Complex Processes

To-do List

Victor Qiu

Mount Albert Grammar

This dissertation is submitted for
91907

Digital Technology **1/7/2021**

Contents

List of Figures

1 Project Management

During the pursuit to develop our digital outcome, it was evident that in order to create an application that was both robust and flexible - within the given time; a plan was a definite prerequisite. Throughout this process, a visual diary, as well as the use of version control software were used to keep track, plan and develop new ideas which would go towards improving our program.

1.1 Visual Diary

A visual diary was used and altered during the process of developing our digital outcome. This was done so to help aid in the process of what features our program should possess. As a simple offline solution; using a visual diary was an elegant and non-convuluted solution to aid in our planning process. This proved to be useful as it allowed us to map out in a visual manner how to approach the program in the upcoming weeks. The initial features which I had planned for, were put under the the "to-do list" title, whilst newly suggested features which were later implemented were added under the "additional" title. The specifications for the features have been identified within the second column, while the minute details of those specifications have been listed in the third column **see figure 1**.

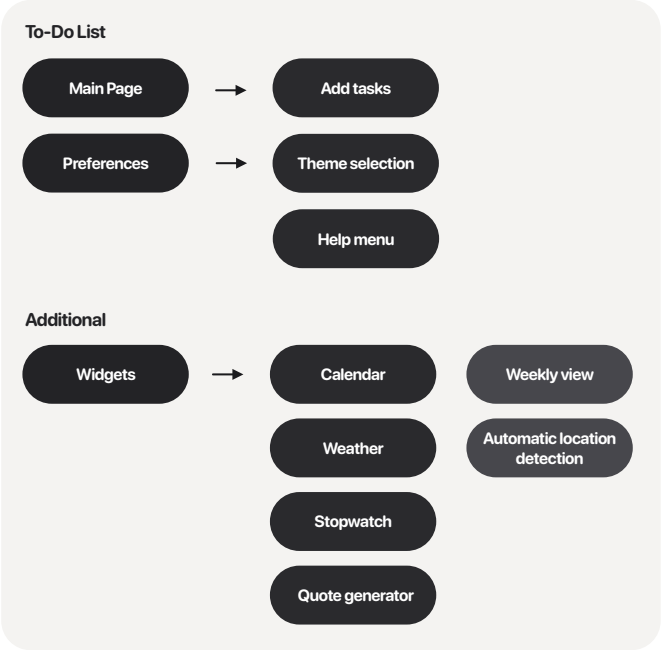


Figure 1: visual diary

1.2 Version Control

Github Link

Another tool which was used was Github **see figure 2**. As a code hosting platform, this particular site allowed us to keep track of any changes made in our application. Having access to this utility was quite beneficial as it is able to provide a timeline of all commits, making it easy to see how much progress has been made, and how much still needs to be done. Alongside this, Github also seamlessly integrates with Pycharm (our IDE); making it easier to use and operate. Another feature of Github which proved to be useful is that all changes which have been made are kept track between commits. This means that reversions, or accidental deletions, could be altered and fixed far quicker then if I had to go through a plethora of undos and redos. Lastly, in case of any crashes; or if the particular laptop that this program

was developed on were to ever break, having it saved on Github would mean it would still be accessible in case of such an event.

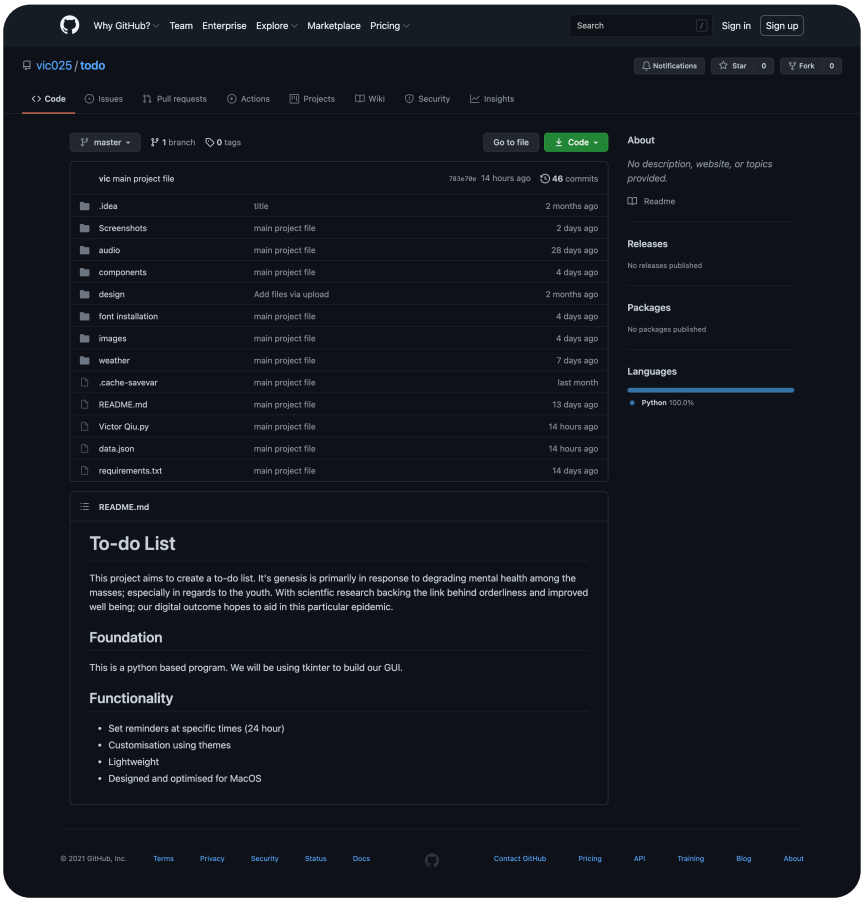


Figure 2: github

1.3 Feedback

Google Form Link

Feedback was received through a variety of different ways. Methods included verbal feedback among peers, as well as through a Google Form. During our development process, using the feedback collected, alterations were made within our code to act upon the constructive criticism. Questions which were often asked during the feedback process involved whether the application crashed and how they crashed. This will be discussed later on within the report.

These project management tools were chosen so that there would be a mixture of both online and offline tools. As this project is being developed by myself only, a visual diary was used as it is offline, and thus accessible whenever I need it, regardless of internet access - hence any new ideas or adjustments that need to be made were able to be done so immediately. Therefore, applications such as Trello was not used - simply due to its online features not being beneficial in our specific scenario. That being said, in regards to saving and recording our work, Github was used as having our work being saved online would mean that any device would be able to access it, thus making it easier to share for general accessibility by others, plus keeping track of how files have changed over time. As such, purely saving files offline was not an viable option in my opinion - and thus was not a method that was used as a collaborative / backup tool. This is because the online advantages here offset the cons by a great margin. Lastly, as for feedback, Google Forms was used. This is because using an online tool makes it much easier to retrieve feedback from others; allowing for timely changes to be conducted in an orderly manner. Therefore,

offline methods such as printing physical copies of surveys was not used. Lastly, all tools mentioned above are free software, meaning legally; we are in the clear, and that financially; we haven't been set back. As for the primary methodology which was conducted; we used the agile method. We chose this over the waterfall model as it follows a convention which allows us to periodically check whether the development of our application is what we had envisioned. This allowed us to make changes as we went along the trialling and testing phases; which wouldn't have been possible if we had stuck to the latter.

Figure 3: pmi chart

The methods discussed regarding project management tools I believe have been both appropriate and useful. They allowed us to keep track, and structure our workflow during the development of our digital outcome; setting clear goals so that we could meet our deadline. This resulted in a final project that was feature rich in comparison to what was initially planned. Alongside this, due to these project management tools, bugs were ironed out quicker as any particular changes which had been made were recorded, allowing us to backtrack accordingly.

2 Components

To create our program, evidently, breaking it down to its very core components was a definite prerequisite if we wanted to proceed. Considering the large scale of the project, by the end of the development process, we settled on 3 individual components. These components were then elaborated upon, introducing 9 minor components.

2.1 Program Outline

Below is a diagram of the full outcome:

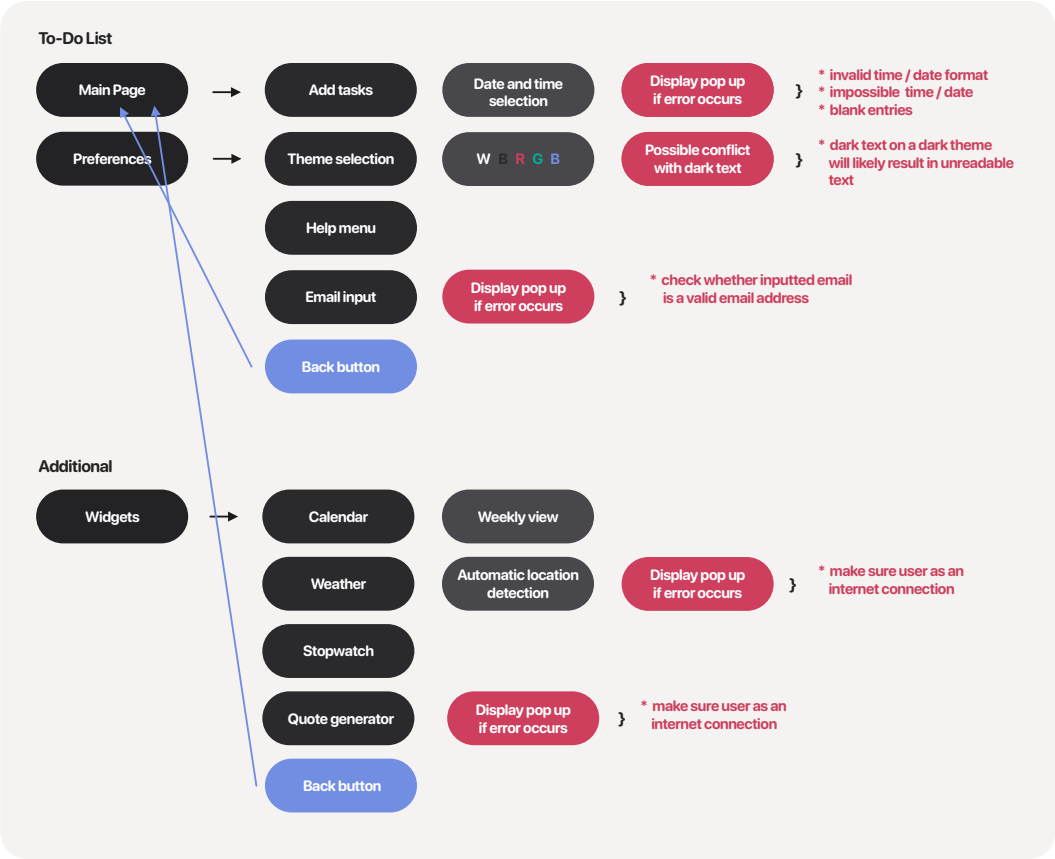


Figure 4: outline

The 'To-Do List' section of our outline was created at the very beginning of developing our programme. This covered the bare essentials of what a To-Do List needs to include, plus a few minor cosmetic features. As such, this proved to be a good starting foundation for our application.

The 'Additional' section within our outline was added on June 25th. Because our application was quite simple and lacked features to be a cohesive program; the concept for the application was then altered to be a general purpose tool. As such; this allowed us to add features such as a calendar, weather, stopwatch and quote generator to create a more capable program. This meant that users would be able to accomplish and do more things with the application compared to before.

2.2 Minimum Viable Product

Below is a diagram of the minimum viable product:

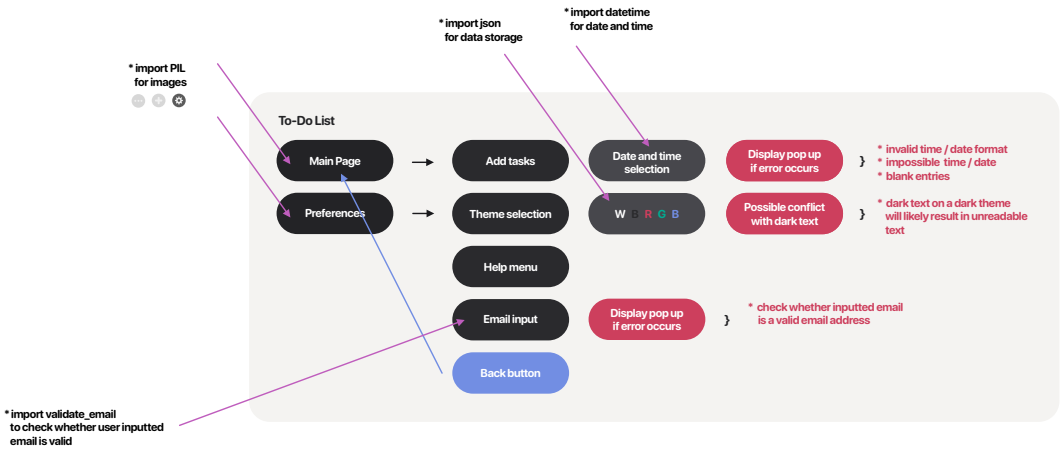


Figure 5: minimum viable product

2.3 Component List

The purpose of def statements are so they can be reused within the program, thus allowing us to link different parts of the program together (returns particular values so that different parts of the program can communicate to each other). In doing so, this creates a more unified experience, making the final application more versatile. Whilst this may be the case, it also tidies up the code of the program, making debugging an easier process.

Components

Below are 3 "pages". These are where all the features are housed. Using a system like this means it will be easier for the user to navigate the program; and access settings related to the page they're on. This means that all the features and settings won't be crammed into one page, but rather through a series of pages that can be accessed through buttons scattered around in the application. This was made possible by using def statements.

Main Page

The main page is the front page of the application. It acts as a hub for what the program can do, as well as houses the key features of a to-do list. Within this section, the user can navigate to either the settings or widgets page, or utilise the features listed below.

- **Add Tasks**

  The 'add tasks' portion of the application will allow the user to select a time and date to which the added task will adhere to. The button used will be round and larger than other buttons used within the program - this is to make it clear to the user of its function. This particular component will use tkinter's native widgets as well as a circular button that will be designed within a program called 'Sketch'. The 'add tasks' button must be significant and obvious to the user. Given that it is the centerpiece of the application, it is important that this button must adhere to design principles that are inviting and clear. In addition, the added task will be saved to a database.

  Complexity: Difficult
  This component was relatively difficult to create. This is because it required knowledge to append to a JSON file as well as a listbox. Alongside this, having to append the data (task name, time and date) in a way which is easily retrievable so that email's can be later sent at the set time - required quite a bit of work.

- **Delete Tasks**

As for this component; the 'delete task' feature will allow the user to remove priorly entered tasks. The button will be created from tkinter's native widget, and will have the words 'delete' engraved upon it, making it clear to the user what it's intended use case is.

Complexity: Simple
This particular component was simpler to make after having created the ability to add tasks. As such, the process of developing this component simply was the reverse as the one above, thus making this section not as challenging.

Settings Page

The settings page acts as a menu to alter and tweak certain elements of the program.

- **Theme Selection**

  The theme selection menu will allow the user to select the background colour of the program. These will be 5 individual buttons each representing 5 distinct colours. The colours that have been chosen are white, black, red, green and blue. These buttons will be created using tkinter's native widgets. It must be obvious to the user the selected colour is the one which they have in their mind. The colours red, green and blue have been chosen due to RGB being the staple behind how LED's work in a display. In addition, the chosen colour will be saved to a database.

  Complexity: Simple
  Changing the background colour of the GUI was a simple task as all it did was require a hex code. This is because tkinter's widgets are set so that this parameter can be altered with ease.

- **Email Input**

  This component will allow the user to input an email. In doing so, it should check whether the email is valid or not. If successful, the email inputted will receive the todo's output by the program. This component will use tkinter's entrybox widget. In addition, the email will be saved to a database.

  Complexity: Simple
  Being able to import python packages (validate_email) which can interpret whether an email was valid or not meant that the email validation process could be accomplished without much effort. As for appending it to the JSON; we could use our knowledge gained from adding and deleting tasks, making this component quite simple to create.

- **Help Menu**

  It's intended purpose is to be a tool the user can use if they are confused as to how the program is to be used. This is designed to be used as a last resort. These buttons will be created using tkinter's native widgets. The help menus will be displayed using tkinter's messagebox. That being said, the instructions must be clear and concise so as not to confuse the user further.

  Complexity: Simple
  Given that this feature can be created using tkinter's messagebox; because it is a preinstalled tool which can be repurposed, creating the help menu was quite simple as all it required was simply writing the instructions.

Widget's Page (Added on June 25th)

The widgets page houses all the additional components of the program. It acts as a place where the user can receive additional information based on their locale.

- **Weather**

  To give insight as to what the weather is like for the day. This like the calendar may assist in selecting an appropriate time to set a to-do. The widget will show the current weather status plus the temperature. An icon will also be displayed. This section will be created using a weather API in addition to tkinter's geometry widget. Pillow will also be used to display an icon.

  Complexity: Difficult
  Although utilising an API to figure out what the weather conditions are isn't necessarily difficult, establishing whether there is an internet connection or displaying images based on what the weather condition is; which proved to be quite challenging.

- **Calendar**

  To display a weekly view to the user. This may assist in selecting an appropriate time to set a to-do. The idea is that it will take the appearance of a rectangular box, with small individual boxes within it - containing the dates.

  Complexity: Difficult
  Using the geometry features of tkinter meant that some mathematical calculations needed to be conducted so that the GUI would look up to par. This increased the difficulty of this component by quite a bit.

- **Stopwatch**

  The purpose of this widget is to act as a timer to do certain tasks. The user for example can time how long they study for, and improve upon it if need be. Icons for the stopwatch will be made using 'Sketch'. These icons will be for a pause, start and reset button. That said, the stopwatch will be in a hh:mm:ss format.

  Complexity: Simple
  Due to the simplicity of a stopwatch program, this component was quite effortless to create.

- **Quotes**

  Its primary goal is to give motivation to the user using the application. This particular button will display a random inspirational quote from an API. This section will be created using tkinter's native widgets.

  Complexity: Difficult
  This particular component utilises an API to fetch quotes. Although displaying the fetched quotes wasn't too much of a challenge, making sure it met the bounds of the program so that the text didn't go off the window was a concern.

Back Button

A universal back button will make navigation a much more pleasant experience when accessing different pages of the program. By using a def statement, this back button feature will be able to be reused, allowing the user to backtrack to a prior page from the one they're currently using.

Complexity: Simple A universal back button was quite pleasant to create as all it needed to do was to go back a page. As such, it didn't require too much work.

Database (JSON)

The database will take the form of a JSON file. It will store the users settings as well as recorded tasks so that upon reopening the application, it doesn't reset itself. Also, It being contained as a def statement will mean whenever we need to read or write to it, we can simply refer to it rather than having to repeatedly state it over and over again.

Complexity: Difficult

The inherent nature of creating a database is quite difficult. Due to this - it proved to be a challenge. This is because creating a persistent file storage meant it must be readable and writeable in a format which the program can understand. Because this was the case, figuring this component out required the most amount of tinkering.

2.3.1 Task Sizing

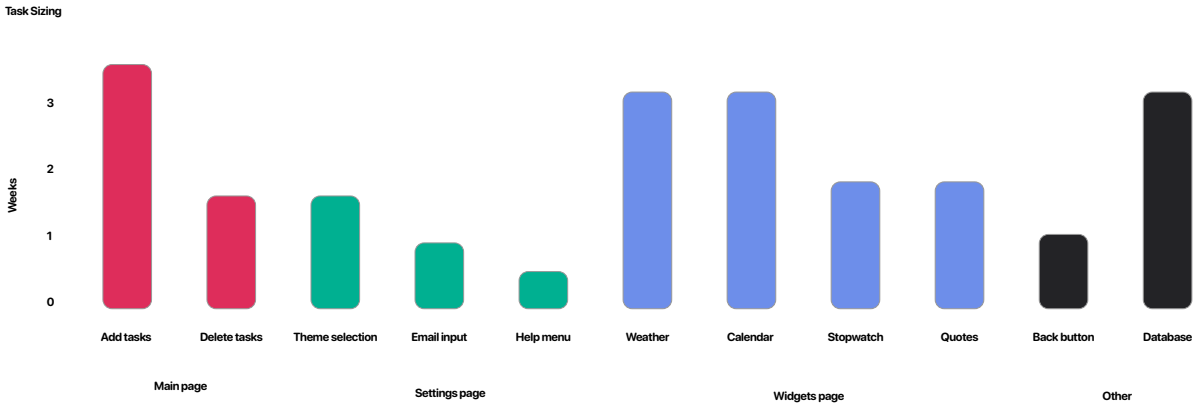
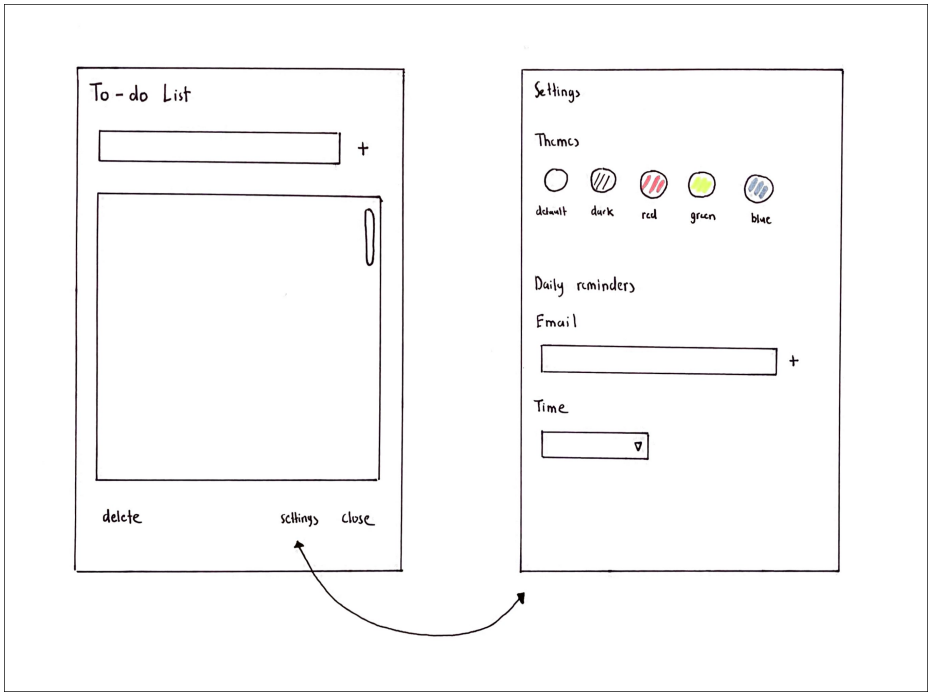


Figure 6: task sizing

3 Develop / Trial Components

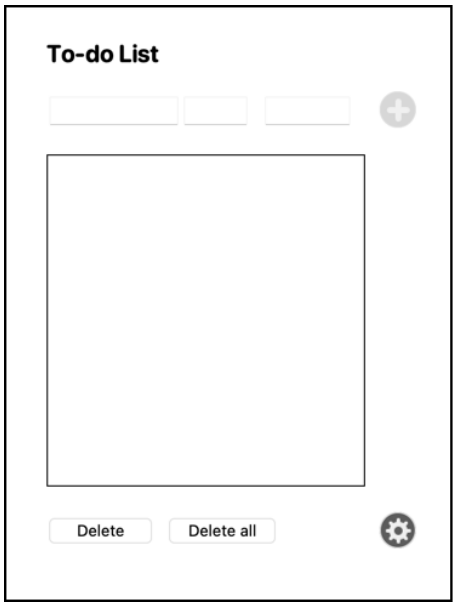
3.1 Main Page

Plan:



Trial:

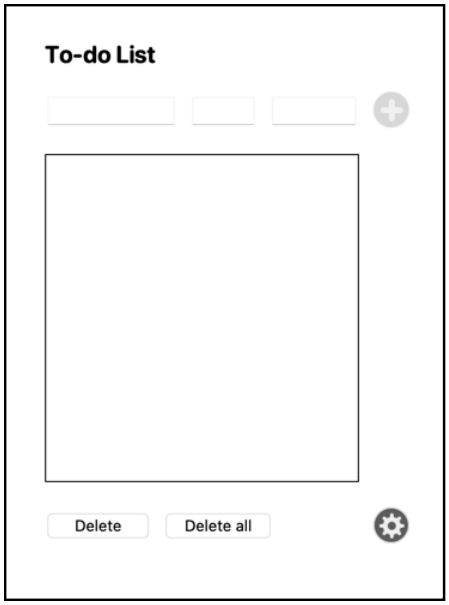
Upon running the program, the goal is to create a skeleton of our final project. This means a perfect UI without the UX; which involved the use of placing text boxes, entry boxes and buttons in the correct location.
However, when running it, this failed to be the case as some of the entry boxes were not evenly spaced. As such, in order to fix this, some further calculations needed to be conducted so that this will no longer be an issue.




```
26  # Inputs (1)
27  # Task name entry
28  new_task_entry = Entry(root, width=11)
29  new_task_entry.place(x=35, y=75)
30  # Time entry
31  new_task_entry_1 = Entry(root, width=5)
32  new_task_entry_1.place(x=146, y=75)
33  # Date entry
34  new_task_entry_2 = Entry(root, width=7)
35  new_task_entry_2.place(x=213, y=75)
36
```

Action:

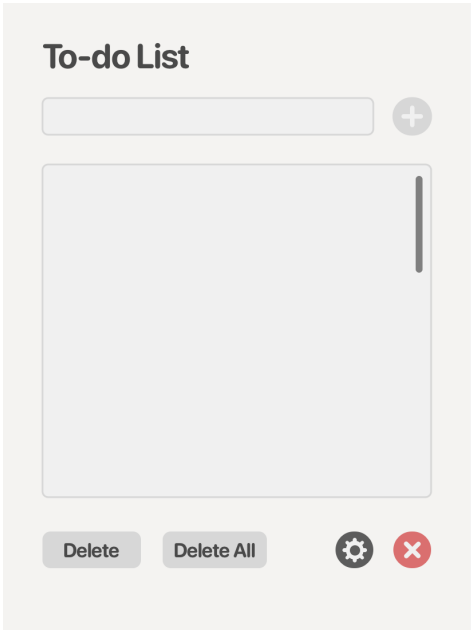
This was fixed through adjusting some of the numbers. The x value for the time entry was altered from x=146 to x=156, whilst the x value for the date entry was altered from x=213 to x=223. In doing so, this evened the spacing between the entry boxes, producing a more uniform UI.

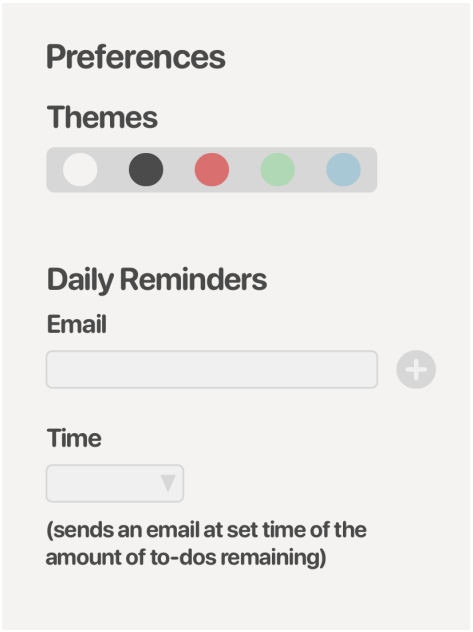



```
26  # Inputs (1)
27  # Task name entry
28  new_task_entry = Entry(root, width=11)
29  new_task_entry.place(x=35, y=75)
30  # Time entry
31  new_task_entry_1 = Entry(root, width=5)
32  new_task_entry_1.place(x=156, y=75)
33  # Date entry
34  new_task_entry_2 = Entry(root, width=7)
35  new_task_entry_2.place(x=223, y=75)
```

3.2 Settings Page

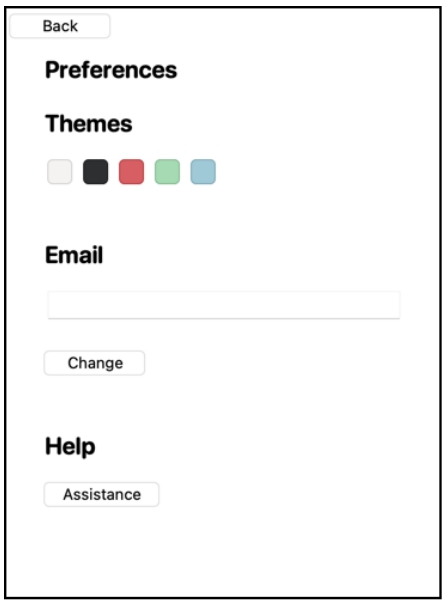
Plan (theme selection buttons):



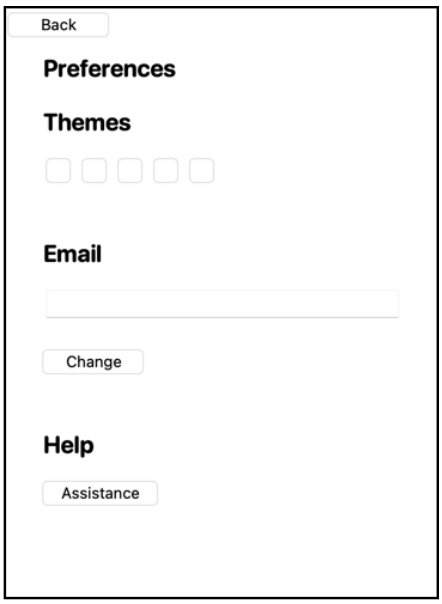


Trial:

The goal here is to make it so we have 2 "pages"; one being our main page, and the other our settings page. By doing so, we want to make it possible for the user to switch back and forth between these windows. This would allow us to set the foundation for how changing windows would work within our program; thus making it easier for the user to navigate throughout the application. However first, we would have to create the settings page.
When ran, although initially everything seemed to work well, upon deselecting the active window, the colours of our theme selection also went away.



When window is selected



When window isn't selected

Action:

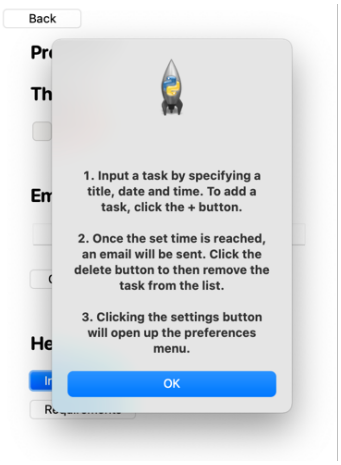
This was something that could not be "fixed" unfortunately. Due to how native MacOS buttons work, this carried the same side affect - meaning because this is simply how buttons work within the operating system, there is litteraly no way to get around it. As such, no actions were needed, however a compromise did have to be made.

Plan (help menu):

Due to the simplicity of this section, a plan was not used.

Trial:

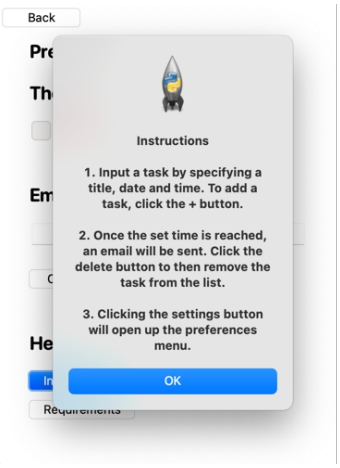
The idea behind this section is to create a pop up which gives the user information about the program. This includes instructions, or a list of requirements.
When ran, the header of our messagebox failed to show. This meant that there was no visible title even though it had been coded in. As seen below, the word "instructions" fails to be mentioned despite being in the code.




```
# Help menu
# Instructions
def instructions_func(self):
    messagebox.showinfo("Instructions",
                        "\n \n 1. Input a task by specifying a title, "
                        "date and time. To add a task, click the + button."
                        "\n \n 2. Once the set time is reached, "
                        "an email will be sent. Click the delete button "
                        "to then remove the task from the list."
                        "\n \n 3. Clicking the settings button will "
                        "open up the preferences menu.")
```

Action:

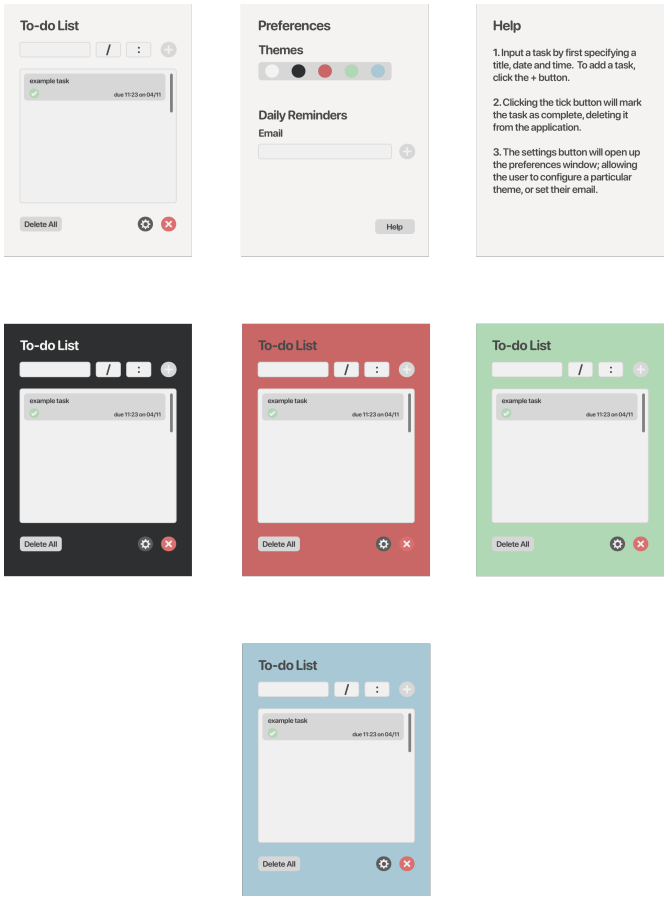
After having done some research, it appeared that this was an issue with tkinter on MacOS. As such, technically there wouldn't be a way to solve this problem. However, we can circumvent this by writing in a placeholder title. This means we will write the title twice, but because of how tkinter works, it will only show once.




```
# Help menu
# Instructions
def instructions_func(self):
    messagebox.showinfo("Instructions",
                        "Instructions "
                        "\n \n 1. Input a task by specifying a title, "
                        "date and time. To add a task, click the + button."
                        "\n \n 2. Once the set time is reached, "
                        "an email will be sent. Click the delete button "
                        "to then remove the task from the list."
                        "\n \n 3. Clicking the settings button will "
                        "open up the preferences menu.")
```

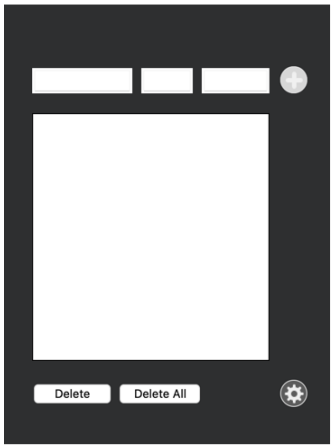
3.3 Database

Plan (theming):



Trial:

This particular section introduces the need for a database as we want to make sure user preference are saved when closing and opening the application.
When trialing, the colours red, green and blue worked without issue. However the colour black didn't work out well. This is because black text on a black background wouldn't show.




```
{"data": [], "email": "v1c.q2500@gmail.com", "bg_colour": "#2E2F30"}
```

```
self.colour_black = Button(self.root, font="15",
                           bg='#2E2F30', fg='#2E2F30', borderless=1,
                           width=25,
                           command=lambda: self.change_bg('#2E2F30'))
```

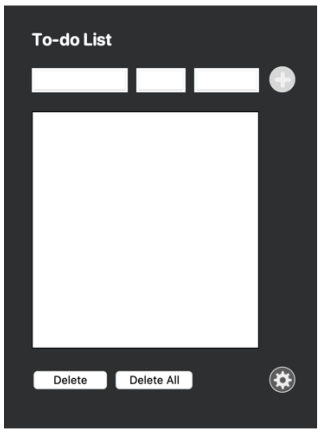
```
# Changes the theme
def change_bg(self, background, foreground):
    # Sets position in JSON file to refer to
    self.data["bg_colour"] = background
    # Writes to JSON file
    with open("data.json", "w") as file:
        json.dump(self.data, file)
    messagebox.showinfo("Done!", "The theme has been altered")
    self.settings()
```

the code here only records
one hex code (background)

Action:

To solve this problem, it is crucial that when selecting a theme, both a background and text colour are defined within the code. This means that when we select a theme, it will allow our program to keep track of what colour the text needs to be in order to be readable. Hence why our database must store both the text and background colour, rather then only the background colour.

```
{"data": [], "email": "v1c.q2500@gmail.com", "bg_colour": "#2E2F30", "text_colour": "white"}
```




```
# Black background, white text
self.colour_black = Button(self.root, font="15",
                           bg='#2E2F30', fg='#2E2F30', borderless=1,
                           width=25,
                           command=lambda: self.change_bg('#2E2F30',
                                                          "white"))
```

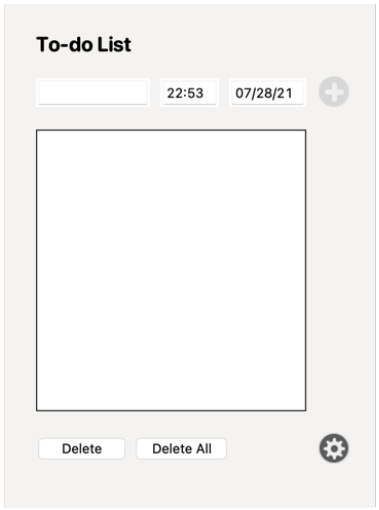
```
# Changes the theme
def change_bg(self, background, foreground):
    # Sets position in JSON file to refer to
    self.data["bg_colour"] = background
    self.data["text_colour"] = foreground
    # Writes to JSON file
    with open("data.json", "w") as file:
        json.dump(self.data, file)
    messagebox.showinfo("Done!", "The theme has been altered")
    self.settings()
```

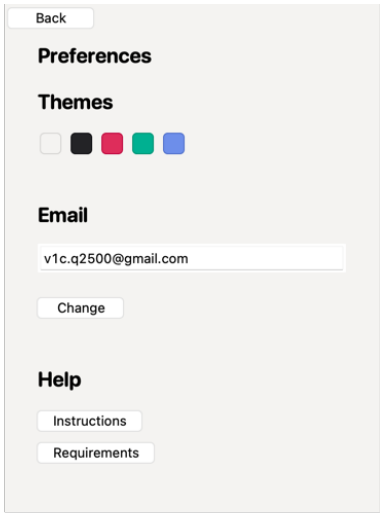
the code here records
two hex codes, one for background the other for text colour

Plan (adding and deleting tasks):

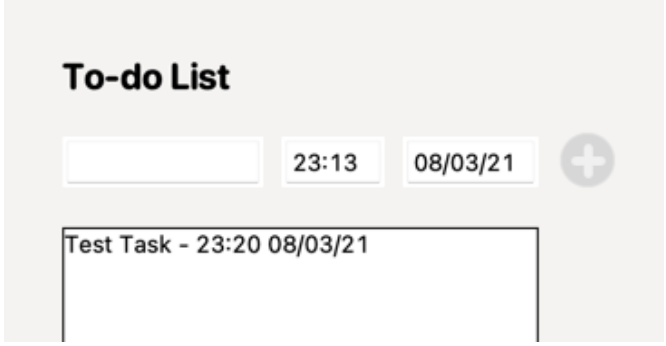
Trial:

Continuing from our database; now we need to make it so the JSON file can accept added tasks. This resulted us in adding a "data" section within our JSON file which the task name, entered date and time would/could be appended to.
Initially, everything appeared to work fine. Tasks would be added, and the JSON file made sure they were persistently stored. However, what we failed to take into account was what could be input as a data and time. This meant that words could be considered a "time", as well as impossible dates could be entered. As such, everytime we tried to read and utilise the recorded data from the JSON, our program would crash.





This is what the JSON file looks like, on the left hand side, there now pertains a data section which records user inputted data



Action:

Currently, the tasks are simply being written to and then appended to the JSON file. As such, there is no error handling. To fix this, we must add certain parameters / checks so that the information which the user inputs can be checked for validity. To do so, we imported the module datetime as well as added a way to identify if any entryboxes were blank. By doing this, we will have checked every logical pathway, making sure there will be no faults with this aspect of the program. As such, if any invalid inputs are entered, an error pop up will instead show.

```
# If the entry task is blank, (\ for syntax purposes)
if self.new_task_entry_var.get() == "" or \
        self.new_task_time_var.get() == "" or \
        self.date_var.get() == "":
    messagebox.showinfo("Invalid Input!",
                        "An input appears to be blank")
```

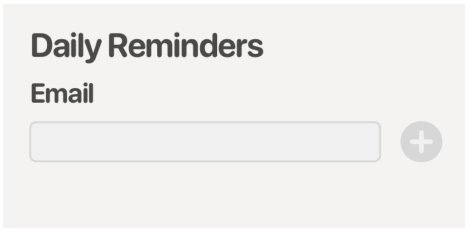
Checks if any of the entryboxes are blank

```
date_old = datetime.datetime(year, day, month, hour, minute)

if datetime.datetime.now() > date_old:
    messagebox.showinfo("Invalid Input!",
                        "Inputted time and/or date is before "
                        "the actual time")
```

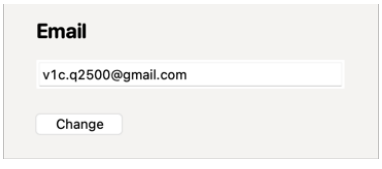
Checks if the time entered is before the actual time

Plan (email input):



Trial:

This particular section allows the program to save an email into its database; will become useful later on when we acquire the functionality to send emails.
When ran however, similar to the prior issue; because emails which are saved aren't checked for validity, when utilising it elsewhere within the program, it may crash as the entered "email" doesn't necessarily have to be an email.



{"data": [["test", "22:59", "07/28/21", "1"]], "email": "v1c.q2500@gmail.com", "bg_colour": "#F4F3F1", "text_colour": "black"}

This is what the JSON file looks like, in the middle, there now pertains a email section

Action:

To solve this particular concern, we imported the verify_email module. This allowed us to automatically

check whether an inputted email was valid or not, thus allowing us to check every possible boundary case without massive amounts of code. As a result, we were able to eliminate another possible way for our program to crash, therefore improving the stability of the program.

```
if validate_email(self.email_entry_var.get()):
```

3.4 Sending Emails

Plan:

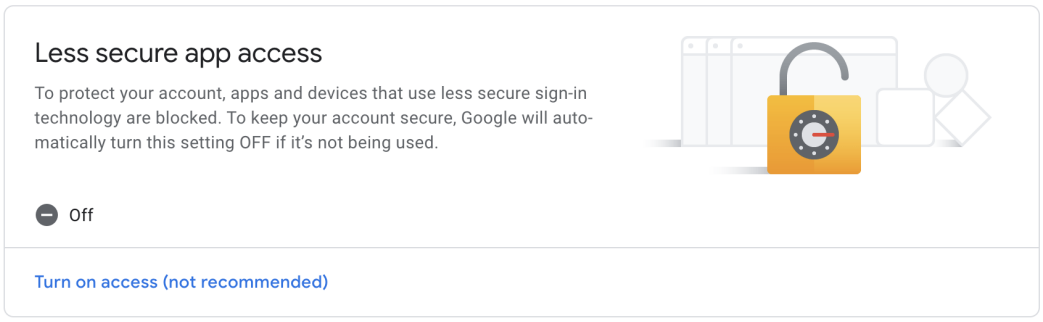
No plan utilised as this feature does not require any special UI elements.

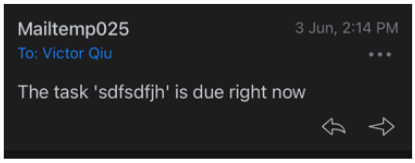
Trial:

This particular component is supposed to allow us to send the user's inputted tasks to the user inputted email.
However, when the program is ran, this failed to be the case as no email was sent.

Action:

After doing some research[1] on the topic, it became apparent that in order to send an email through a third party, the security of the email address used must be lowered. This is because Google has measures in place to prevent botting. As such, we had to delve into the Google settings of the email account which we were going to use to send emails, and toggle on "less secure apps". In doing so, emails were now able to be sent.





Format of the email sent / example of sent email

```
# Email message sent (f-string)
# i[0] refers to the message i[1] refers to th time
# i[2] refers to the date, i[3] refers to the
# indicator
self.message = f"The task '{i[0]}' is due right now"

self.smtp_session.sendmail("mailtemp025@gmail.com",
                           self.data["email"],
                           self.message)
```

```
# SMTP port number: 587
self.smtp_session = smtplib.SMTP('smtp.gmail.com', 587)
self.smtp_session.starttls()
# Mailing address (email and password)
self.smtp_session.login("mailtemp025@gmail.com",
                        "Fluffy15010hi")
```

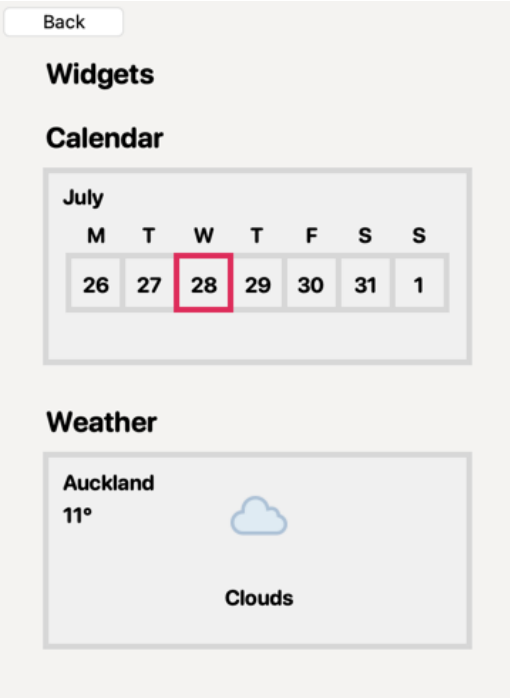
[1]Real Python. (2021, February 27). Sending Emails With Python. Retrieved from https://realpython.com/python-send-email/

3.5 Widgets

Plan:

Trial:

The intention behind this part of the program is to display a weekly calendar view as well as the current weather.
Although for the most part everything did work, spacing for the weather icon was not appropriate. Alongside this, location data was only being sourced from Auckland, and thus wouldn't change if you moved locations. These are concerns which will need to be fixed.



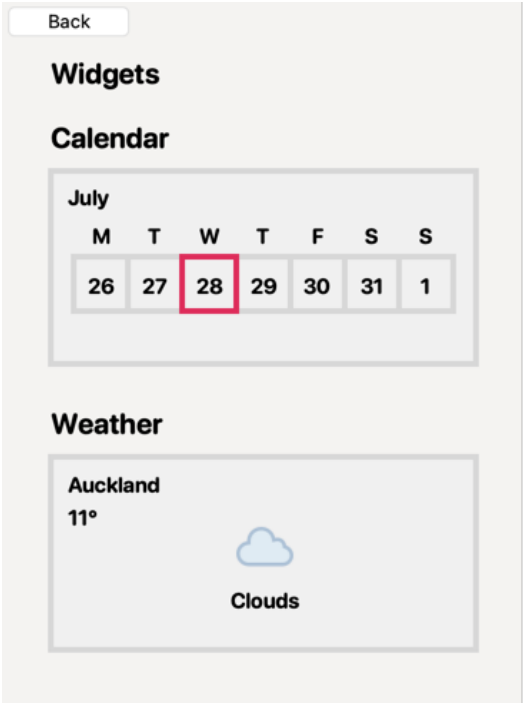

```
# location
self.city = 'Auckland'
```

Action:

To fix the two errors mentioned above, we'd have to do a multitude of different changes. Firstly, as

for the positioning of the icon - this was due to the PNG files being used having a large blank space below them. Thus to fix this issue, all we had to do was crop the image.

Secondly however; in order to automatically detect the user's location, we would need to import an module called geocoder. Thus instead of setting the location within the code, we instead utilise the user's IP to identify their location, thus making this feature usable for more people. Doing it this way is secure as the program is purely reading the IP. Because this is the case, the user's IP will be kept safe as we aren't manipulating it in any other way.




```
# Auto detect location
self.city = geocoder.ip("me").city
```

3.6 Internet Connection

Plan:

No plan utilised as this component doesn't require any UI elements.

Trial:

This component was created due to the program crashing every time it required internet access when it didn't have it. As such, a check needed to be created so that if there wasn't internet access, instead of crashing; a pop up would instead show - letting the user know why a certain component won't be able to function at the current time. This way, errors are well handled, making the program more user friendly. A benefit of this is that all offline features will still remain fully functional.

Action:

To implement such a check, the code below was used. From now on, every time a component requires

```
# Internet detection
def internet_on(self):
    # To check whether there is an internet connection, we'll try to
    # connect to google.com - error handling
    try:
        socket.create_connection(('google.com', 80))
        return True
    except OSError:
        return False
```

internet access, the def function internet_on will run. If there is a connection, the operation will continue successfully, otherwise a pop up will instead show.

4 Develop / Test Components

4.1 Testing Outcome

Before our final stage, our code was reviewed and edited to meet PEP8 standards. Doing so allowed us to organise our code, making it easier to look for and make alterations during the testing process.

4.1.1 Entering Tasks

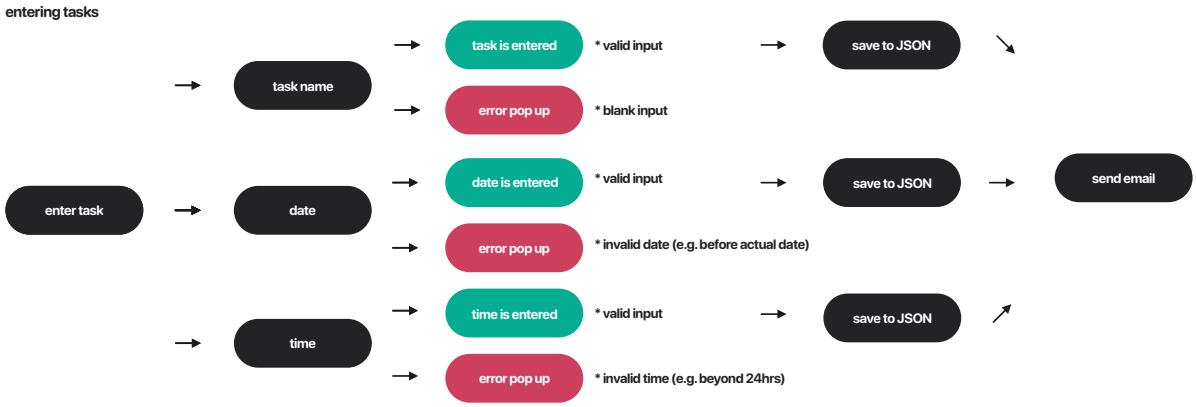


Figure 7: entering tasks

Test Case	Expected Result	Actual Result	What needs to be changed?
Blank inputs	Error pop up	Error pop up	Nothing
Enter impossible time (over 24hrs)	Error pop up	Error pop up	Nothing
Enter impossible date (before actual date)	Error pop up	Error pop up	Nothing
Valid inputs	Adds task to JSON (name, date and time)	Adds task to JSON (name, date and time)	Nothing

User Testing

What did the user do?	Expected Result	Actual Result	What needs to be changed?
Fill in the empty entryboxes	Successfully enters all information correctly with ease	Resulted in confusion surrounding how the program works	Add some sort of marker to indicate to the user what each button and/or entrybox does

Although functionally everything works as intended, due to the ambiguity of some elements within the program; users who tested it stated that they were confused as to what each button did. As such, measures were required to be put in place to help make the program communicate to the user more effectively.

4.1.2 Theme Selection

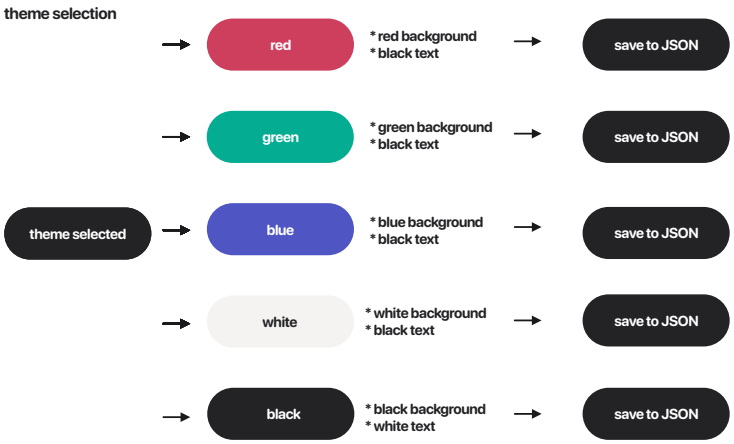


Figure 8: theme selection

Test Case	Expected Result	Actual Result	What needs to be changed?
Red, green, blue or white colour's selected	Background changes to the respective chosen colour + text colour changes to black	Background changes to the respective chosen colour + text colour changes to black	Nothing
Black colour selected	Background colour changes to black + text colour changes to white	Background colour changes to black + text colour changes to white	Nothing

User Testing

No particular concerns here after having several users test the program. Considering the theme selection portion of the application received the most amount of praise from the Google Form sent out to a small Y13 cohort; it was apparent that this component was both functional and stable already.

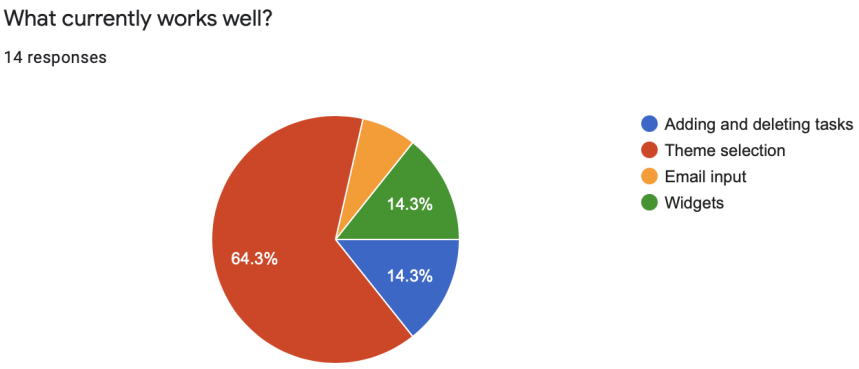


Figure 9: google form

What did the user do?	Expected Result	Actual Result	What needs to be changed?
Select a colour	Successfully selects colour	Although a colour was selected, the user though the colour was a bit muted and not that vibrant.	To use colours which are more exciting and brighter so that users find the colours a bit more fun and appealing.

Although users thought the theme selection worked well, some did report the colours not being as vibrant as they would've liked. As such, a simple change in hex colours would be required. This is to make this particular feature more appealing to the average user.

4.1.3 Email Input

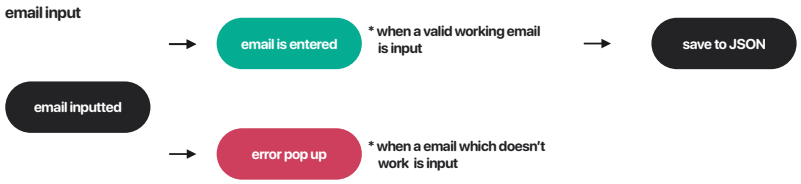


Figure 10: email input

Test Case	Expected Result	Actual Result	What needs to be changed?
Invalid email input (e.g. doesn't exist or incorrect format)	Error pop up	Error pop up	Nothing
Valid email input	Saves email to JSON	Saves email to JSON	Nothing

User Testing

What did the user do?	Expected Result	Actual Result	What needs to be changed?
Input an email	Successfully enters email	Although successfully enters email, due to lack of feedback, the user was confused for a moment.	To make it more obvious to the user, audio feedback could be added to the buttons within the application.

Some users noticed a lack of feedback when utilising buttons within the application; especially the email input button. Despite there being a pop up already, to further make it more obvious, audio could be utilised to make it clearer to the average user that the program is working - and that it isn't stuck / isn't working.

4.1.4 Internet / Weather

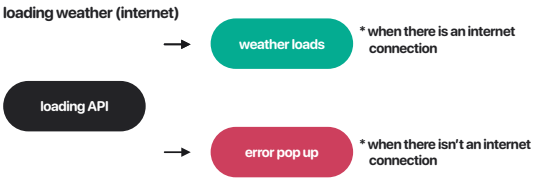


Figure 11: loading weather / internet

Test Case	Expected Result	Actual Result	What needs to be changed?
No internet access	Error pop up	Error pop up	Nothing
Internet access	Loads the required task (e.g. API)	Loads the required task (e.g. API)	Nothing

User Testing

Due to the simplicity of this section, no errors arose during testing.

4.1.5 Image Resolution

Test Case	Expected Result	Actual Result	What needs to be changed?
Load pages with images	Successfully loads without issue	Although loads, takes a lot of time	Downscale used images

When running the application for the first time, as well as when switching between different pages, it became apparent that the program started to become more sluggish. This was eventually deduced to the fact that the image icons used were of 256x256 pixels - which was overkill for the program to load. As such, it became evident that we should downscale the image (resolution will not be impacted due to how small the icons are).

4.1.6 Screen Resolution

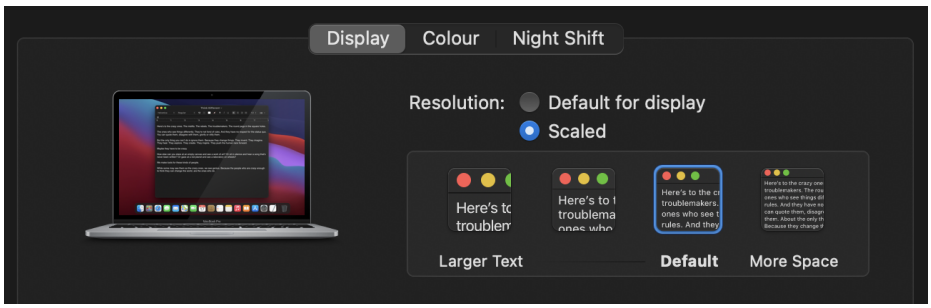


Figure 12: macos settings

MacOS has 3 primary resolution settings; these being "Larger Text", "Default' and "More Space". Given that users all around the globe use different settings, it is vital that our program is functional irrespective of the user's system settings.

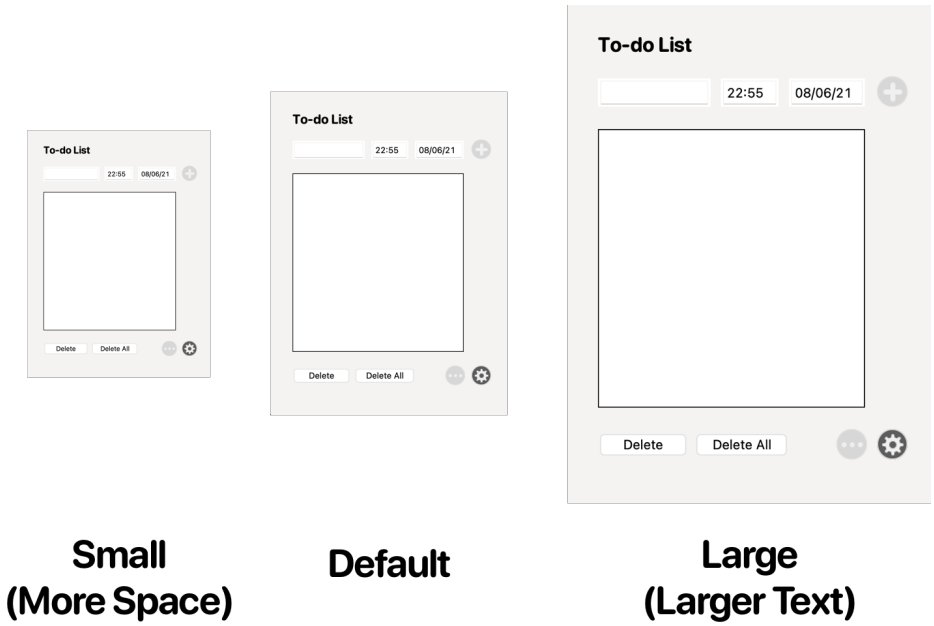


Figure 13: resolutions

Given that tkinter's widgets utilise MacOS's native widgets; because the OS is able to upscale them without issue, our program is hence operable at any resolution. This is to say that using any setting within the display pane will not effect the resolution / make the program blurry.

4.2 Improvements

Within section **4.1** of the report, we pointed out certain components which needed to be changed. To do so would allow us to improve the program, thus increasing consumer satisfaction. Below lists a few changes which were made:

4.2.1 Tool Tips

Figure 14: tool tip

To fix the issue brought up in **4.1.1 Entering Tasks**, tool tips have been added as seen below. Now, several areas of the program, when hovered over for 2-3 seconds, will show a tooltip, instructing the user what that particular part of the program is intended for. This in general makes it more convenient for the user to figure out how the program functions, thus improving ease of use; making user's want to continue to use it.

4.2.2 Audio

Audio Files Link

Addressing section **4.1.3 Email Input**, to make the program easier to use, audio feedback has been added to most buttons. Having some sort of feedback will prompt the user; thus making it clear that the action which they perform is being done. Not only would this make the program more inviting to use, it also improves upon the overall cohesiveness of the application, thus improving the user experience.

- add.mp3

  This audio file will be used for the 'adding tasks' button as well as the 'adding email button'.

- delete.mp3

  This audio file will be used for the 'delete' task button.

- delete_all.mp3

  This audio file will be used for the 'delete_all' tasks button.

The reason for not adding an audio file to every button is to not over-clutter the user with sounds as that could achieve the opposite effect of what we want. Considering studies have suggested people communicate best through mutli-modal approaches[2]; which is to say through more than 1 of the 5 human senses, adding audio can only improve interactivity.

4.2.3 Colours

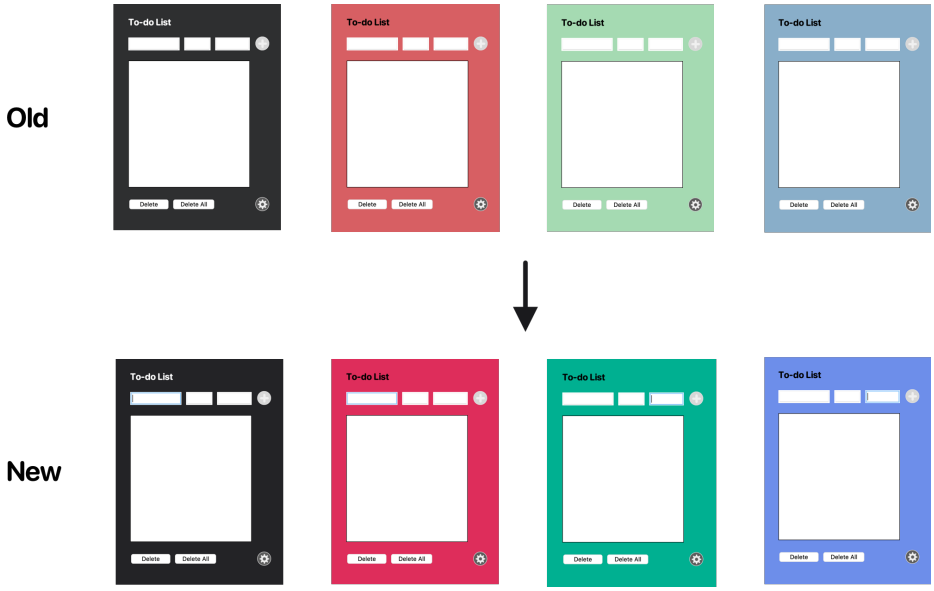


Figure 15: colour change

Given how pale the colours chosen were at the beginning, it was no suprise that within section **4.1.2 Theme Selection**, user's recommended / wanted something more vibrant. As studies have suggested colours are a primary factor in evoking emotion, having the user select a colour which they don't like would in turn disincentive them to use this program. Considering several links have been made between vibrancy and emotional impact[3], having our default colours be brighter we thought ought to be a better decision.

4.2.4 Image Resolution

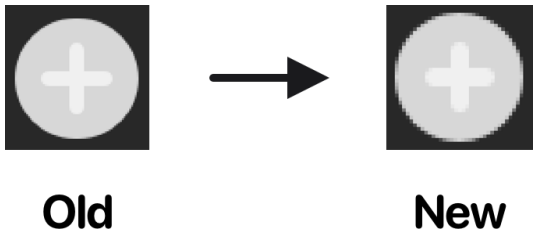


Figure 16: image resolution

From section **4.1.5 Image Resolution**, the issue of image sizes being too large was addressed. Because loading times would be dramatically decreased, reducing image resolution would be a small price to pay for exponential increases in efficiency. In addition to this, given that the visual buttons used are quite small, a drop in resolution realistically will not be noticeable by the naked eye.

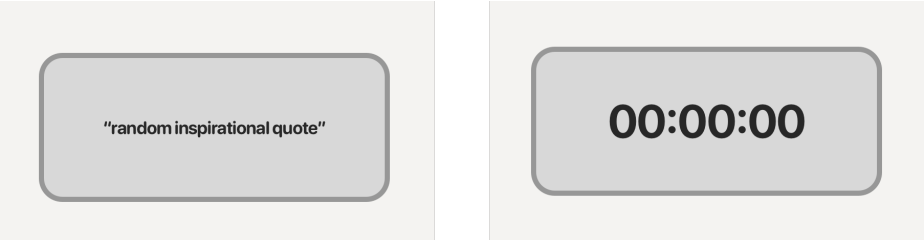
[2]Human Communication. (n.d.). Retrieved from https://www.sciencedirect.com/topics/neuroscience/human-communication

[3]Gremillion, A. S. (2020, June 30). How Color Impacts Emotions and Behaviors. Retrieved from https://99designs.com/blog/tips/how-color-impacts-emotions-and-behaviors/

4.2.5 Additional Widgets

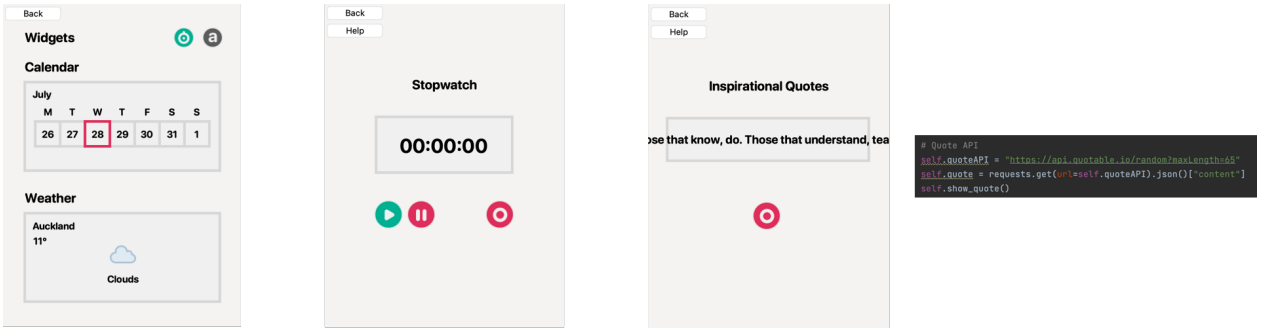
As we are adding a new component; we will be falling back to the trialing process.

Plan:



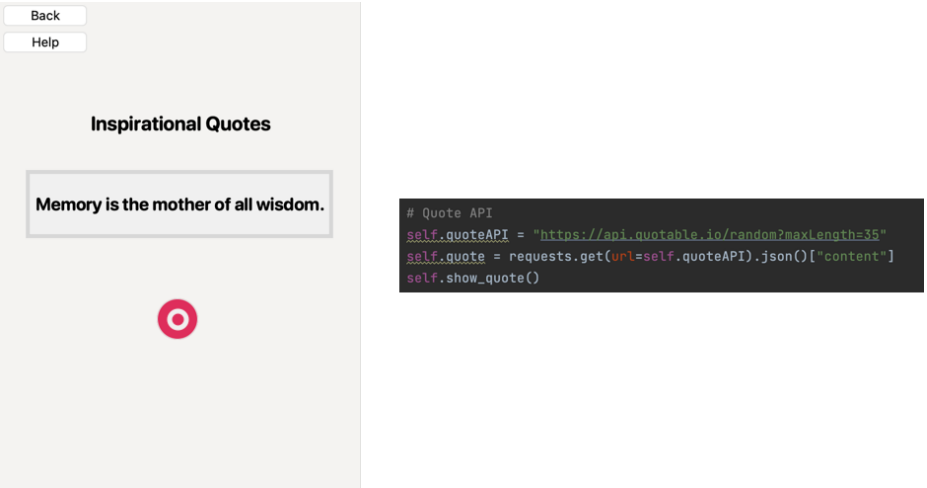
Trial:

Upon running the new component, the location for their respective buttons and the stopwatch widget seemed to work perfectly. However, as for the quotes widget, the text which was generated seemingly went over the the physical constraints of the window, thus cutting off.



Action:

Given that the quotes generated were 65 characters long - the error caused above was inevitable. Given that the quote API allows us to adjust the amount of characters generated however, to fix this issue, all we had to do was change 65 characters to 35. This results in the quote generated fully being bounded, therefore eliminating the prior concern.



5 Relevant Implications

5.1 Intellectual Property

Intellectual property is the concept which gives people exclusive right to "own" a piece of creative work. Making sure our program stays on the side of the law not only makes sense ethically, but also financially (infringement can result in thousands of dollars).
Given that this is the case, copyright law shouldn't be taken lightly; and as such, throughout the process of developing our application, all material which wasn't created by myself has been credited accordingly. Although most assets used were completely original,

- The weather icons used were sourced from Icons8,

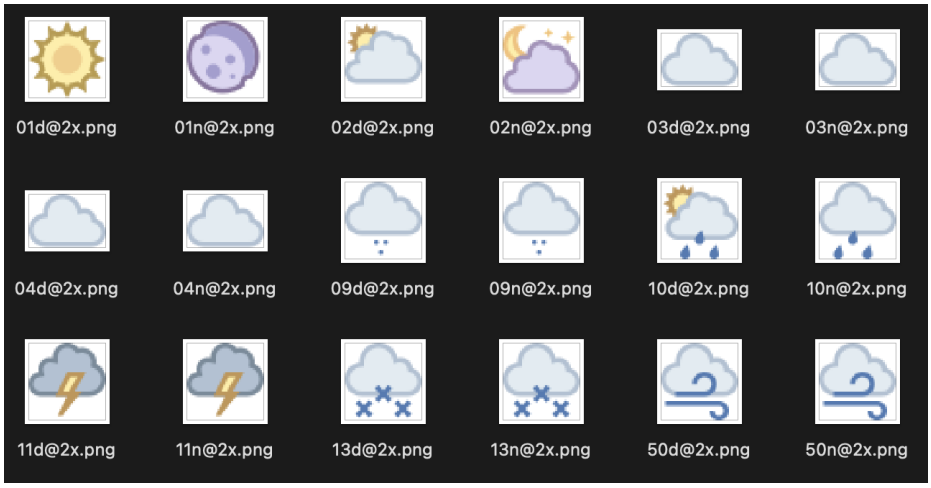


Figure 17: weather icons

- and the audio files used were retrieved and edited from Zapsplat.

With both particular sites, use of these icons are allowed on the basis that within the "description" of the application, a link is provided to their site. Because attribution has been both stated within the source code as well as Github page; no infringement has been breached.

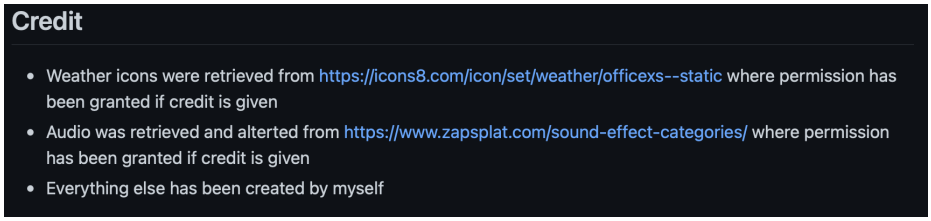


Figure 18: github credit

Having utilised royalty free assets - means that they are completely free to use. Given this is the case, there would be no financial set backs if this application were to enter the market. This would mean the contribution margin for each sale would be purely positive, thus resulting in only profit. This is good as not only does it minimise the workload, but also helped get this application created much faster. Overall, because we've fulfilled the specifications under Icons8 and Zapsplat's licensing agreements, use of these assets are deemed fair; as such, we are not susceptible to any legal repercussions.

5.2 Accessibility

Accessibility is the idea that anybody can use our program; irrespective of age, background, and etc. Given the intention of the program is to improve orderliness among the masses; making sure our application is useable by the majority of people can only further our cause - and as such, accessibility is extremely important to us. Therefore, when designing the program, certain key elements were taken into consideration.

- Short sighted

For those who may be short sighted, texts which appear on screen can often appear blurry. Given this is the case, it is important our application can be upsized appropriately if need be. This way if certain elements are too small to be read, enlarging them won't cause any distortion. Previously it was mentioned that tkinter's widgets are native MacOS widget's - which therefore do not distort upon resizing. Given that there are currently an estimated 1.4 billion people[4] who are short sighted, making sure our program was able to accommodate this large portion of the population was definitely a priority. Having accomplished this; makes our application more accessible.

- Colour blindness

Within our world, deuteranomaly is the most common type of colourblindness[5]. It impacts about 6% of all males and 0.4% of all females. In retrospect these percentages are fairly small; however, they still do represent quite a large amount of people - and as such, it is in our best intention to make sure their experience is just as good if they weren't colour blind. Primarily, the only component within our application that could demerit the experience would be the theme selection. To identify whether or not there would be a difference, an online piece of software called Coblis was used. This allows us to import images which then applies a filter that simulates common kinds of colourblindness.

Figure 19: deuteranomaly filter

Although there is a difference to what it would've look liked, the colours are still clearly distinct and different. Because this is the case, those who suffer from colour blindness although may see a slight difference; the program will still function as intended - thus improving accessibility despite the small compromise.

5.3 Privacy

The concept of privacy is that anybody is able to seclude themselves, or the information which they own to only themselves.

[4]The sharp rise of short sight: Is myopia becoming a global epidemic? (n.d.). Retrieved from https://www.optimax.co.uk/blog/short-sight-myopia-global-epidemic/

[5]Types of Color Blindness. (n.d.). Retrieved from https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/color-blindness/types-color-blindness

Privacy is an essential human right; and particularly within this decade - has become more important then ever. Considering the current status quo, it is important that our program is not subject to current norms. Living in a digital society already reduces one's privacy by a large margin, and it is vital that our application does not contribute to this. As such, all information that is saved is only ever stored locally. This in turn means that a user's private information is not susceptible to being hacked through online attacks or data breaches, as the only way to access it would be through opening the JSON file. Considering there is an cyber attack every 43 seconds[6] (estimate), using an offline database benefits us in this regard. In doing so, user's can be confident that their data is their data, and that not only is it safe, it is also not being shared. Designing our program this way is beneficial as it'll likely improve consumer satisfaction in the long run.

5.4 Social

Social impact is the net effect on a society. As this is the case, it was important for us to concern ourselves with this particular implication. Given the purpose of our digital outcome (To-Do List), due to its simplicity; will not misrepresent any social groups. This by extension means it will not cause offence to anyone, nor will it stigmatise any stereotypes - as our digital outcome can only deliver data. Because there is no bias with that data, socially it should also not offend anyone. As for appeal, the program in question should be usable by all; irrespective of age, gender or ethnicity. The program follows design conventions which look aesthetically appealing; and the colours chosen were vibrant and clear. As this is the case, the majority of those who use this program theoretically should be able to enjoy it irrespective of their respective backgrounds. Implications wise in regards to functionality; users should be able to improve their orderliness which as a result will improve their performance in their respective fields - providing a net positive impact.

5.5 Health and Safety

Health and safety is the idea which essentially dictates whether our application is considered safe or not. The program in question is safe to use. Given at it's core - it is simply a to-do list application, being a piece of software which does not overreach any boundaries; it should not impact the user in any harmful way. Because this is the case, the application will not require any warnings regarding this topic as this program does not and cannot be used for any illegal activity. If anything, the program can only encourage safe practices, as it is only intended to be used as a planning tool to improve orderliness.

[6] 15 Alarming Cyber Security Facts and Stats. (2021, January 12). Retrieved from https://www.cybintsolutions.com/cyber-security-facts-stats/

6 Evaluations

The use of planning, trialling and testing helped aid us during the development of our digital outcome.

Through the use of our project management tools; these being our visual diary, github and feedback forms - allowed us to plan in an effective way. In doing so, it gave us an initial idea as to what the final digital outcome would look like, providing us with something to strive towards. Having everything outlined as well meant we knew what needed to be in the program - thus mapping out what each component would do, and how it links towards other components. As for version control; Github was able to keep track of all changes made, keeping a history of our program, making it easier to identify what changes have been made and whether it was an improvement worth keeping. If not, we were able to back track accordingly. Utilising a Google Form was also beneficial as it allowed us to keep a backlog of feedback. Having this tool meant we had a consistent flow of feedback, making it easier to identify what needed to be fixed.

In regards to the trialling process; this was quite beneficial as it allowed us to deconstruct our application into functioning components. This meant during the development of the project, we weren't creating a fully functioning program right off the bat, but instead components with one or two features. Doing so meant the workload was more manageable. Alongside this, planning each individual component separately meant in the testing process; if there was an error, we could identify which component was causing it, saving us time in the long run. This by extension meant we were able to iron out bugs more efficiently, improving the stability of our program.

The testing process was our final step in fixing all bugs. By considering every possible case, we were able to make sure the application worked as intended. This acted as a last resort to our testing phase - on the off chance we had missed something important. During this particular process, we also had other users test the program, allowing us to receive even more valuable feedback. Doing so meant that we were able to add additional features which weren't originally planned; making our program a lot more cohesive than it originally was. The testing process also required us to combine all our components, thus by the end of this phase, we were able to complete a fully functional application ready to be used by anyone.

Maintaining this program should not require too much work. Given its underlying foundation uses the tkinter framework - due to its popularity and commitment to stability and compatibility, making sure our program works in the upcoming years shouldn't be too much of a concern. With that said, although functionally our program will work on both MacOS and Windows 10, there are still issues with UI on Windows. This is something which we won't be able to do much about as it is an issue with the tkinter framework. However, given the surge of popularity with MacOS in the past decade, this may not be an issue in the near future. In regards to legality, because most assets which were used are original, and those that were sourced from 3rd parties all have the appropriate licenses, legally this program should be in the clear for as long as it is supported.

As for what could have been done better; if we had utilised something other than tkinter, our UI possibly could have looked better. Having to use the preset widgets meant that design was fixed to only what tkinter could offer, which did limit the possibilities of what we could produce. Alongside this, there are still some minor errors in regards to cross platform compatibility with tkinter. The UI elements tend to shift, and some widgets just do not work properly. If we had explored other alternatives, some of these concerns could have been addressed.

7 Present your Outcome

Video Presentation Link