**91907**  
COMPLEX PROCESSES TO DEVELOP AN OUTCOME - L3

**BENJAMIN HUME**

0152085189

91906 - 91907 Complex Programming

**Google Drive Link:**

https://drive.google.com/drive/folders/15VMVrWBqDgSHcWEH9GGeQ4gwsRPk0IgR?usp=sharing

PROJECT MANAGEMENT: SETUP

Effective design processes rely on planning; setting achievable goals and managing time accordingly to meet those goals. Milestones have been planned throughout my development process; I have recorded these below. Firstly, I created a Gantt chart based on an estimate of how long each process within the development would take. This Gantt chart provided milestones throughout development that guided the process.

I decided to format my goals as a Gantt chart because it accommodates easier communication between my teammate and I. Key information can be located very quickly from a Gantt chart due to the chart’s simplicity, this makes using the chart easy, especially when contrasting colours are used to draw attention to important details such as time blocks.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 |
| Begin designing digital outcome |  |  |  |  |  |  |  |  |  |
| Design data input / retrieval processes |  |  |  |  |  |  |  |  |  |
| Create framework of GUI |  |  |  |  |  |  |  |  |  |
| Create mockup GUI |  |  |  |  |  |  |  |  |  |
| Implement data entry process / landing GUI |  |  |  |  |  |  |  |  |  |
| Implement account creation process / account GUI |  |  |  |  |  |  |  |  |  |
| Implement location creation process / location GUI |  |  |  |  |  |  |  |  |  |
| Implement data retrieval process / search GUI |  |  |  |  |  |  |  |  |  |
| Implement final features |  |  |  |  |  |  |  |  |  |
| Review / redesign / adjust digital outcome |  |  |  |  |  |  |  |  |  |
| Analyze digital outcome |  |  |  |  |  |  |  |  |  |

To manage these milestones more effectively as I completed and altered them, I decided to use a webapp called Trello. Trello features the ability to move milestones between *to do*, *in progress* and *done* sections, making it ideal for recording progress and changing milestones throughout development. This is one of Trello’s most significant features because, unlike most other management tools, it allows objectives to be marked as in progress again if they need to be updated or reintegrated.

|  |  |
| --- | --- |
| **Trello Logo ➞** | Trello Logo and symbol, meaning, history, PNG |
| **Workspace During Week 1 ➞** |  |
| **Workspace During Week 3 ➞** | Graphical user interface  Description automatically generated |
| **Workspace During Week 5 ➞** |  |
| **Workspace During Week 7 ➞** |  |

|  |  |
| --- | --- |
| Additionally, I took advantage of Trello’s *Due Date* feature to set realistic and achievable goals for each milestone, according to the Gantt chart I had created prior. Of course, these dates were just estimations based on limited prior experience, so I had to adjust them throughout the development process. |  |

|  |  |
| --- | --- |
| Graphical user interface, application  Description automatically generatedGraphical user interface, text, application, chat or text message  Description automatically generated | Trello was also chosen because of its collaboration and sharing features; my outcome was collaborative project between myself and another year 13 student, thereby utilizing Trello allowed us to effectively divide tasks evenly between ourselves to meet the time goals we had set out previously.  Furthermore, Trello can be accessed from almost any device in any location, making it ideal for effectively managing checkpoints as they are completed.  In addition to this, Trello allows users to assign tasks to specific collaborators, this feature was used extensively in conjunction with the due date feature to organize what my group partner and I should be doing at any given time. We also took advantage of the comment feature; each ‘card’ can have a comment / description attached to it. Feedback was placed here so that either of us had immediate access to all the information needed when developing a component. |

Feedback is a critical part of the design process. To effectively collect, collate and integrate feedback from a diverse range of end users I actively took all opportunities to get feedback from as many test users as possible. I then collected this feedback in a grid, sorting comments into categories based what aspect of the program they relate to and whether they’ve been addressed or not. Comments that have been addressed have been ruled out in order to record progress. By doing this I was able to incorporate all feedback quickly and methodically.

|  |  |  |  |
| --- | --- | --- | --- |
| **BUGS / ERRORS** | **GUI LAYOUT** | **GUI DESIGN** | **EASE OF USE** |
| *~~“Program crashes when I search for a previous entry.”~~* | *~~“Text on the landing screen is too small.”~~* | *~~“Pure black and pure white are difficult to look at.”~~* | ***“Entering the date every time an entry is recorded time takes up time.”*** |
| ***“New entries are not added to the interface properly when they are entered.”*** | *~~“It would be easier to navigate if the back button was always in the same place.”~~* | *~~“The gradient looks cool, but it would be less distracting if the two colours didn’t contrast as much.”~~* | *~~“It would be easier to navigate if the back button was always in the same place.”~~* |
| ***“A random comma is added to the start of my username whenever I try to log in.”*** | ***“Back button moves around; this makes it harder to find.”*** | *~~“This could be hard to look at when it’s darker”~~* |  |

ADDRESSING RELEVANT IMPLICATIONS

**Accessibility**

Accessibility is among the most important considerations because the outcomes audience encompasses a wide range of people of varying abilities. If the outcome is to be successful, it must be accessible to the greatest amount of people possible, making the contact tracking process more effective. Because my outcome will feature a GUI, I must consider visual elements of the GUI including how colour interacts with all forms of colour blindness, how size and font interact with visual impairments, how shape and size interact with limited motor function and how certain symbols may interact with differing cognitive function.

**Usability**

Usability considerations are at the heart of the GUI design choices I have made. Contact tracking can only work if the majority of a population take part in the process, which is only possible when the tracking method is easily usable by the majority of the population. I must identify and use established UI visual hierarchy rules so that users can easily identify important features in the GUI, reducing the time taken to contact track and thereby increasing the chances of a greater participation rate. Additionally, I should incorporate features to reduce the chances of human error when entering data into the contact tracking system because if inaccurate data enters the system, it could create false leads and slow down the contact tracking process significantly.

**Functionality**

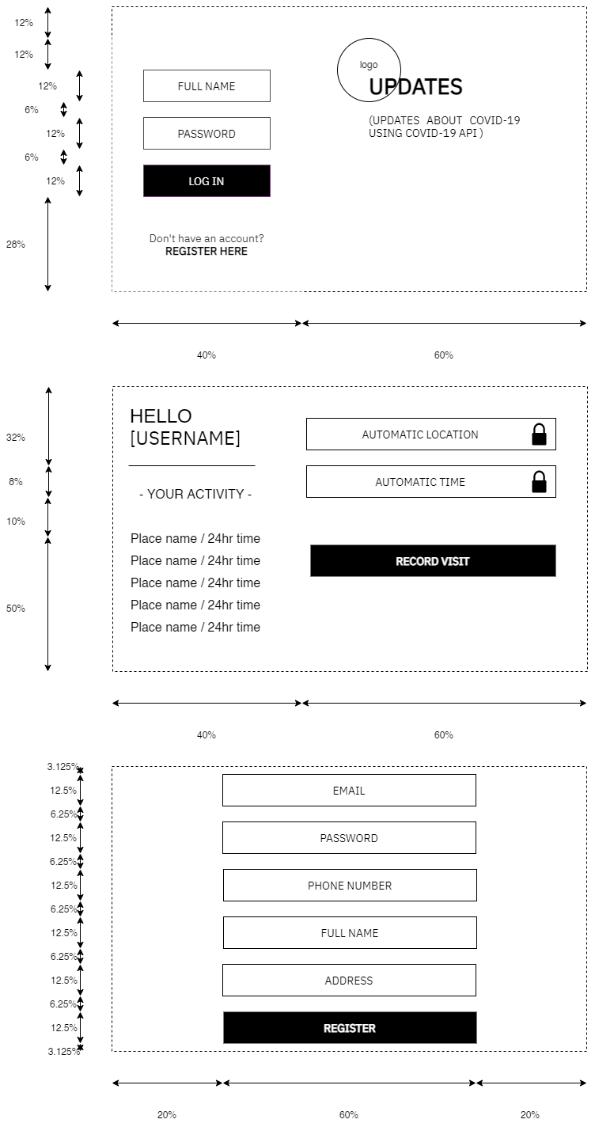
Functionality is a critical consideration of the outcome because the safety of the public depends on how functional the adopted tracking process is. If the outcome is not functional, it cannot accomplish the goal of providing effective contact tracking to communities without access to it. Furthermore, usability is only important if intended audience decide to use the program. During my initial inquiry I conducted research into what makes contact tracing more effective, and specifically how I can increase user interaction with the program. “Appealing to customers’ brains (knowledge) rather than their hearts (emotions) can increase the effectiveness of contact tracing as people are more likely to participate” - Dr Raymond Xia. An individual is more likely to participate in a contact tracing system if they think that it is reliable. In systems where there is skepticism, far fewer individuals tend to participate, reducing the effectiveness of the contact tracing system. In order to increase the effectiveness of contact tracing the system should “demonstrate professionalism, competence, and reliability”. To do this I must consider attributes of the aesthetic carefully to appeal to the logical part of the user’s brain. Additionally, personal benefit can increase the chances of an individual taking part in the contact tracking process. Added personal benefit should be taken into account when designing the outcome in order to take advantage of this. Finally, the functionality of a contact tracking system depends largely on how long it takes contact tracking services to react based on the information entered into the system. The contact tracking process must be as time optimized as possible.

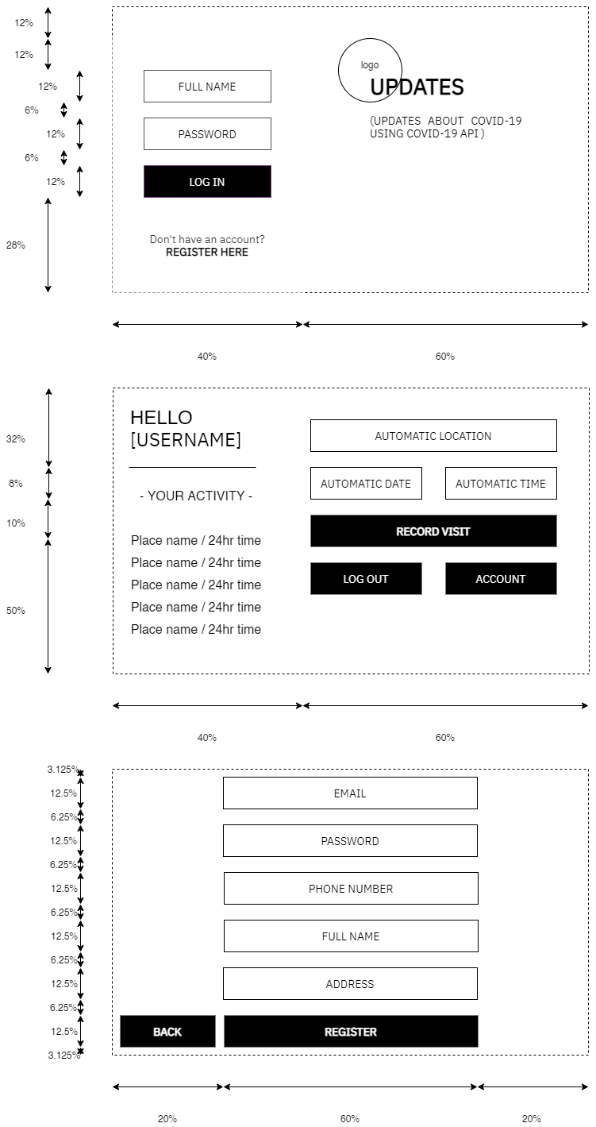
**Health and Safety**

Health and safety are possibly the most important considerations for my outcome, taking into account the purpose of a contact tracing program. The purpose of the outcome is to provide effective contact tracking to communities without it and, ultimately, assist with the user’s health during the pandemic and prevent the health problems related to COVID-19 to the wider population. This goal can only be achieved if the outcome does not negatively affect the health of the user. Within health and safety, I have considered how I can increase the outcomes ability to prevent the spread of COVID-19, how to mitigate the health effects associated with frequent device use and how I can ensure the outcome does not invertedly spread misinformation surrounding COVID-19 and the pandemic, thereby decreasing the chance of encouraging action which may put users in harms way.

GRAPHICAL USER INTERFACE DESIGN

Throughout the GUI design process, I developed multiple evolutions of each interface in order to address different issues that had come up or facilitate more effective interactions with the user. Firstly, I created a wireframe for each interface within the application, although this too was improved with each iteration to produce the most effective layout possible.

Graphical user interface, website

Description automatically generated**Iteration 1 Iteration 2 Iteration 3**

**Key evolutions that took place from the first to the third iteration of the GUI include:**

* An added back button on the account creation screen, allowing the user to navigate back to the landing screen.

* Incorporation of the “F” layout in the landing screen and user dashboard, replacing the vertically ascending layout. This was incorporated based on GUI principals relating to how users scan a GUI. This layout presents key information to the user first, making important features of the GUI immediately obvious to the user.
* Addition of the “DATE” textbox in the user dashboard; later on in the development process I realized that this information would be important when extracting information accurately.
* Automatically updating “USERNAME” text was implemented into the user dashboard to make the dashboard seem more personal.
* Additional text was implemented into the GUI sparingly to explain features of the GUI where necessary, such as “YOUR ACTIVITY” above the user’s recent activity.

**I then investigated colour schemes and possible graphic designs. These have been included below:**

Graphical user interface

Description automatically generatedA picture containing text, screenshot, businesscard

Description automatically generatedGraphical user interface, website

Description automatically generatedGraphical user interface

Description automatically generated with low confidence

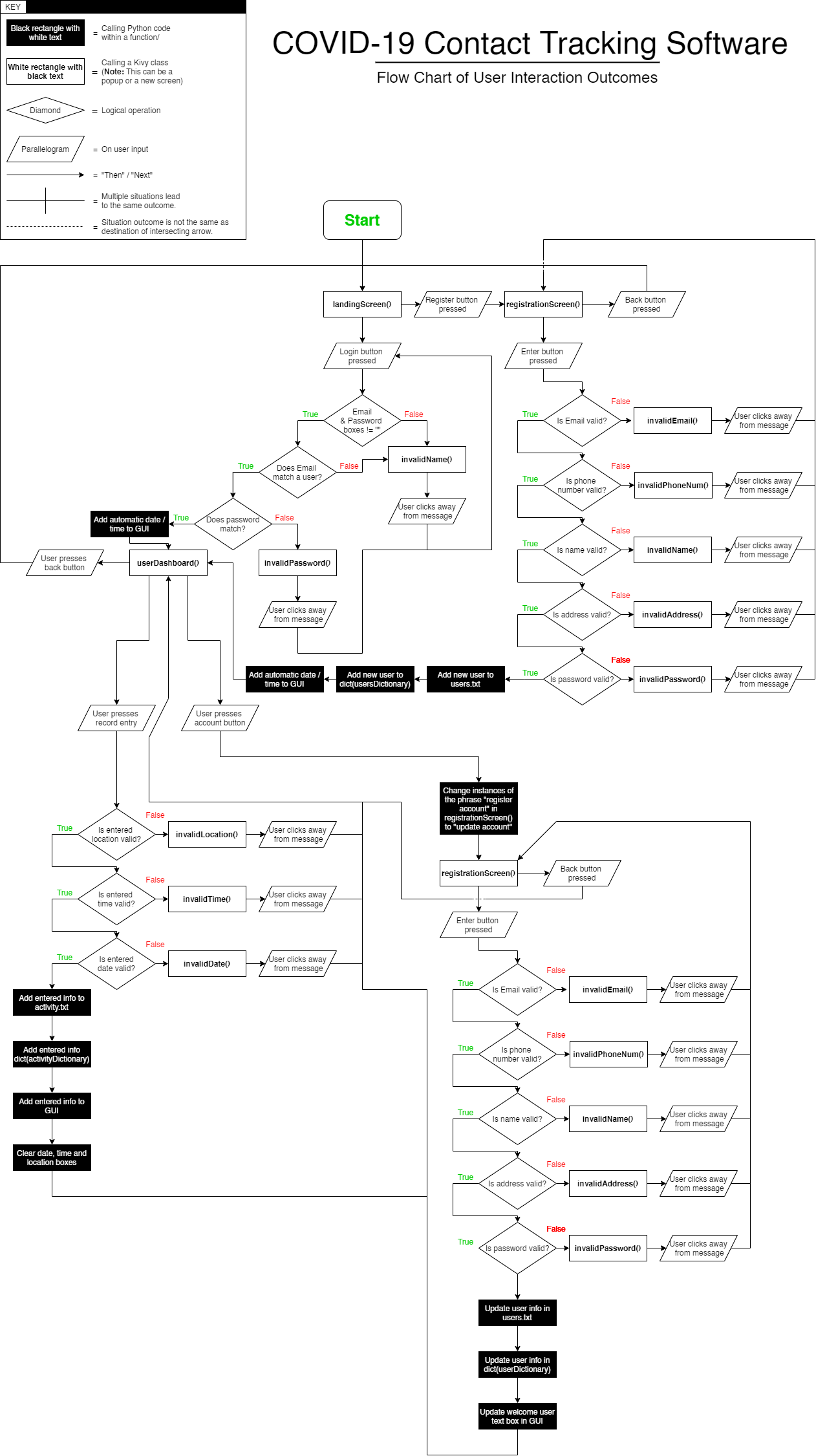
I eventually decided to further develop the bottom left UI design and incorporate it into the program because, after interviewing a range of possible users, I realized that it was unanimously recognized as the most visually pleasing and intuitive GUI out of those I had created. While the first option looks aesthetically pleasing, the background image concept is distracting, and takes away from the visual hierarchy. Additionally, the slanted stripes seen at the top of the first and second design serve no purpose and add very little to the aesthetics of the GUI. The colour pallet and design features seen in the second design are not visually pleasing, however the hierarchal, left aligned placement of key elements was used in the final design because it allows users to quickly locate these features. Therefore, the slanted layout of elements seen in the first and fourth designs was scrapped in favor of the more intuitive vertical orientation.

PROGRAM STRUCTURE

The program structure underwent several changes during the development process, **these include:**

* The addition of a back button in the account registration screen, allowing the user to easily navigate back to the landing screen.
* Implementation of a date text input and validation for this input, thereby collecting all key information about each recorded activity.
* Location entry validation that ensures the user has not made any typing errors when entering their location; certain characters and symbols cannot be present in a valid address, by excluding these symbols the program reduces the chance of information being recorded incorrectly.
* Elements of the account registration class being slightly altered mid-program and repurposed to create a very similar account settings screen, without the need for a new class or needlessly copied code.
* More accurate phone number validation that reduces the chance of input error when the user is entering their phone number, thereby mitigating the risk of potentially infectious individuals being uncontactable in the event of an outbreak.
* A greater number of error messages, allowing the program to better convey what must be amended in the user’s entry at any given point in the program’s runtime.

I have added the final copy of the program structure diagram below on the next page, with each of the previously mentioned changes included.



Each class is represented by a white rectangle with bold, black text enclosed within it. The full list of classes can be split into two categories; classes designed to act as a separate screen, and classes designed to present themselves as a pop-up box above another screen class.

|  |  |  |
| --- | --- | --- |
| **NAME** | **CATAGORY** | **PURPOSE** |
| landingScreen() | Screen | Allows user to login, provides up to date COVID-19 statistics and links to registration screen. |
| registrationScreen() | Screen | Allows users without an account to create one and allows current users to update their information. |
| userDashboard() | Screen | Shows user’s recently logged entries, allows users to log new entries and links to account information screen. |
| invalidPassword() | Pop-Up | Informs user that the password they have entered is invalid or incorrect. |
| invalidName() | Pop-Up | Informs user that the name they have entered is invalid. |
| invalidEmail() | Pop-Up | Informs user that the Email they have entered is invalid or does not exist in the database. |
| invalidPhoneNum() | Pop-Up | Informs user that the phone number they have entered is invalid. |
| invalidAddress() | Pop-Up | Informs user that the address they have entered is invalid. |
| invalidLocation() | Pop-Up | Informs user that the location they have entered is invalid. |
| invalidDate() | Pop-Up | Informs user that the date they have entered is invalid. |
| invalidTime() | Pop-Up | Informs user that the time they have entered is invalid or does not match the 24-hour time format. |

PROBLEM DECOMPOSITION

By planning out the individual components of the outcome and breaking the task down, I was able to manage time more effectively while ensuring each part of the program solves the problems it was implemented to solve before continuing onto the next component. I have separated the problem into the following tasks and listed the corresponding classes beneath.

|  |  |  |  |
| --- | --- | --- | --- |
| **Landing Screen**    landingScreen() | **Registration Screen**  registrationScreen() | **User Dashboard**  userDashboard() | **Error Popups**  invalidPassword(), invalidDate(),  invalidTime(), invalidPhoneNum(),  invalidAddress(), invalidLocation(),  invalidEmail(), invalidName() |

By breaking each task down into its corresponding class(es) I was able to better understand how parts of the program would interact with each other before I started programing, this made the programing process far easier and significantly more accurate. By following the plan I had created previously, I was able to reduce the number of errors I encountered and quickly identify flaws in the program’s structure using the diagram, rather than having to retroactively search for them in my code. I have recorded this diagram as well as each individual class under **Program Structure**.

I have correlated these tasks to approximate deadlines in order to track my progress and make sure I am on track. The due dates for each task were approximations based on the amount of work we would have to do to achieve the task, however throughout development we learned a lot about how long different tasks take. For example, we found that coding GUIs and popup boxes tends to take significantly less time than working with data handling because elements can be copied from one class to another. Throughout development we made a number of changes to our schedule, these have been recorded below in the form of the first and final Gantt charts.

**Schedule pre-development:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 |
| Begin designing digital outcome |  |  |  |  |  |  |  |  |  |
| Design data input / retrieval processes |  |  |  |  |  |  |  |  |  |
| Create framework of GUI |  |  |  |  |  |  |  |  |  |
| Create mockup GUI |  |  |  |  |  |  |  |  |  |
| Implement data entry process / landing GUI |  |  |  |  |  |  |  |  |  |
| Implement account creation process / account GUI |  |  |  |  |  |  |  |  |  |
| Implement location creation process / location GUI |  |  |  |  |  |  |  |  |  |
| Implement data retrieval process / search GUI |  |  |  |  |  |  |  |  |  |
| Implement final features |  |  |  |  |  |  |  |  |  |
| Review / redesign / adjust digital outcome |  |  |  |  |  |  |  |  |  |
| Analyze digital outcome |  |  |  |  |  |  |  |  |  |

**Changes to schedule during development:**

* Each screen was split up into its GUI and its data handling process, this allowed us to make sure we weren’t spending too long on the screen’s GUI which shouldn’t take as long as the other features of the screen. This change was made during **week 3**.
* Creating popup boxes were separated from the other tasks and grouped into one separate task because creating popup boxes using Kivy is not a difficult or time-consuming process. This change was made during **week 3**.
* Creating all of the GUIs was rescheduled to take place in one week because it took significantly less time than we had planned. Furthermore, making the GUIs first made implementing and testing the data handling processes much easier because it meant we’d already have the buttons and textboxes to link our code to. This change was made during **week 3**.
* The implementation of data handling for each screen was given a full week because, accounting for error handling and data validation, coding this feature took longer than we had predicted. This change was made during **week 4**.
* The order that each data handling process was implemented in was changed so that each screen could be more easily integrated with the last. For example, it was significantly easier to create and test the login process once the registration process had been completed because we already had accounts registered with the application from the account registration testing. This change was made during **week 4**.
* Creating popups was moved to the end of development because we knew that this process would take far less than a week, and so if another task took longer than expected we could move everything forward by a few days to compensate. This change was made during **week 7**.

**Schedule with changes made during development:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 |
| Begin designing digital outcome |  |  |  |  |  |  |  |  |  |
| Design data input / retrieval processes |  |  |  |  |  |  |  |  |  |
| Create framework of GUI |  |  |  |  |  |  |  |  |  |
| Create mockup GUI |  |  |  |  |  |  |  |  |  |
| Create landing screen GUI |  |  |  |  |  |  |  |  |  |
| Create user dashboard GUI |  |  |  |  |  |  |  |  |  |
| Create account registration GUI |  |  |  |  |  |  |  |  |  |
| Implement account registration process |  |  |  |  |  |  |  |  |  |
| Implement login process |  |  |  |  |  |  |  |  |  |
| Implement log entry recording process |  |  |  |  |  |  |  |  |  |
| Implement user record search process |  |  |  |  |  |  |  |  |  |
| Create and implement popups |  |  |  |  |  |  |  |  |  |
| Review / redesign / adjust digital outcome |  |  |  |  |  |  |  |  |  |
| Analyze digital outcome |  |  |  |  |  |  |  |  |  |

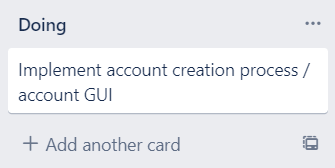
**DEVELOP YOUR COMPONENTS**

COMPONENT PLANNING

**REGISTRATION SCREEN**

**|** AKA registrationScreen()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 |
| Create framework of GUI |  |  |  |  |  |  |  |  |  |
| Create mockup GUI |  |  |  |  |  |  |  |  |  |
| Create account registration GUI |  |  |  |  |  |  |  |  |  |
| Implement account registration process |  |  |  |  |  |  |  |  |  |
| Implement log entry recording process |  |  |  |  |  |  |  |  |  |

****The account registration screen has been split into the same two main tasks as the other screens, also seen above: creating the GUI and implementing the login process. My partner created the GUI and data entry process for this component so that the GUI and data entry code could be more easily integrated. This GUI was also planned out prior, this has been explained under **graphical user interface**.

Once again we brainstormed how this component would work before beginning development during the design data input / retrieval phase. This brainstorm allowed us to settle on what constitutes as valid information for a new account. We decided that every email must contain an ‘@’ symbol and a ‘.’, that the phone number must be an integer, that the full name must contain at least one space and that address must also contain at least one space. These specifications were decided upon in order to reduce the chance of human error when entering information into the application.

**LANDING SCREEN**

**|** AKA landingScreen()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 |
| Create framework of GUI |  |  |  |  |  |  |  |  |  |
| Create mockup GUI |  |  |  |  |  |  |  |  |  |
| Create landing screen GUI |  |  |  |  |  |  |  |  |  |
| Implement login process |  |  |  |  |  |  |  |  |  |

*Graphical user interface, text, application

Description automatically generatedCalendar

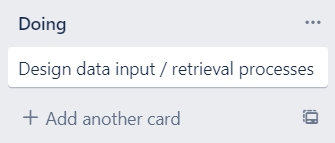
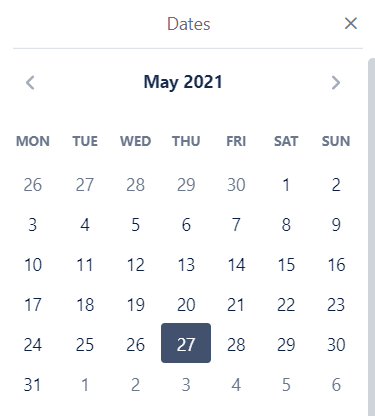
Description automatically generated*The landing screen has been split into two main tasks seen above: creating the GUI and implementing the login process. The full list of tasks were divided evenly between my partner and I and added to the Trello board seen to the right. I created the GUI for this component and the data entry process, this allowed us each to work more effectively because I understood my code and could integrate the GUI more easily.

We created the wireframe and mockup for the landing screen GUI beforehand, this has been discussed under **graphical user interface**. We also brainstormed how the login process would work during the first week and, while we initially decided that the user should log in with their full name, we settled on using their Email as our primary key because multiple people can share the same full name, but not the same Email. During this brainstorm we also decided that we should have COVID-19 stats and tips in the landing screen to keep users informed.

**USER DASHBOARD**

**|** AKA userDashboard()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 |
| Create framework of GUI |  |  |  |  |  |  |  |  |  |
| Create mockup GUI |  |  |  |  |  |  |  |  |  |
| Create user dashboard GUI |  |  |  |  |  |  |  |  |  |
| Implement log entry recording process |  |  |  |  |  |  |  |  |  |
| Implement user record search process |  |  |  |  |  |  |  |  |  |

**As with the other screens, the user dashboard was split into two tasks: creating the GUI and implementing the login process. This time I was responsible for creating both the GUI and the backend data handling process.

We created the wireframe and mockup for the user dashboard GUI beforehand, this has been discussed under **graphical user interface**. We also brainstormed how data from the dashboard should be formatted, stored, and displayed. Ultimately, we decided to format entries as a dictionary, with the time, date and location concatenated onto each other and separated with at ‘!’ symbol so that they could be split apart again later. We decided to user 24-hour time, and the DD/MM/YY format because we believed that these would be the most universally recognized formats. We also decided to display the user’s recent activity in the GUI as a ‘scoreboard’ to encourage frequent use of the contact tracker.

While it is a feature of the admin user dashboard, and so considered to be a part of the user dashboard, it is also worth mentioning that the record search process was developed by both myself and my partner at this time. The brainstorming for this subcomponent was done during the first week. We decided that a dedicated admin account with a secure, predetermined password, should be able to search for users by their email, and that all of the user’s activity should be presented in the GUI to allow contact tracking services to quickly collate information. Additionally, we decided that new admin accounts should not be able to be created as this presents possible security risks, such as outside parties having access to personal information without the knowledge of contact tracking officials.

**POPUP BOXES**

**|** AKA invalidPassword(), invalidDate(), invalidTime(), invalidPhoneNum(), invalidAddress(), invalidLocation(), invalidEmail(), invalidName()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 |
| Create and implement popups |  |  |  |  |  |  |  |  |  |

Calendar

Description automatically generatedGraphical user interface, text, application

Description automatically generatedFinally, a series of popup boxes were implemented into each GUI by my partner to communicate any invalid entries that occur to the user. We decided that the text contained within these popup boxes should be specific to each invalid entry case, and that a brief summary of how to fix the issue should be included in the popup. For example, if the user enters a date in the incorrect format a textbox will appear explaining the correct format and instructing them to amend their entered time.

Before beginning the popup box development, we discussed how many popup boxes there should be in order to have one for every possible invalid entry case. We concluded that there should be a total of 8 unique popup boxes.

A template popup box was made, we planned to copy the styling of this box and replace the text for every subsequent popup to reduce the amount of time it would take to implement all 8 popup boxes.

COMPONENT TEST PLAN

**REGISTRATION SCREEN**

**|** AKA registrationScreen()

|  |  |
| --- | --- |
| *User presses register* ***and*** *Email box is empty:* | ***Display all fields must be filled out popup.*** |
| *User presses register* ***and*** *password box is empty:* | ***Display all fields must be filled out popup.*** |
| *User presses register* ***and*** *phone number box is empty:* | ***Display all fields must be filled out popup.*** |
| *User presses register* ***and*** *full name box is empty:* | ***Display all fields must be filled out popup.*** |
| *User presses register* ***and*** *address box is empty:* | ***Display all fields must be filled out popup.*** |
| *User presses register* ***and*** *entered Email does not contain an ‘@’ symbol:* | ***Display invalid Email popup.*** |
| *User presses register* ***and*** *entered Email does not contain a ‘.’ symbol:* | ***Display invalid Email popup.*** |
| *User presses register* ***and*** *phone number contains any non-integer characters:* | ***Display invalid phone number popup.*** |
| *User presses register* ***and*** *full name does not contain a space:* | ***Display invalid full name popup.*** |
| *User presses register* ***and*** *full name contains a number or character:* | ***Display invalid full name popup.*** |
| *User presses register* ***and*** *address contains less than 5 letters or symbols:* | ***Display invalid address popup.*** |
| *User presses back:* | ***Change screen to landing screen.*** |
| *User presses register* ***and*** *address contains more than 4 letters or symbols* ***and*** *Email contains an ‘@’ symbol* ***and*** *Email contains a ‘.’ Symbol* ***and*** *phone number contains only integers* ***and*** *full name contains a space* ***and*** *full name contains letters* ***and*** *password box is not empty:* | ***Change screen to user dashboard and add user to database.*** |

**LANDING SCREEN**

**|** AKA landingScreen()

|  |  |
| --- | --- |
| *User presses enter* ***and*** *email box is empty:* | ***Display invalid Email/password popup.*** |
| *User presses enter* ***and*** *password box is empty:* | ***Display invalid Email/password popup.*** |
| *User presses enter* ***and both*** *text boxes are empty:* | ***Display invalid Email/password popup.*** |
| *User presses enter* ***and neither*** *text boxes are empty* ***and*** *username does not match a user in database:* | ***Display Email not recognized popup.*** |
| *User presses enter* ***and neither*** *text boxes are empty* ***and*** *username matches a user in database* ***but*** *password does not match for that user:* | ***Display incorrect password popup.*** |
| *User presses enter* ***and neither*** *text boxes are empty* ***and*** *username matches a user in database* ***and*** *password is correct:* | ***Clear Email/Password text boxes and change screen to the user dashboard.*** |
| *User presses register account button:* | ***Change screen to account registration menu.*** |
| *User presses show password button* ***and*** *password box is not empty* ***and*** *password is currently hidden:* | ***Unhide password (replace dashes with password characters).*** |
| *User presses show password button* ***and*** *password box is not empty* ***and*** *password is not hidden:* | ***Hide password (replace password characters with dashes).*** |
| *User presses show password button* ***and*** *password box is empty:* | ***Nothing happens.*** |

**USER DASHBOARD**

**|** AKA userDashboard()

|  |  |
| --- | --- |
| *User presses record entry* ***and*** *location textbox is empty:* | ***Display one or more textbox is empty popup.*** |
| *User presses record entry* ***and*** *date textbox is empty:* | ***Display one or more textbox is empty popup.*** |
| *User presses record entry* ***and*** *time textbox is empty:* | ***Display one or more textbox is empty popup.*** |
| *User presses record entry* ***and*** *time contains non-integer values:* | ***Display invalid time popup.*** |
| *User presses record entry* ***and*** *time contains only integer values* ***and*** *time > 2400:* | ***Display invalid time popup.*** |
| *User presses record entry* ***and*** *date contains non-integer values:* | ***Display invalid date popup.*** |
| *User presses record entry* ***and*** *date contains only integer values* ***and*** *date is not a valid value:* | ***Display invalid date popup.*** |
| *User presses record entry* ***and*** *location textbox contains text* ***and*** *date textbox contains only integers pertaining to an actual date value* ***and*** *time textbox contains only integers* ***and*** *time <= 2400:* | ***Clear location textbox, clear time text box, clear date textbox, add record to GUI, add record to database.*** |
| *User presses back:* | ***Change screen to landing screen.*** |
| *User presses account:* | ***Replace title of registration screen, fill out text boxes with user’s information, change screen to registration menu.*** |

**POPUP BOXES**

**|** AKA invalidPassword(), invalidDate(), invalidTime(), invalidPhoneNum(), invalidAddress(), invalidLocation(), invalidEmail(), invalidName()

|  |  |
| --- | --- |
| *User clicks anywhere on the popup box while it is visible:* | ***Nothing happens.*** |
| *User clicks while popup box is visible, and mouse is not over popup box:* | ***Popup box closes.*** |
| *User clicks anywhere on the popup box while it is* ***invisible****:* | ***Nothing happens, popup box should not interfere with GUI while invisible.*** |

COMPONENT TESTING EVIDENCE

ERROR TESTING

This section covers errors that prevented the program from running completely, both when compiling and when triggered mid-program.

**REGISTRATION SCREEN**

**|** AKA registrationScreen()

|  |  |  |  |
| --- | --- | --- | --- |
| ***Code causing error*** | ***Error message*** | ***Reason for error*** | ***Corrected code*** |
| show invalidRegNamePopup() | **Invalid Syntax** | *The kivy ‘show’ adds elements to the current window, the element that is being shown is specified with the ‘=’ symbol.* | show = invalidRegNamePopup() |
| elif False == ((self.phoneNumber.text) (this is a new line with no ‘.’ preceding it) isdigit()) or self.phoneNumber.text == "": | **Invalid Syntax** | *When continuing a line of python code onto a new line, a ‘.’ is required to mark that the two lines should be read as one (excluding certain circumstances, E.g. lists).* | elif False == ((self.phoneNumber.text).  isdigit()) or self.phoneNumber.text == "": |
| Empty .txt file | **SyntaxError: unexpected EOF while parsing** | *Users.txt was wiped after logging out of the program while signed in, this caused the program to try to get information of the still signed in user despite the fact that it did not exist in the .txt file referenced.* | .txt file was amended, no further changes were made because this situation would not happen outside of testing. |
| def submit(self): | **No Error Message** | *The function needed more information, it was not linked to the submit button because it was not connected to ‘widget’.* | def submit(self, widget): |

**LANDING SCREEN**

**|** AKA landingScreen()

|  |  |  |  |
| --- | --- | --- | --- |
| ***Code causing error*** | ***Error message*** | ***Reason for error*** | ***Corrected code*** |
| elif accessState = 2: | **Invalid Syntax** | *One ‘=’ when two are required for comparative statement* | elif accessState == 2: |
| from kivy.config import Configuration | **ImportError: cannot import name 'Configuration' from 'kivy.config'** | *The actual name of the module is ‘config’, not ‘configuration’. The python compiler did not find configuration because it does not exist.* | from kivy.config import Config |
| else tipVariable == 1: | **Invalid Syntax** | *Else can only be used to trigger an event when all other statements above are false.* | elif tipVariable == 1: |
| size=(0.6, 0.4)) | **No Error Message** | *This code is technically correct, however it made the element it is referring to unreadably small because, while Kivy uses relative size in decimals, python uses exact pixel measurements.* | size=(600,400)) |

**USER DASHBOARD**

**|** AKA userDashboard()

|  |  |  |  |
| --- | --- | --- | --- |
| ***Code causing error*** | ***Error message*** | ***Reason for error*** | ***Corrected code*** |
| Animation( | **UnboundLocalError: local variable 'button\_animate' referenced before assignment** | *Animation code must be assigned to a variable so that it can be called later.* | button\_animate = Animation( |
| @ not in self.location.text | **Invalid Syntax** | *Without quotation marks, python sees @ as an unassigned variable and not a string.* | "@" not in self.location.text |
| def submit(self): | **No Error Message** | *The function needed more information, it was not linked to the submit button because it was not connected to ‘widget’.* | def submit(self, widget): |
| content=”show” | **No Error Message** | *Content did was still hidden because ‘show’ is a Kivy state and not referenced by a string.* | content=show |

**POPUP BOXES**

**|** AKA invalidPassword(), invalidDate(), invalidTime(), invalidPhoneNum(), invalidAddress(), invalidLocation(), invalidEmail(), invalidName()

|  |  |  |  |
| --- | --- | --- | --- |
| ***Code causing error*** | ***Error message*** | ***Reason for error*** | ***Corrected code*** |
| size\_hint: 0.6 | **No Error Message** | *Sizing in kivy is specified using two values for width and height, rather than just one.* | size\_hint: 0.6, 0.2 |
| pos\_hint: 0.2, 0.7 | **No Error Message** | *The pos\_hint property is specified using a set of coordinates that must be accompanies by how the element should be aligned.* | pos\_hint: {"x":0.2, "top":0.7} |

UNEXPECTED OUTCOMES

This section covers errors that did not stop the program completely, but instead resulted in something other than the intended result. Test tables have been recorded below.

**REGISTRATION SCREEN**

**|** AKA registrationScreen()

|  |  |  |  |
| --- | --- | --- | --- |
| ***Input tested*** | ***What should happen*** | ***What did happen*** | ***Fixes needed*** |
| User presses register **and** Email box is empty: | Display all fields must be filled out popup. | Displayed all fields must be filled out popup. | No fix needed. |
| User presses register **and** password box is empty: | Display all fields must be filled out popup. | **Popup box displayed but the text is missing.** | **Ensure kivy text field is specified** |
| User presses register **and** phone number box is empty: | Display all fields must be filled out popup. | **Popup box not displayed.** | **Add code to call the popup box class.** |
| User presses register **and** full name box is empty: | Display all fields must be filled out popup. | Displayed all fields must be filled out popup. | No fix needed. |
| User presses register **and** address box is empty: | Display all fields must be filled out popup. | Displayed all fields must be filled out popup. | No fix needed. |
| User presses register **and** entered Email does not contain an ‘@’ symbol: | Display invalid Email popup. | **Popup box not displayed.** | **Add code to call the popup box class.** |
| User presses register **and** entered Email does not contain a ‘.’ symbol: | Display invalid Email popup. | Displayed invalid Email popup. | No fix needed. |
| User presses register **and** phone number contains any non-integer characters: | Display invalid phone number popup. | Displayed invalid phone number popup. | No fix needed. |
| User presses register **and** full name does not contain a space: | Display invalid full name popup. | Displayed invalid full name popup. | No fix needed. |
| User presses register **and** full name contains a number or character: | Display invalid full name popup. | Changed screen to landing screen. | No fix needed. |
| User presses register **and** address contains less than 5 letters or symbols: | Display invalid address popup. | Displayed invalid address popup. | No fix needed. |
| User presses back: | Change screen to landing screen. | Changed screen to landing screen. | No fix needed. |
| User presses register **and** address contains more than 4 letters or symbols **and** Email contains an ‘@’ symbol **and** Email contains a ‘.’ Symbol **and** phone number contains only integers **and** full name contains a space **and** full name contains letters **and** password box is not empty: | Change screen to user dashboard and add user to database. | **Changed screen to user dashboard, but user was added into the user dictionary in the wrong format (Email was listed after name, rather than before, causing the program to record the user’s full name where Email should be.)** | **Fix the formatting error by swapping the “Email” and “Name” variables in the formatting code.** |

**LANDING SCREEN**

**|** AKA landingScreen()

|  |  |  |  |
| --- | --- | --- | --- |
| ***Input tested*** | ***What should happen*** | ***What did happen*** | ***Fixes needed*** |
| User presses enter **and** email box is empty: | Display invalid Email/password popup. | Displayed invalid Email/password popup. | No fix needed. |
| User presses enter **and** password box is empty: | Display invalid Email/password popup. | Displayed invalid Email/password popup. | No fix needed. |
| User presses enter **and both** text boxes are empty: | Display invalid Email/password popup. | Displayed invalid Email/password popup. | No fix needed. |
| User presses enter **and neither** text boxes are empty **and** username does not match a user in database: | Display Email not recognized popup. | **Popup box not displayed.** | **Add code to call the popup box class.** |
| User presses enter **and neither** text boxes are empty **and** username matches a user in database **but** password does not match for that user: | Display incorrect password popup. | **Displayed invalid Email/password popup rather than incorrect password popup.** | **Change code to call the correct popup class.** |
| User presses enter **and neither** text boxes are empty **and** username matches a user in database **and** password is correct: | Clear Email/Password text boxes and change screen to the user dashboard. | Cleared Email/Password text boxes and change screen to the user dashboard. | No fix needed. |
| User presses register account button: | Change screen to account registration menu. | **Nothing happened.** | **Link register account button to screen.** |
| User presses show password button **and** password box is not empty **and** password is currently hidden: | Unhide password (replace dashes with password characters). | Unhid password (replaced dashes with password characters). | No fix needed. |
| User presses show password button **and** password box is not empty **and** password is not hidden: | Hide password (replace password characters with dashes). | Hid password (replaced password characters with dashes). | No fix needed. |
| User presses show password button **and** password box is empty: | Nothing happens. | Nothing happened. | No fix needed. |

**USER DASHBOARD**

**|** AKA userDashboard()

|  |  |  |  |
| --- | --- | --- | --- |
| ***Input tested*** | ***What should happen*** | ***What did happen*** | ***Fixes needed*** |
| User presses record entry **and** location textbox is empty: | Display one or more textbox is empty popup. | **Popup is displayed, but is unreadably small.** | **Amend size\_hint property in kivy code.** |
| User presses record entry **and** date textbox is empty: | Display one or more textbox is empty popup. | **Popup is displayed, but is unreadably small.** | **Amend size\_hint property in kivy code.** |
| User presses record entry **and** time textbox is empty: | Display one or more textbox is empty popup. | **Popup is displayed, but is unreadably small.** | **Amend size\_hint property in kivy code.** |
| User presses record entry **and** time contains non-integer values: | Display invalid time popup. | **Popup is displayed, but is unreadably small.** | **Amend size\_hint property in kivy code.** |
| User presses record entry **and** time contains only integer values **and** time > 2400: | Display invalid time popup. | **Popup is displayed, but is unreadably small.** | **Amend size\_hint property in kivy code.** |
| User presses record entry **and** date contains non-integer values: | Display invalid date popup. | **Popup box displayed but the text is missing.** | **Ensure kivy text field is specified** |
| User presses record entry **and** date contains only integer values **and** date is not a valid value: | Display invalid date popup. | Displayed invalid Email popup. | No fix needed. |
| User presses record entry **and** location textbox contains text **and** date textbox contains only integers pertaining to an actual date value **and** time textbox contains only integers **and** time <= 2400: | Clear location textbox, clear time text box, clear date textbox, add record to GUI, add record to database. | **Cleared location textbox, cleared time text box, cleared date textbox, added record to GUI, but did not add record to .txt file (database).** | **Fix incorrect file path, place the .txt file in the same root folder as the .py file.** |
| User presses back: | Change screen to landing screen. | Change screen to landing screen. | No fix needed. |
| User presses account: | Replace title of registration screen, fill out text boxes with user’s information, change screen to registration menu. | Replace title of registration screen, fill out text boxes with user’s information, change screen to registration menu. | No fix needed. |

**POPUP BOXES**

**|** AKA invalidPassword(), invalidDate(), invalidTime(), invalidPhoneNum(), invalidAddress(), invalidLocation(), invalidEmail(), invalidName()

|  |  |  |  |
| --- | --- | --- | --- |
| ***Input tested*** | ***What should happen*** | ***What did happen*** | ***Fixes needed*** |
| User clicks anywhere on the popup box while it is visible: | Nothing happens. | **Popup box did not stop user from interacting with hidden elements beneath the popup.** | **Change kivy class properties to prevent “click through”.** |
| User clicks while popup box is visible, and mouse is not over popup box: | Popup box closes. | Popup box closed. | No fix needed. |
| User clicks anywhere on the popup box while it is **invisible**: | Nothing happens, popup box should not interfere with GUI while invisible. | Nothing happened, popup box did not interfere with GUI while invisible. | No fix needed. |

TRIALING / USABILITY

This section covers trialing, usability feedback, and changes that were made to the program based on this feedback.

**REGISTRATION SCREEN**

**|** AKA registrationScreen()

|  |  |  |  |
| --- | --- | --- | --- |
| **Trial One** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“The gradient is very distracting in this GUI when combined with how the textboxes have been laid out”.* | The contrast between the two colours in this screen alone was reduced by taking a section of the gradient and stretching it to fit. We did not change the gradient used in other screens because this problem was specific to the registration screen. | **Graphical user interface, diagram  Description automatically generated with medium confidence** |  |
|  | | | |
| **Trial Two** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“During the up-scroll transition from the landing screen to the registration screen, you can see that the gradients do not line up. This creates a line between them that looks very unprofessional in terms of design”.* | The gradient background in the registration screen was flipped vertically in order to form a smooth, seamless connection to the landing screen. | ***Shape, rectangle  Description automatically generated*** | ***Shape  Description automatically generated*** |
| **Trial Three** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“The back button doesn’t animate when clicked, even though this animation would be seen for less than a second while the transition animation happens, it would make it seem more interactable”.* | Button was given the same “on click” animation as all other buttons, allowing the design of elements to be more consistent and allowing users to identify it more easily as a button. | **(not animated)** | ***A picture containing text, gear  Description automatically generated*(not animated)** |
| **Trial Four** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“The popup boxes that appear when I try to enter something incorrectly don’t give much information about what specifically is wrong, making it more difficult to figure out how I can solve the issue”.* | More specific popup boxes were implemented in this screen to give specific information about what is wrong with the user’s account details and how it can be fixed. | **Graphical user interface, text, application  Description automatically generated** | **Graphical user interface, text, application  Description automatically generated** |

**LANDING SCREEN**

**|** AKA landingScreen()

|  |  |  |  |
| --- | --- | --- | --- |
| **Trial One** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“Pure black and pure white are difficult to look at.”* | White and black elements were changed to light gray and dark gray in order to reduce the contrast, reducing eye strain and making it easier to look at. | Graphical user interface, text, application, letter  Description automatically generated |  |
| *“The gradient looks cool, but it would be less distracting if the two colours didn’t contrast as much.”* | The background gradient was edited to be less distracting, colours that are more alike were used to create the gradient. | A picture containing text, screenshot, businesscard  Description automatically generated | **Graphical user interface, diagram  Description automatically generated with medium confidence** |
| *“Text on the landing screen is too small.”* | The font size of text in the landing screen was increased from 25 to 40. | Text  Description automatically generated | **Graphical user interface, diagram  Description automatically generated with medium confidence** |
| **Trial Two** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“Dark gray and light gray look unappealing and boring; the GUI is less aesthetically pleasing”.* | The replacement colours were replaced with cooler variants of their monotone counterparts, with a slight tint of blue. |  | ***Text  Description automatically generated*** |
| **Trial Three** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“This could be hard to look at when it’s darker”.* | A night-specific colour theme was introduced that is far less bright than the original design, this was connected to the system time module to activate after 8pm. | **Graphical user interface, diagram  Description automatically generated with medium confidence** |  |
|  | | | |
| **Trial Four** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“The hide / unhide password icon doesn’t animate when clicked like all the other buttons, it doesn’t feel as interactive”.* | Button was given the same “on click” animation as all other buttons, allowing the design of elements to be more consistent and allowing users to identify it more easily as a button. | ***A picture containing black, night sky  Description automatically generated***  **(not animated)** | ***A picture containing black, night sky  Description automatically generatedA picture containing black, night sky  Description automatically generated*(Animated)** |

**USER DASHBOARD**

**|** AKA userDashboard()

|  |  |  |  |
| --- | --- | --- | --- |
| **Trial One** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“White text on a yellow background is difficult to read.”* | The font colour of header text in the dashboard was changed to make it easier to read. | ***A picture containing graphical user interface  Description automatically generated*** |  |
| *“The hello user message comes across as quite cold and unwelcoming, it would be better if it said something more personal and maybe more enthusiastic.”* | The welcome message was changed completely to read “*good morning”, “hello”, “good afternoon”, ‘good evening”,* and *“good night”* followed by the user’s first name based on the time of day. |  | ***Diagram  Description automatically generated*** |
| *“Pure black and pure white are difficult to look at.”* | White and black elements were changed to light gray and dark gray in order to reduce the contrast, reducing eye strain and making it easier to look at. | Graphical user interface  Description automatically generated with medium confidence |  |
| **Trial Two** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“Entering the date and time is repetitive and makes it take longer to enter your whereabouts.”* | Automatic date and time features were implemented based off the device date and time. | ***Shape, square  Description automatically generated*** | ***Shape, square  Description automatically generated*** |
| *“Dark gray and light gray look unappealing and boring; the GUI is less aesthetically pleasing”.* | The replacement colours were replaced with cooler variants of their monotone counterparts, with a slight tint of blue. |  | ***Logo  Description automatically generated with medium confidence*** |
| **Trial Three** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“The buttons are small and might be difficult to press on a smaller phone like mine.”* | The buttons were increased in size dramatically, making them significantly easier to press regardless of device specifications or motor ability. | ***Diagram  Description automatically generated*** |  |
| **Trial Four** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“Headings don’t really grab my attention; I have to look at it for a bit before I know what to do.”* | Headings in the dashboard were doubled in size and a bolder font was used. |  | ***A picture containing text  Description automatically generated*** |

**POPUP BOXES**

**|** AKA invalidPassword(), invalidDate(), invalidTime(), invalidPhoneNum(), invalidAddress(), invalidLocation(), invalidEmail(), invalidName()

|  |  |  |  |
| --- | --- | --- | --- |
| **Trial One** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“Popup box text is too small and is difficult to read”.* | Popup box text was doubled in size. |  | ***Graphical user interface, text, application  Description automatically generated*** |
| **Trial Two** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“The popup boxes are too large, this makes it difficult to close them as this can only be done by clicking outside of the box”.* | The width of the popup box was reduced from 0.9 to 0.6. |  | ***Graphical user interface  Description automatically generated*** |
|  | | | |
| **Trial Three** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| *“The popup box works well; I can’t think of any further feedback”.* | No changes were needed. | No changes were needed. | No changes were needed. |
| **Trial Four** | | | |
| ***Feedback*** | ***What was changed*** | ***Before change*** | ***After change*** |
| No further feedback was given. | No changes were needed. | No changes were needed. | No changes were needed. |

**COMPLETE PROGRAM**

COMPLETE PROGRAM TEST PLAN

|  |  |
| --- | --- |
| *User successfully creates an account:* | ***All information contained within their account is correctly added to users.txt in the format {0:[users information in the form of a list],1:[next user’s information in a list],2:[Etc.]}*** |
| *User successfully records an entry:* | ***Their entry is added to activity.txt in the same format as user information, with date, time and location concatenated with ‘!’ symbols to preserve spaces.*** |
| *User successfully signs in:* | ***User’s recent activity is retrieved from activity.txt and all ‘!” are replace with ‘/’.*** |
| *User enter program:* | ***Current COVID-19 stats are retrieved from the John Hopkins database and presented in the GUI.*** |

COMPLETE PROGRAM TESTING EVIDENCE

ERROR TESTING

|  |  |  |  |
| --- | --- | --- | --- |
| ***Code causing error*** | ***Error message*** | ***Reason for error*** | ***Corrected code*** |
| Empty .txt file | **SyntaxError: unexpected EOF while parsing** | *Users.txt was wiped after logging out of the program while signed in, this caused the program to try to get information of the still signed in user despite the fact that it did not exist in the .txt file referenced.* | .txt file was amended, no further changes were made because this situation would not happen outside of testing. |
| from kivy.config import Configuration | **ImportError: cannot import name 'Configuration' from 'kivy.config'** | *The actual name of the module is ‘config’, not ‘configuration’. The python compiler did not find configuration because it does not exist.* | from kivy.config import Config |
| def submit(self): | **No Error Message** | *The function needed more information, it was not linked to the submit button because it was not connected to ‘widget’.* | def submit(self, widget): |

UNEXPECTED OUTCOMES

|  |  |  |  |
| --- | --- | --- | --- |
| ***Input tested*** | ***What should happen*** | ***What did happen*** | ***Fixes needed*** |
| User successfully creates an account: | All information contained within their account is correctly added to users.txt in the format {0:[users information in the form of a list],1:[next user’s information in a list],2:[Etc.]} | **Information contained within their account was added to users.txt, however it was recorded with a ‘,’ at the start, when this information was extracted, that ‘,’ was concatenated onto the start, making it difficult for the user to log in.** | **Prevent the program from recording the user’s account, which is supposed to be a list, as a string by using the AST module.** |
| User successfully records an entry: | Their entry is added to activity.txt in the same format as user information, with date, time and location concatenated with ‘!’ symbols to preserve spaces. | Their entry was added to activity.txt in the same format as user information, with date, time and location concatenated with ‘!’ symbols to preserve spaces. | No fix needed. |
| User successfully signs in: | User’s recent activity is retrieved from activity.txt and all ‘!” are replace with ‘/’. | User’s recent activity was retrieved from activity.txt and all ‘!” are replace with ‘/’. | No fix needed. |
| User enter program: | Current COVID-19 stats are retrieved from the John Hopkins database and presented in the GUI. | **JHU database is accessed, but the extracted information is not added to the GUI.** | **Move the code responsible for adding this text into the GUI back so that text is added before the GUI is built.** |

TRIALING / USABILITY

Trialing and usability had been done extensively prior and, while we attempted to get feedback during this phase, we weren’t able to get any because all of the main improvements had been made during component development. The only usability improvement that was made during this phase was reducing the speed of the scrolling transition between screens because one of the testers mentioned that “*it could potentially make someone susceptible to motion sickness feel ill”*. Nevertheless, the program was trialed further with a wide range of people. Mostly parents and friends, individuals were asked to trial the program based on their unique background and needs. One tester was chosen because of their visual impairments, and another was chosen because English was not their first language. By trialing the program with these individuals, we were able to ensure that our wording was not difficult to understand for someone with limited experience of the English language (less than 5 years). We were also able to make sure that elements, particularly text, were large enough and featured enough contrast for visually impaired users to easily use the program. Ultimately, all 11 testers were happy that the program could be usable by the majority of possible users.

PEP8 COMPLIANCE:

Initially, the program had over 100 PEP8 compliance issues because it was not written with PEP8 in mind. Instead, it was checked repeatedly against an online PEP8 checker for these issues and modified retroactively to comply with PEP8 guidelines while still maintaining the same functionality.

Graphical user interface, application, table

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated***Before PEP8 Modifications: After PEP8 Modifications:***

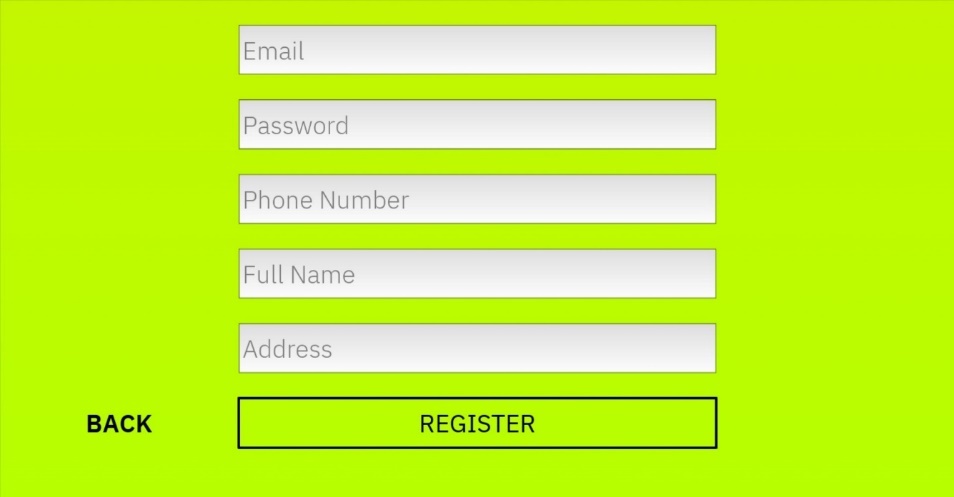
TESTING EVIDENCE:

<https://drive.google.com/file/d/1TANwhQFYeNSkRHRYIAoUAaOWzPZ50IA5/view?usp=sharing>

COMPLETE PROGRAM SCREENSHOTS:

**Graphical user interface, diagram

Description automatically generated with medium confidence**

****

**

TEST AGAINST REQUIREMENTS:

In this section I have listed each of the requirements set out during my initial inquiry, followed by the testing done (where applicable) to ensure this requirement was met, and the outcome of this testing.

**The application must be intuitive and easy to use for a wide variety of people from different backgrounds and with differing technical experience.**

*The application was trialed and tested with people from a wide range of backgrounds, including a few individuals (my parents) who claimed that they are not very tech savvy. Usability changes were made based on their feedback such as utilizing consistent styles and animations in order to make the application usable to people with very little technical knowledge. After these changes were made, the application was trialed again, and more feedback was given. Eventually, all of my clients were happy with the usability of the program.*

**The application must be able to record the time, date, and an individual’s location effectively.**

*This requirement has been met. Automatic date and time features were implemented during trailing, in conjunction with the ‘stay signed in’ feature this allows the program to be very time effective when recording the time, date and location. After timing different users logging their location (excluding sign in because the user stays signed), the times ranged from* ***1.4*** *to* ***8.1*** *seconds, and averaged at around* ***3*** *seconds. This is well within an acceptable range of values needed to meet this requirement.*

**The application must be able to produce case reports quickly in order to optimize the contact isolation process.**

*A contact tracking authority dashboard was added that allows tracking services to immediately retrieve all entries added by a user. This requirement has been met.*

**The application must be compatible with IOS and Android.**

*Kivy was chosen for this application because of it’s continued developer support and compatibility with a wide range of OS, including* ***IOS****,* ***Android****, Windows 10 and MacOS.*

**Modifying the application’s code to suit a community’s needs must be relatively easy.**

*Comments and sufficient line spacing have been used consistently throughout the program, this allows it to be easily modified.*

**The application’s GUI must be visually appealing and professional.**

*Research was done into design techniques used for similar programs, this culminated in a professional and visually appealing GUI. Changes were made during trialing to make it more visually appealing, such as lining up the gradients to create smooth, seamless transitions between screens.*

**Copyright and privacy must be addressed**

*All resources used were investigated prior in order to make sure this requirement was met.*

**Information about the user’s privacy and what is done with the information they enter into the application must be easily accessible.**

*This requirement would be met later if the application were to be published, however it is not relevant to the application in its current state because the user’s private information is not used by the app. This requirement relates to each individual party intending to use the program.*

RELEVANT IMPLICATIONS ADDRESSED

**Accessibility**

|  |  |  |  |
| --- | --- | --- | --- |
| In order to make my GUI colour-blind friendly, I have tested all UI colour combinations before implementing them using the [Coblis colour blind simulator](https://www.color-blindness.com/coblis-color-blindness-simulator/). I specifically investigated whether the contrast between the colours a colour blind user would see is sufficient, as if this were not the case the application would be significantly more difficult for a colour blind person to use. This investigation process was repeated with each type of colour-blindness and each colour scheme I had planned on using in the final GUI. I had to adjust the colours used in some of the colour schemes slightly in order to make them more easily visible when viewed by people with certain colour-blindness. 3 of the 40 tests have been shown to the right, with a circle of the original colour scheme for comparison. | | **Graphical user interface  Description automatically generatedGraphical user interface  Description automatically generatedGraphical user interface  Description automatically generated** | |
| Not all users with a visual impairment are colour blind, however. In fact, the majority of visual impairments are minor. 43% of all visual impairments are caused by refractive errors in the lens of the eye while a further 33% are attributed to cataracts. While these users still retain most of their visual abilities, they may be unable to identify or position their cursor over smaller or less visually obvious elements. In order to account for the widest range of users, I have made all elements of the GUI large enough and with enough tone and colour contrast to be easily identifiable by users with minor visual impairments. Making these elements larger also assists users with limited motor skills, as positioning the cursor over a larger object is far less challenging than positioning it over the element’s smaller alternative. I have chosen to use bold, easily readable fonts in conjunction with contrasting colours and appropriate font size and spacing in order to make text easily readable for all users. | | | |
| Graphical user interface, diagram  Description automatically generated with medium confidenceLarge interactive elements | Large / bold fonts  Graphical user interface, diagram  Description automatically generated with medium confidence | | Graphical user interface, diagram  Description automatically generated with medium confidenceContrast considered in colour scheme |
| Finally, I have used tactile GUI features such as button animations and responsive visual changes to user input in order to make using the program more intuitive to users with limited or otherwise different cognitive abilities. | | | |

**Usability**

|  |  |
| --- | --- |
| Buttons and text inputs within the program adhere to two different and easily recognizable styles, making them easy for users to identify and distinguish from each other. Additionally, text boxes include temporary text explaining their function to the user, allowing information to be entered effectively. As mentioned previously, all buttons within the program have a single, consistent pressing animations in order to provide tactile feedback to the user, reinforcing the idea that all buttons within the program can be interacted with in the same way. As mentioned in the previous section, I have chosen to use bold, easily readable fonts in conjunction with contrasting colours and appropriate font size and spacing in order to make text easily readable for all users. | A picture containing graphical user interface  Description automatically generated  Text boxes:  Gray gradient background and temporary text with a black outline. Semitransparent temporary text reflects the idea that the text is not permanent and is supposed to be edited.  A picture containing graphical user interface  Description automatically generated  Buttons:  Colour-filled box with capitalized black text.  The opaque, capitalized text is a ‘call to action’ and demonstrates that the text cannot be changed. The coloured background is unique to buttons and is used consistently to differentiate them from text boxes. |
| IBM Plex Sans, the font seen above and used for all text in my outcome, is a free font commonly used in modern municiple designs to reflect professionalism. It was chosen because it is very easy to read when obscured and from a distance. | |
| Date and time textboxes automatically fill based on device date and time in order to streamline the location recording process. These text boxes can be altered, however, as this allows users to accurately record their activity after an event if they do not have the chance to do so during the event. | Graphical user interface, diagram  Description automatically generated with medium confidenceAutomatic Time Automatic Date |
|  |  |
| Specific, detailed popups inform the user exactly what they must do next when an invalid input error occurs, even providing details on how times can be changed to the 24-hour time format for users that do not know how to use this format. These popups increase the overall usability of the program by ensuring users do not get lost when something goes wrong, and instead know exactly how to rectify the problem each time. The three error message popups from the user dashboard are shown to the right. One of these textboxes appears when the user tries to submit an invalid data entry. Each of them briefly explains what the problem with the user’s entry is and how they can fix it. | Text  Description automatically generatedGraphical user interface, text, application  Description automatically generatedGraphical user interface, text, application  Description automatically generated |
|  |  |
| Finally, I have chosen to use the F-layout of elements on the page, placing important elements in a vertical list on the left of the page in descending order of importance. This allows the majority of users to immediately identify key features of the GUI. | |

**Functionality**

In order to increase functionality, I considered how I could increase the chance of users interacting with the tracking system. As discussed in **Addressing Relevant Implications**, appealing to the logical part of the user’s brain and increasing the perceived formality of the application has a large effect on this. To do this, I have chosen colour schemes and design conventions commonly found in municipal applications and websites, such as minimalist, modern GUI features and basic but appealing colour schemes, utilizing a single colour gradient in conjunction with dark gray and off-white.

**Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generatedGraphical user interface, PowerPoint

Description automatically generatedGraphical user interface, diagram

Description automatically generated with medium confidenceGraphical user interface

Description automatically generated**

Additionally, I have adopted a professional style of writing that avoids the use of unnecessary jargon or technical language. These features make the program more usable while ensuring that it conveys its own professionalism. Personal benefit has also been identified as a key factor which increases the likelihood of an individual participating in the contact tracing process. If an individual can gain something by participating, for example entry to an establishment, they are far more likely to participate. Adding some form of personal gain or euphoric response can increase the effectiveness of a contact tracing system. I took advantage of this effect by adding tactile responses to the entry process and providing a euphoric response to the user in the form of their most recent entries being added to the GUI in the style of a leaderboard. During my inquiry I also found that contact tracing systems rely on the ability to respond quickly to a case, and that the efficiency of a system will be reduced if it is not time optimized as the timeframe the infected individual has to spread the virus before they are isolated is increased. My program reduces the time taken for authorities to locate and contact a potentially contagious individual A picture containing table

Description automatically generatedby allowing administrators to search for a user’s records effectively, returning all their records along with their contact details almost immediately. Additionally, users can be located by information contained within their profile, meaning that even if contact tracing services do not know an individual’s name, they can still quickly find all necessary information needed to prevent the virus’ spread.

**Health and Safety**

Firstly, the program must work well in order to prevent the spread of COVID-19 and protect the health of program user’s and the rest of the human race alike. I have discussed how I have optimized the effectiveness of my digital technology outcome during **Functionality**. I have investigated multiple sources to ensure that the information the application provides surrounding proper COVID-19 procedures are accurate, thereby ensuring that users follow all the necessary precautions properly to prevent the spread of COVID-19. Outside of functionality considerations, I have also considered any possible negative health effects that a user may encounter while using the outcome, such as digital eye strain or overexposure to bright light. In order to mitigate the effects of eye strain, I have considered the size of all elements in the GUI, especially text, so that users do not damage their iris while viewing GUI features and reading GUI information. Additionally, I have implemented five different colour schemes that change automatically depending on the time of day that the application is opened. Each colour scheme is designed for a specific brightness and to contain different amounts of blue light. For example, the colour scheme that the user will get during the night contains a dark gray background with light gray text, whereas the mid-day colour scheme utilizes a white background with black text. Regardless of what time the user opens the application, the overall light emitted by the GUI will match their environment and their natural body clock. This prevents excessive amounts of light from entering the eye and damaging the retina. Absorption of blue light by the eyes has been linked to preventing the secretion of hormones that would usually allow us to sleep. Absorbing excess blue light during the evening can have a profound effect on our ability to sleep, however absorbing excess blue light before bed can delay the release of hormones massively because our internal body clock is ignored in favor of what our body believes to be daylight. This can lead to changes in our sleep pattern and delayed tiredness, usually resulting in a harmful lack of sleep. In order to avoid this, each colour scheme has been designed to reduce the user’s blue light intake gradually throughout the day, changing the background gradient colour from green, to blue, to orange, to red and eventually to a dark gray colour gradient containing almost no blue light.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A picture containing text, sign, room, gambling house  Description automatically generated | Icon  Description automatically generated with medium confidence | Icon  Description automatically generated with low confidence | A red circle with black numbers  Description automatically generated with low confidence | Icon  Description automatically generated |
| **Morning Theme**  *Brightest*  *42.7% Blue Light* | **Mid-Day Theme**  *High Brightness*  *31.6% Blue Light* | **Afternoon Theme**  *Med Brightness 18.0% Blue Light* | **Evening Theme**  *Low Brightness*  *5.3% Blue Light* | **Night Theme**  *Darkest*  *4.1% Blue Light* |

* *Blue light is calculated by dividing the RGB blue value by 255.*

**COMPLEX PROCESSES**

EVALUATE

Throughout the planning and development process many different complex features were used to ensure that the digital technology outcome produced was of a high standard. By planning out how long each activity should take beforehand I was able to stay on top of my work during the development process; working more during certain periods where necessary to stick to the deadlines I had set out earlier.

**By formatting this information into a Gantt chart**, I was able to facilitate more effective communication between myself and my partner. The Gantt chart was also chosen because it can be edited easily, by using this type of chart we were able to change the timeline quickly and easily during development once we had a better idea of how long tasks should take. **By using** **Trello**, I was able to manage milestones more effectively as I completed and altered them. Trello features the ability to move milestones between *to do*, *in progress* and *done* sections, making it ideal for recording progress and changing milestones throughout development. This is one of Trello’s most significant features because, unlike most other management tools, it allows objectives to be marked as in progress again if they need to be updated or reintegrated. Trello also supports multiple users in different locations because it is a web-based application, this allowed for better communication between myself and my partner when tracking our progress. By improving communication between myself and my partner with these processes, the consistency and effectiveness of each component when integrated into the final outcome was improved, thereby making the final outcome of a higher standard.

**By creating different versions of the layout and design** and then collecting and collating feedback I was able to ensure that the most effective GUI was chosen, this improved the quality of the final outcome and meant less code would have to be rewritten after trialing the outcome.

**Research was done into the key considerations of the outcome** including usability, functionality, health and safety and accessibility of the program. By considering these beforehand I was able to inform the design of the outcome. This meant that key considerations could be effectively addressed in the outcome and properly integrated.

**The program was then planned out fully and decomposed** into individual components, this assisted with time management and reduced the chance that me or my partner would have too much work to complete within the timeframe. By doing this we ensured that no parts of the program were rushed, allowing every part of the program to be consistently high quality.

**During development the schedule was amended** to make the timeframe for each component more realistic based on what we learned during development. This allowed us to stay on top of our work and ensure that no components were rushed out at a lower quality.

**Additionally, information collected during consideration research was used** to inform the design of the outcome. This information was critical in making sure the program was able to fulfill its purpose effectively, as the userbase encompasses people of all backgrounds. Feedback collected during GUI design was also used to create the most effective GUI possible.

**Throughout and after development the program was tested and trialed** so that any usability features, errors, and unexpected results were properly addressed. This was done very extensively and was assisted by the feedback collected during GUI design. This was a crucial step in the development process because it allowed us to get ideas and viewpoints we would not have been able to consider otherwise, such as how usable the program is for someone with visual impairments or someone who is relatively new to the English language. This feedback was incorporated into the next iteration of the program, allowing the program to be usable by the wide range of users needed for contact tracking to effectively happen. Initially, individual components were tested and trialed, meaning that the most suitable components were identified before being integrated into the outcome. Once integrated, the entire program was tested and trialed to make sure that the integration of components had been done properly, ensuring that the outcome was developed to a high standard.

**Next, the program was tested against for PEP8 compliance and edited where necessary** to make it easier to work with for an outside party. As a result of these changes, the program is significantly easier to modify by its users. This is important because it makes the outcome compliant with the industry standard while making it a lot more usable by users with specific needs such as alternate language options.

**Finally, the program was tested against its requirements** to ensure that we had produced a high-quality solution for the problem we had set out to solve. This was assisted largely by the feedback collected during testing and trialing, allowing us to consider a wide range of users.

PROJECT WALK THROUGH

(TAKEN FROM TESTING EVIDENCE)

[Program Demonstration](https://drive.google.com/file/d/1TANwhQFYeNSkRHRYIAoUAaOWzPZ50IA5/view?usp=sharing)

[A picture containing text

Description automatically generated](https://drive.google.com/file/d/1TANwhQFYeNSkRHRYIAoUAaOWzPZ50IA5/view?usp=sharing)