].

[8] - Developer.accuweather.com. 2022. AccuWeather APIs | home. [online] Available at: <https://developer.accuweather.com/?gclid=CjwKCAiA4KaRBhBdEiwAZi1zzmvTfzvmET2Jd5P-U4iNVyAU599fBq9PQ-7-AvssATtoDWOOCdpqUBoCxLcQAvD\_BwE> [Accessed 20 March 2022].

[9] - W3schools.com. 2022. Python RegEx. [online] Available at: <https://www.w3schools.com/python/python\_regex.asp> [Accessed 21 March 2022].

[10] - Docs.microsoft.com. 2022. The Component Object Model - Win32 apps. [online] Available at: <https://docs.microsoft.com/en-us/windows/win32/com/the-component-object-model> [Accessed 21 March 2022].

[11] - TestResults.io - Get confidence in your software. Test Automation With Swiss Precision. 2022. What is black box testing and how to use it in functional tests ⋆ TestResults.io - Get confidence in your software. Test Automation With Swiss Precision. [online] Available at: <https://www.testresults.io/blog/what-is-black-box-testing-and-how-to-use-it-in-functional-tests/> [Accessed 19 May 2022].

[12] – www.javatpoint.com. 2022. Learn Software Testing Tutorial - javatpoint. [online] Available at: <https://www.javatpoint.com/software-testing-tutorial> [Accessed 19 May 2022].

[13] - Guru99. 2022. What is WHITE Box Testing? Techniques, Example & Types. [online] Available at: <https://www.guru99.com/white-box-testing.html#1> [Accessed 19 May 2022].

[14] – Guru99. 2022. What is Grey Box Testing? Techniques, Example. [online] Available at: <https://www.guru99.com/grey-box-testing.html> [Accessed 19 May 2022].

[15] - www.javatpoint.com. 2022. Cause-Effect Graph Technique in Black Box Testing - javatpoint. [online] Available at: <https://www.javatpoint.com/cause-and-effect-graph-technique-in-black-box-testing> [Accessed 19 May 2022].

[16] - Blackbox and whitebox types, techniques and differences. 2022. Blackbox and whitebox types, techniques and differences. [online] Available at: <https://buglance.com/blog/black-box-vs-white-box-software-testing> [Accessed 19 May 2022].

[17] - Okta Developer. 2022. What the Heck is OAuth?. [online] Available at: <https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth> [Accessed 19 May 2022].

[18] - Oauth.net. 2022. OAuth 2.0 — OAuth. [online] Available at: <https://oauth.net/2/> [Accessed 19 May 2022].

[19] - Google Cloud. 2022. Authentication overview | Google Cloud. [online] Available at: <https://cloud.google.com/docs/authentication#:~:text=An%20OAuth%202.0%20client%20identifies,Environment%2Dprovided%20service%20account> [Accessed 19 May 2022].

[20] - Nielsen Norman Group. 2022. 10 Usability Heuristics for User Interface Design. [online] Available at: <https://www.nngroup.com/articles/ten-usability-heuristics/> [Accessed 19 May 2022].

[21] - www.javatpoint.com. 2022. Agile Model (Software Engineering) - javatpoint. [online] Available at: <https://www.javatpoint.com/software-engineering-agile-model> [Accessed 19 May 2022].