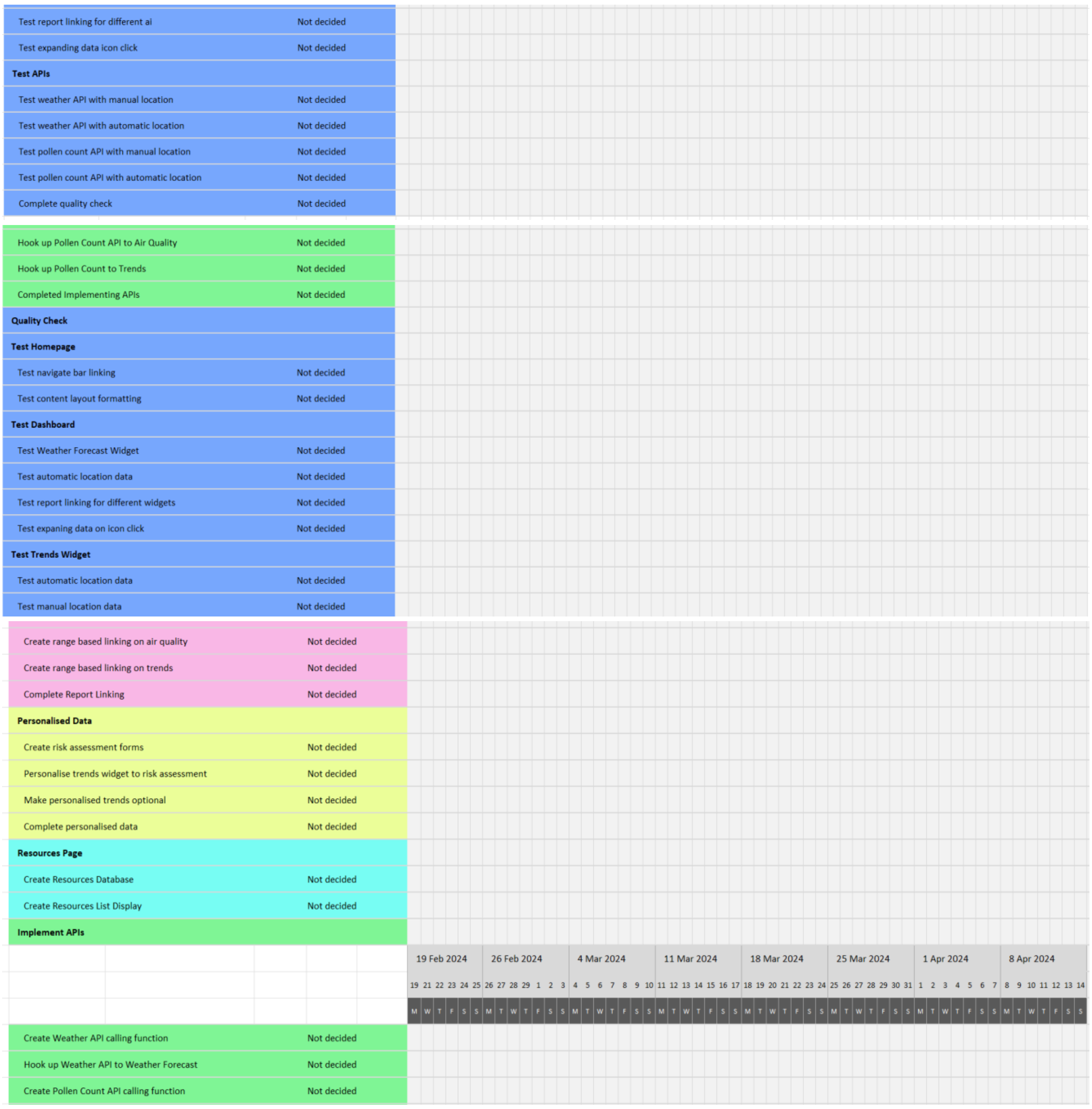
DESIGNS

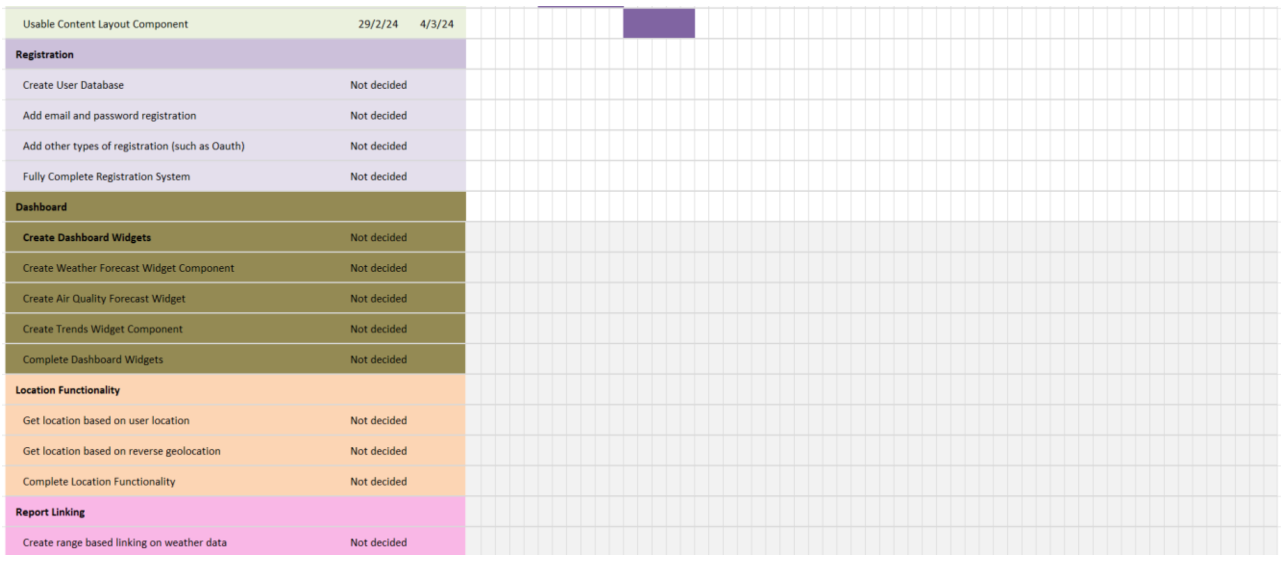
Health Advice Group

Gantt Chart

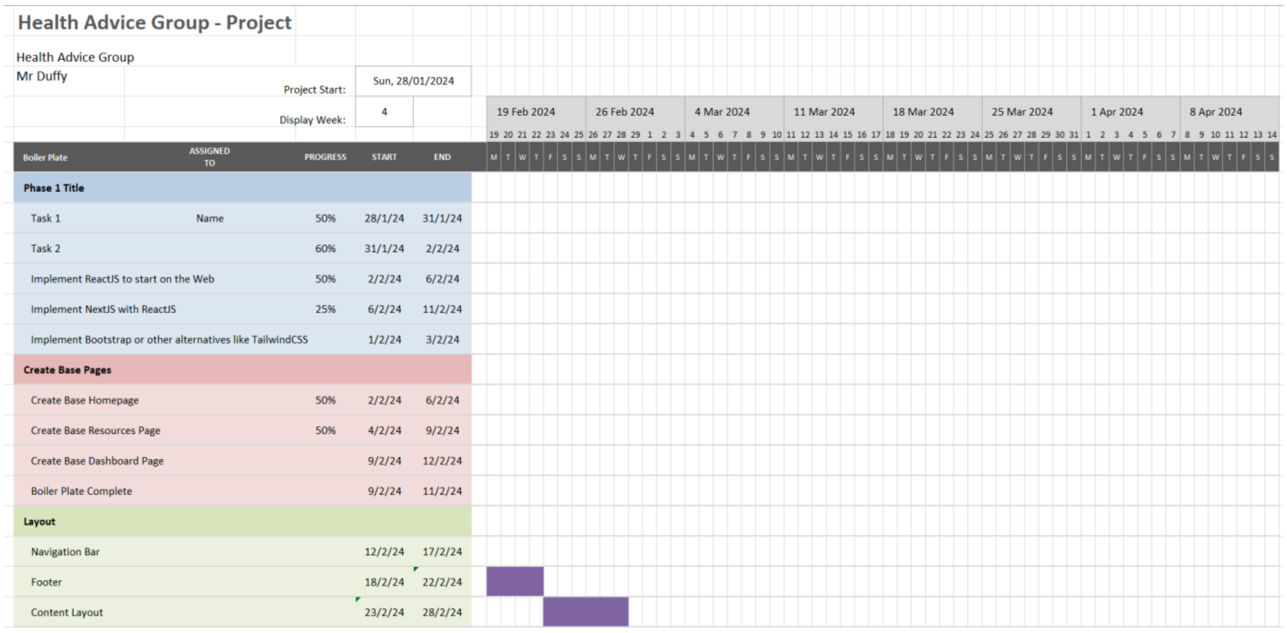


Test report linking for different ai	Not decided
Test expanding data icon click	Not decided
Test APIs	
Test weather API with manual location	Not decided
Test weather API with automatic location	Not decided
Test pollen count API with manual location	Not decided
Test pollen count API with automatic location	Not decided
Complete quality check	Not decided
Hook up Pollen Count API to Air Quality	Not decided
Hook up Pollen Count to Trends	Not decided
Completed Implementing APIs	Not decided
Quality Check	
Test Homepage	
Test navigate bar linking	Not decided
Test content layout formatting	Not decided
Test Dashboard	
Test Weather Forecast Widget	Not decided
Test automatic location data	Not decided
Test report linking for different widgets	Not decided
Test expaning data on icon click	Not decided
Test Trends Widget	
Test automatic location data	Not decided
Test manual location data	Not decided
Create range based linking on air quality	Not decided
Create range based linking on trends	Not decided
Complete Report Linking	Not decided
Personalised Data	
Create risk assessment forms	Not decided
Personalise trends widget to risk assessment	Not decided
Make personalised trends optional	Not decided
Complete personalised data	Not decided
Resources Page	
Create Resources Database	Not decided
Create Resources List Display	Not decided
Implement APIs	
Create Weather API calling function	Not decided
Hook up Weather API to Weather Forecast	Not decided
Create Pollen Count API calling function	Not decided

19 Feb 2024						26 Feb 2024							4 Mar 2024							11 Mar 2024							18 Mar 2024							25 Mar 2024							1 Apr 2024							8 Apr 2024						
19	21	22	23	24	25	26	27	28	29	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14
M	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S



Usable Content Layout Component	29/2/24	4/3/24
Registration		
Create User Database	Not decided	
Add email and password registration	Not decided	
Add other types of registration (such as Oauth)	Not decided	
Fully Complete Registration System	Not decided	
Dashboard		
Create Dashboard Widgets	Not decided	
Create Weather Forecast Widget Component	Not decided	
Create Air Quality Forecast Widget	Not decided	
Create Trends Widget Component	Not decided	
Complete Dashboard Widgets	Not decided	
Location Functionality		
Get location based on user location	Not decided	
Get location based on reverse geolocation	Not decided	
Complete Location Functionality	Not decided	
Report Linking		
Create range based linking on weather data	Not decided	



Health Advice Group - Project

Health Advice Group
Mr Duffy

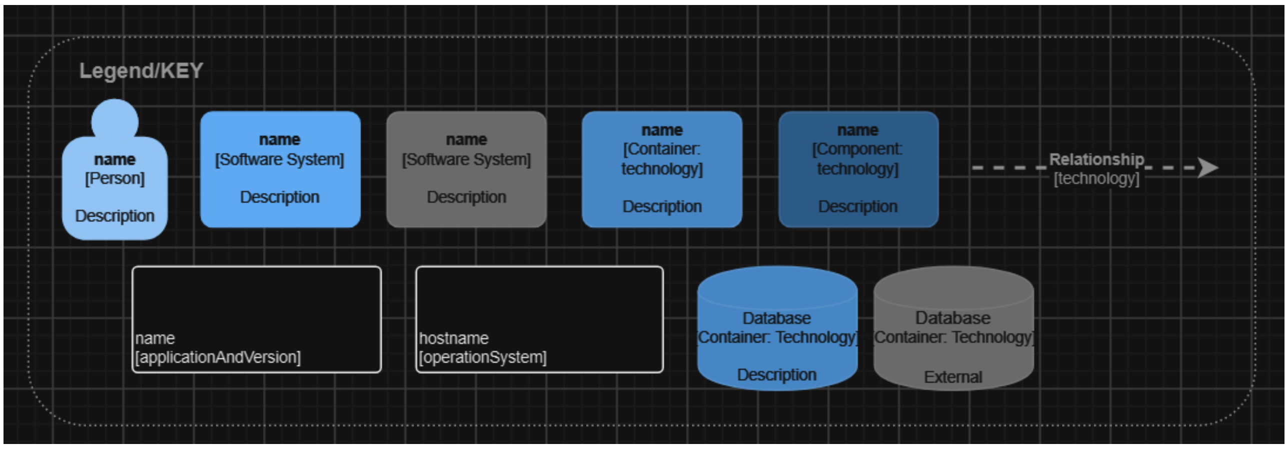
Project Start: Sun, 28/01/2024
Display Week: 4

19 Feb 2024 | 26 Feb 2024 | 4 Mar 2024 | 11 Mar 2024 | 18 Mar 2024 | 25 Mar 2024 | 1 Apr 2024 | 8 Apr 2024

Boiler Plate	ASSIGNED TO	PROGRESS	START	END
Phase 1 Title				
Task 1	Name	50%	28/1/24	31/1/24
Task 2		60%	31/1/24	2/2/24
Implement ReactJS to start on the Web		50%	2/2/24	6/2/24
Implement NextJS with ReactJS		25%	6/2/24	11/2/24
Implement Bootstrap or other alternatives like TailwindCSS			1/2/24	3/2/24
Create Base Pages				
Create Base Homepage		50%	2/2/24	6/2/24
Create Base Resources Page		50%	4/2/24	9/2/24
Create Base Dashboard Page			9/2/24	12/2/24
Boiler Plate Complete			9/2/24	11/2/24
Layout				
Navigation Bar			12/2/24	17/2/24
Footer			18/2/24	22/2/24
Content Layout			23/2/24	28/2/24

C4 Model

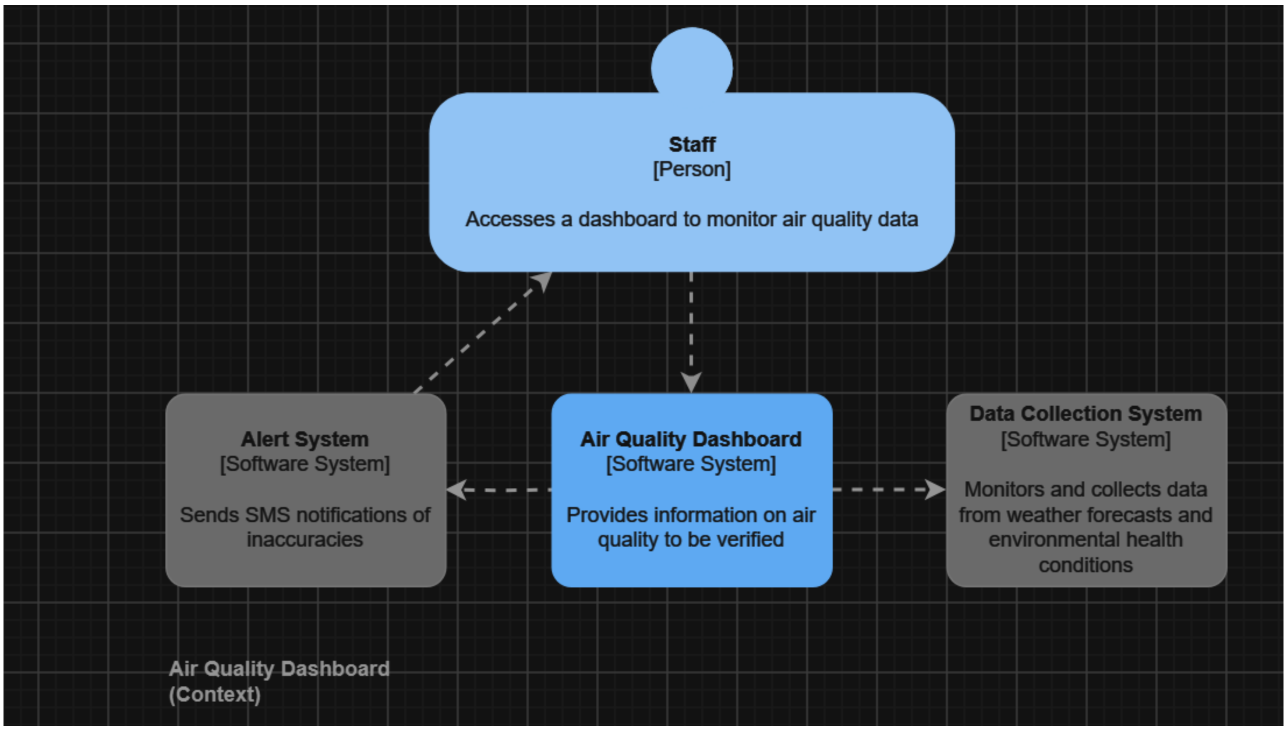
Legend



Before getting into the designs shown below, it is important to note that the C4 model consists of 4 levels, these being: Context, Container, Component and Class, all of which being more in-depth than the last level.

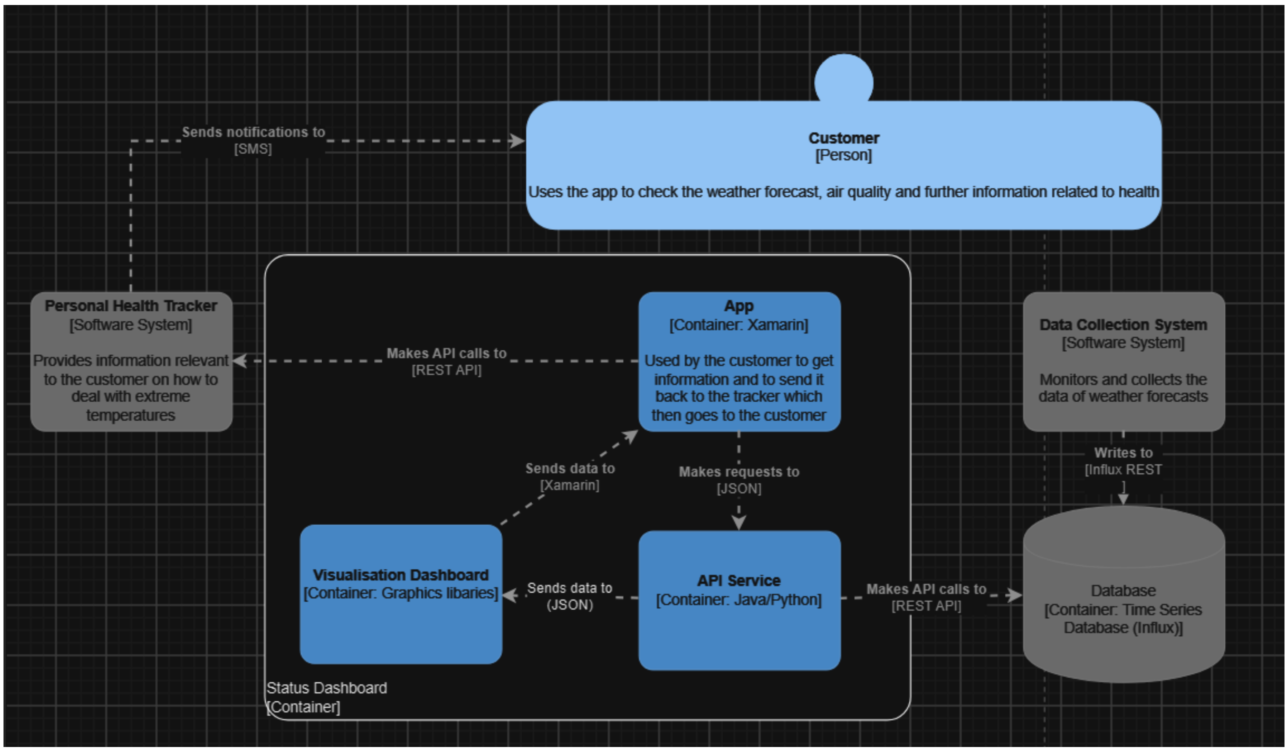
Here displays the keys to consider in the C4 model, this will help viewers understand which element is which such as identifying a person, database and/or software system.

Context


Staff
[Person]
Accesses a dashboard to monitor air quality data
Alert System
[Software System]
Sends SMS notifications of inaccuracies
Air Quality Dashboard
[Software System]
Provides information on air quality to be verified
Data Collection System
[Software System]
Monitors and collects data from weather forecasts and environmental health conditions
Air Quality Dashboard
(Context)

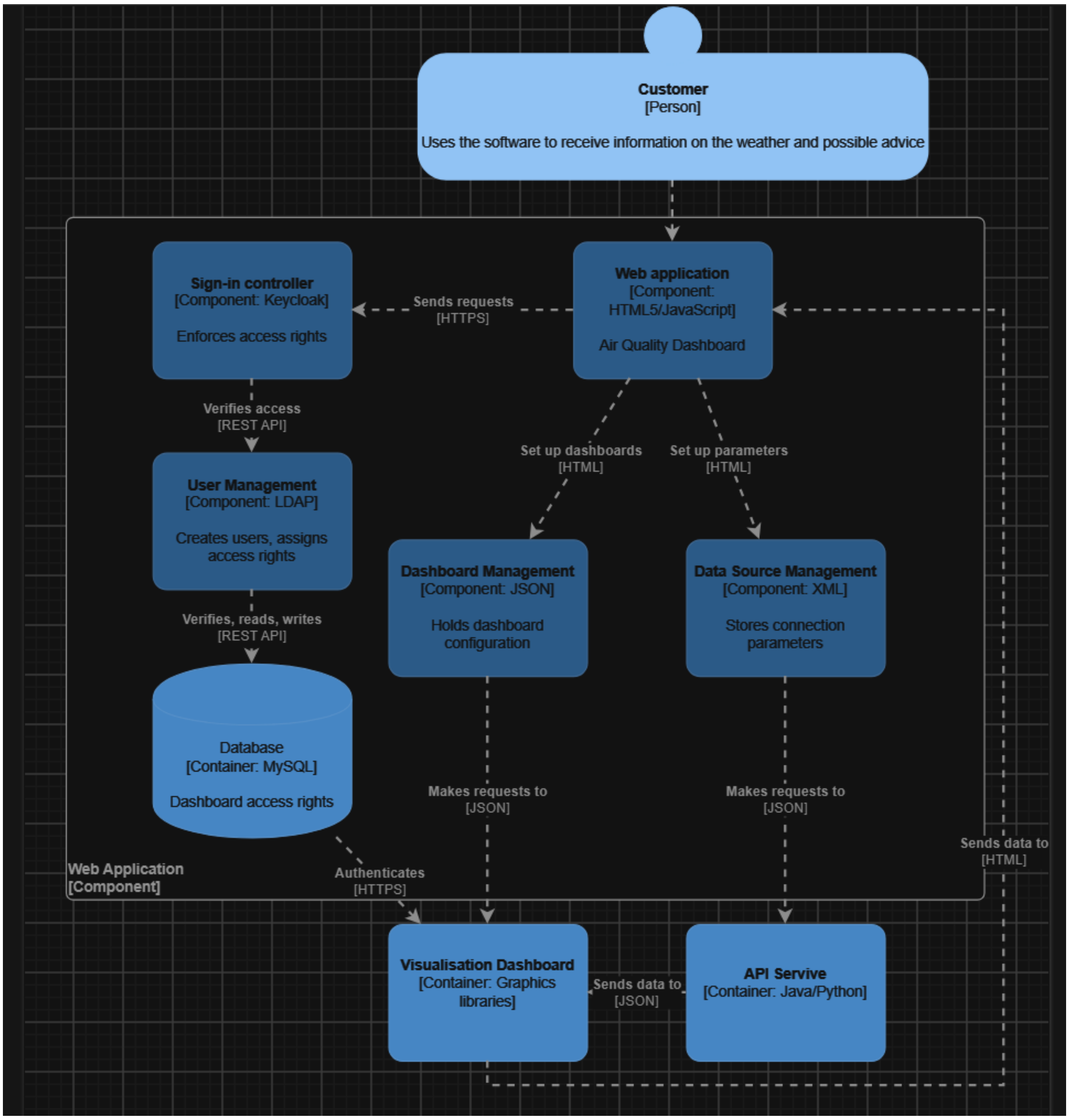
Here is the first level of the C4 model Context, this level contains how the overall application/system will function, it gives a very basic understanding of how it will operate. As shown in the image, staff will use the air quality dashboard to collect data and the dashboard will send any inaccuracies through an alert system back to the staff member.

Container


Sends notifications to [SMS]
Customer [Person]
Uses the app to check the weather forecast, air quality and further information related to health
Personal Health Tracker [Software System]
Provides information relevant to the customer on how to deal with extreme temperatures
Makes API calls to [REST API]
App [Container: Xamarin]
Used by the customer to get information and to send it back to the tracker which then goes to the customer
Data Collection System [Software System]
Monitors and collects the data of weather forecasts
Sends data to [Xamarin]
Makes requests to [JSON]
Writes to [Influx REST]
Visualisation Dashboard [Container: Graphics libaries]
Sends data to (JSON)
API Service [Container: Java/Python]
Makes API calls to [REST API]
Database [Container: Time Series Database (Influx)]
Status Dashboard [Container]

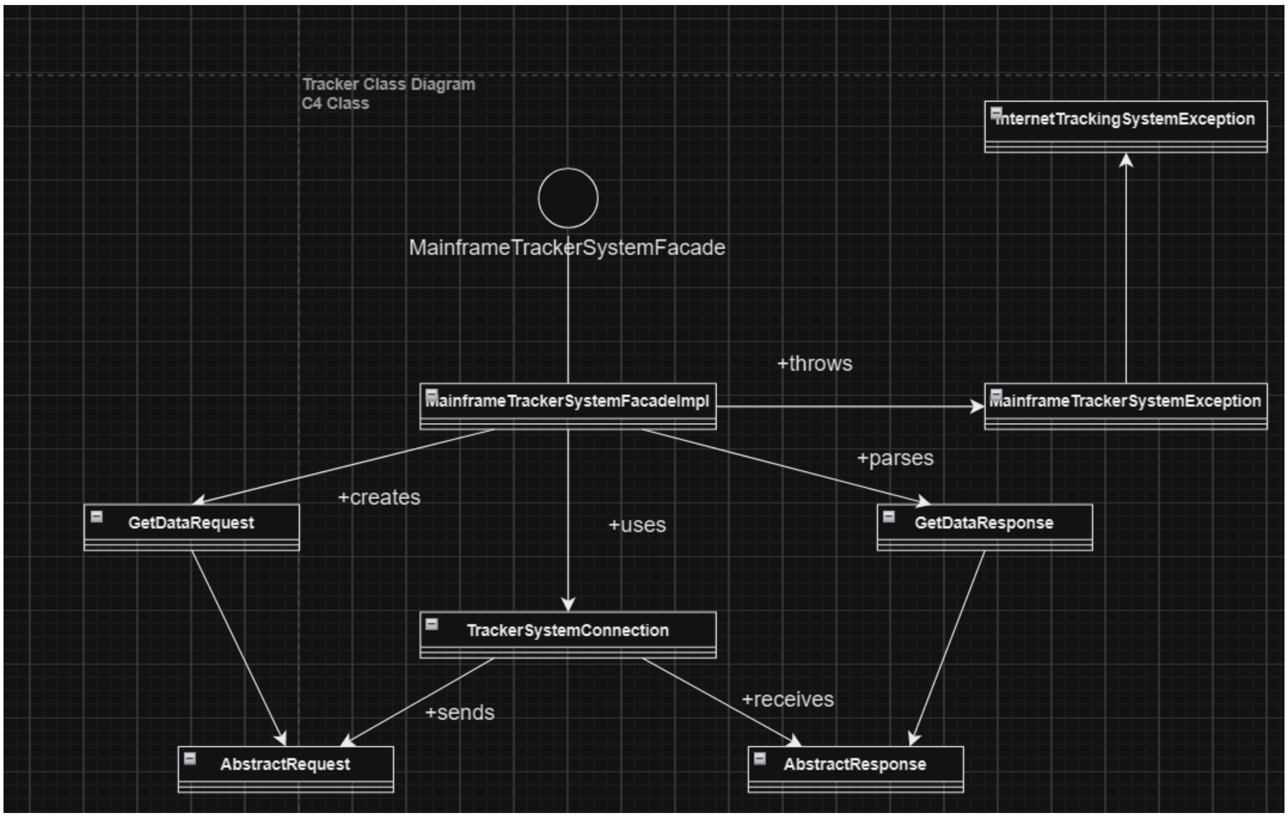
Here is the second level of the C4 model Container, this level provides a more-in depth look at how the system will function for the user as well as how the system provides information. As shown in the image, the application will make requests to an API service which makes a call to the database as the data collection system writes to the database. Once the call has been made, data is sent to a visualisation dashboard which is then returned back to the app. Once the app has received the data, another API call is made to the relevant health tracker and sends notifications back to the user. All of this happens when the user chooses to gain information.

Component



Here is the third level of the C4 model Component, this level provides an in-depth look at how users can sign-in and what action is taken in-order for that event to be completed. As shown in the image, the user opens the application whether through the web or by a downloaded version tries to sign-in. Once complied, the application will set-up dashboards and parameters as well as make requests to an API service and dashboard (data is also sent to the dashboard through the application and API service). The application will also send requests to the sign-in controller which enforces access rights, this access is then verified with the user management and is thus verified, read and write in the database.

Class

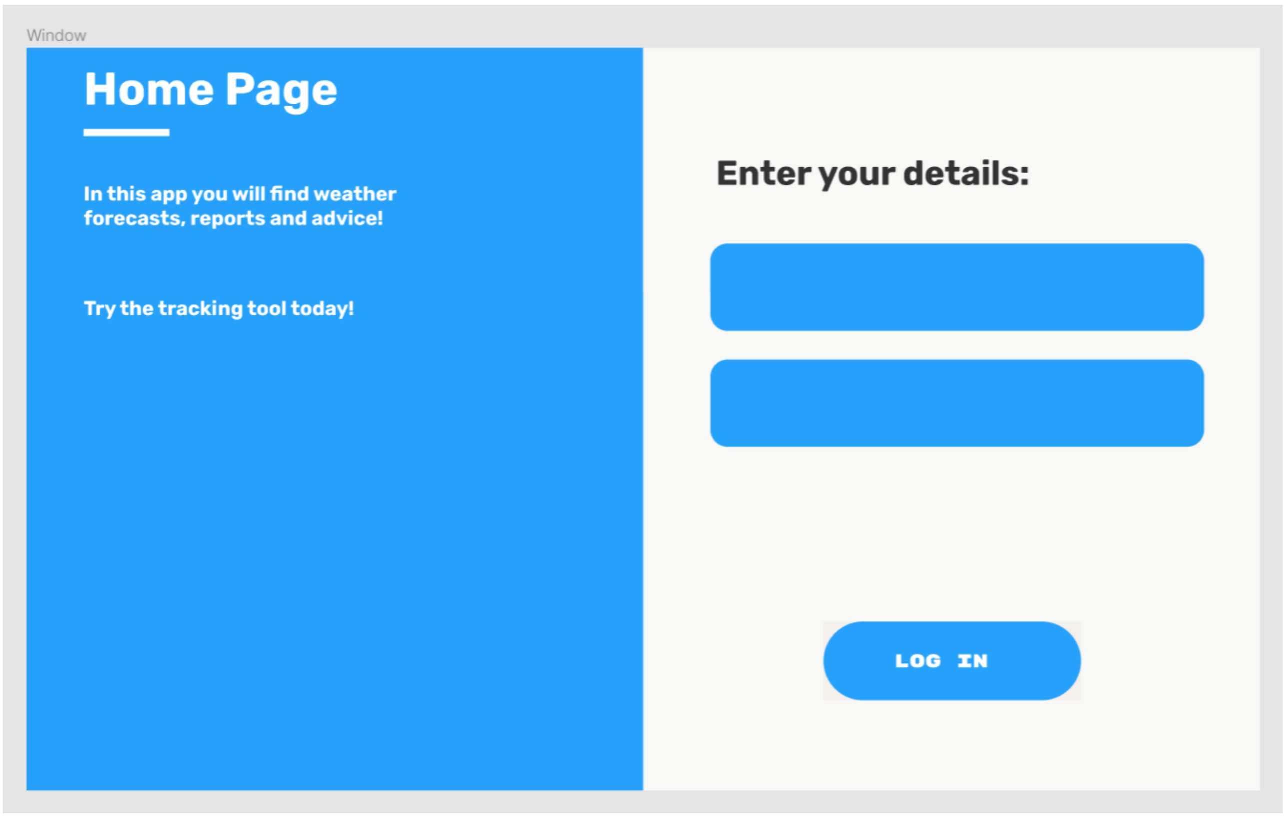


Here is the last level of the C4 model Class, although not as important as the other levels it still holds some benefit as to how the system will function as code. Essentially, this is how the tracker will send requests and receive responses for data through a connection, which is then sent back to the tracker and thrown to the exceptions made.

Proposed Design Interface

Here will be the proposed designs of the application, they have been designed through Figma as there is a variety of tools that can be used on the platform which help improve designs even further compared to other platforms. Once the platform was chosen, it was important to decide the aesthetics of the app as many users would be drawn to an eye-catching app thus determining the overall design. When it comes to the final designs, they were kept informal, playful and had colours to match that as it is important to stand out from other applications.

The Home Page:

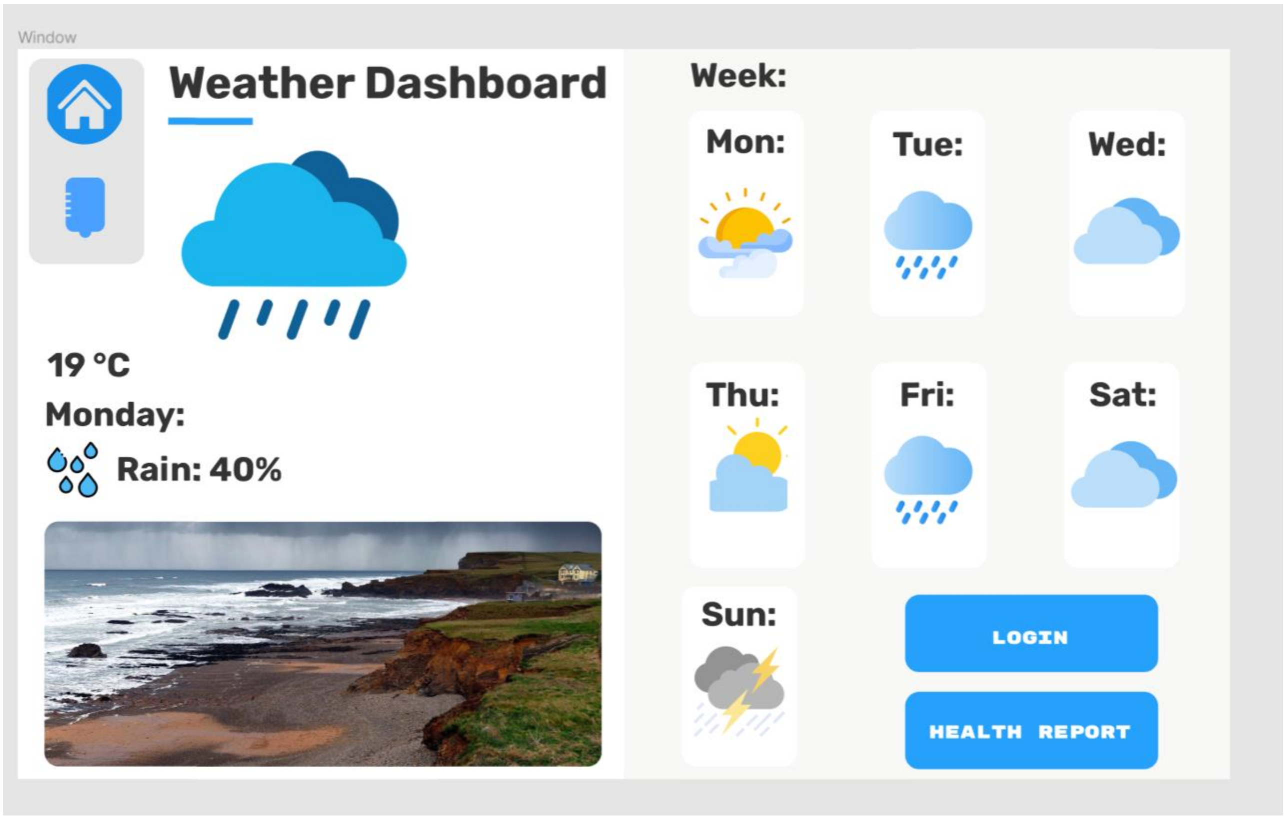

Window
Home Page
In this app you will find weather forecasts, reports and advice!
Try the tracking tool today!
Enter your details:
LOG IN

As mentioned previously it was decided that the designs of the application would be kept informal, meaning bright colours and language to make users feel welcome rather than intimidated. The home page is kept simple as there is no need to add content to it besides to log in. Once the user is logged in then would be redirected to the dashboard. The home page will only direct users to the dashboard, from there users can choose to go to other parts of the application such as the health reports and air quality sections.

Visual hierarchy has also been taken note as shown in the designs, key information has larger text making it more noticeable to the user. Another example of this would be that the username and password boxes are above the log-in button, to show that users are required to fill in their information first before using the application, the boxes of the username and password have been made larger to reinforce this reason as well.

Contrasted colours is also prevalent in the designs as shown with the black text and white background, this also includes the dashboard page. Space between elements has also been used to draw in the eyes of users to signify the importance of the elements shown such as the log-in button.

Dashboard:



As seen above, this is the dashboard where users will be able to see the days and what the weather will be like for each day. Not only that, but it will also show the temperature and percentage of rain in your area through a tracker which the user would have to allow access to for it to be initialized. Icons are also used to give users an easier time navigating the application, the health icon will direct the user to health reports relevant to them. Lastly, the air quality as well as other relevant information will be provided in another window to prevent overloads of information that may disinterest users. The dashboard will use an API to gain and output information on the weather, the content of the dashboard will also contain dynamic elements relevant to the user.

Code Examples

To give an idea of what the code will look like, here are a few examples:

Canvas

```
Canvas = Canvas
Window,
Bg = #FFFFFF,
Height = 519,
Width = 862,
Bd = 0
```

Button

```
Button = assets(button.png)
Image = button_image,
Borderwidth = 0,
Highlightthickness=0,
Command = callback (file)
Relief=”flat”
```

Button Place

```
Button.place
X= 631.0,
Y= 388.0,
Width = 180.0,
Height = 55.0
```

The reason to why the dashboard will not contain content related to health reports and air quality is to prevent bloats of code for one window as each window will include their own python file for example, the dashboard, making it easier to keep track of code and changing code whenever required. Due to many elements such as buttons and images being duplicated, it would be easy to reuse some of the code for different python files thus reducing the time required for development.

The repetition of styles in the dashboard window is frequent as shown through the weather icons, to display to users that they are inter-related and is also mirrored through them being closely placed together.

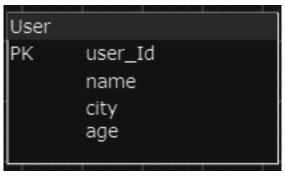
Common conventions

It should also be noted that the designs of the application follow common conventions for graphics design such as the simplicity shown in the designs, to prioritize certain elements included thus providing users with the most necessary features, information, and components.

As mentioned previously, hierarchy has also been considered for the designs as multiple elements would be included, deeming it important to prioritize certain elements over others in-order to provide weight to them. White space, referring to the areas of the design that does not include text, elements or images has also been prioritized to improve the flow of the overall design and avoid clutter. By following this, it allows to convey messages more effectively throughout the application. In simple terms, the more whitespace is prevalent, the more users would be drawn into the design.

Database User Table

Before getting into the binary tree algorithm, it is important to get an idea of what the database will look and behave like:



User	
PK	user_Id
	name
	city
	age

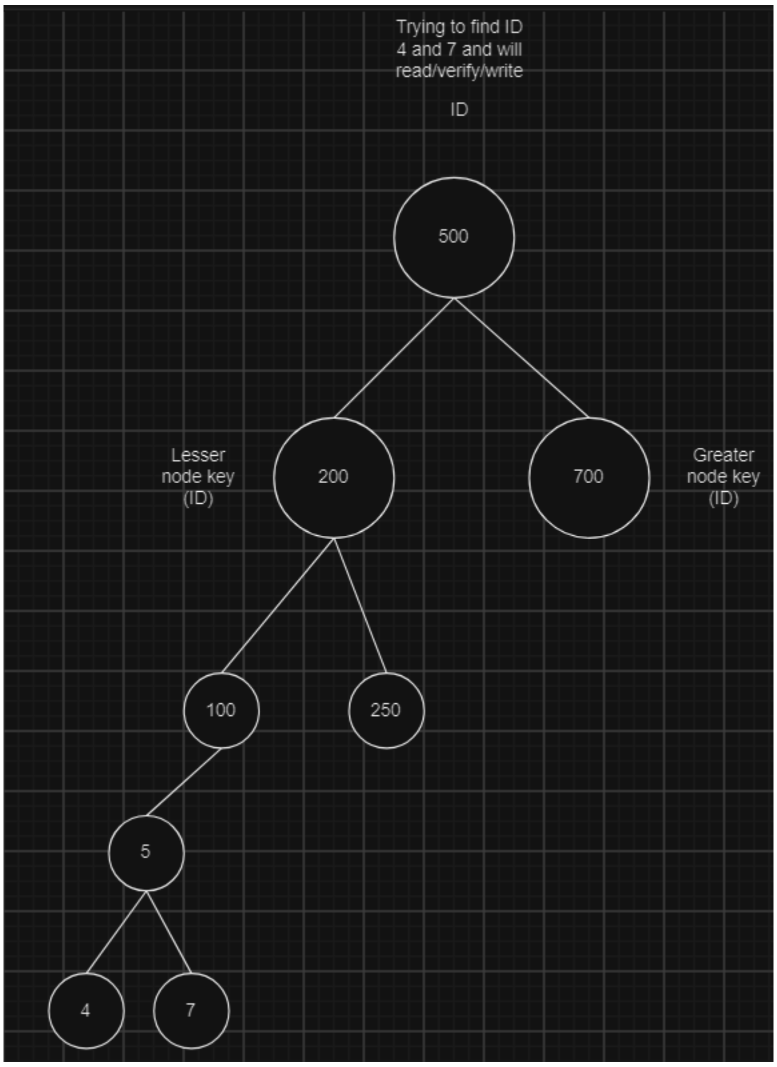
As the system does not require a lot of information from the user, the user table will be kept simple due to it being a weather application.

Interval Search/Binary Tree Algorithm

Firstly, before getting into the search algorithm it is important to note that the application will also be dealing with a database as mentioned in previous designs shown. Each user will be provided with an ID for a more efficient way of tracking whenever needed, this is important when an action is executed such as the user logging in or requesting to receive information through the application.

As the database would be dealing with structured data, an interval search algorithm would be ideal for tasks in-order to help them be executed.

The reason why interval search was chosen is due to its efficiency when applied to similar scenarios as compared to sequential searches. And would be closer to what would be expected when these tasks are executed.



The image above shows a task where the database needs to find ID 4 and 7 and then complete any action relevant such as reading, writing and/or verifying. It can be seen on the left that the key (id) is less than the original and the right includes keys greater than the original, this will be important when searching for the two IDs asked for. The image displays how a database search would behave, it is important to show this as a database will be implemented to handle a significant amount of data/records. The binary tree search is like a hierarchy by searching for ID's high and low, following a similar principle.

Data requirements for the proposed solution

Variables Table

As the application will be handling many elements, it is important to take note of variable names in-order to not get confused during development. A benefit of including a variable table would be due to the duplication of elements, with the access to a table of variable names it would make it easy to track and reuse code for other files related to development or possibly different sections of a file.

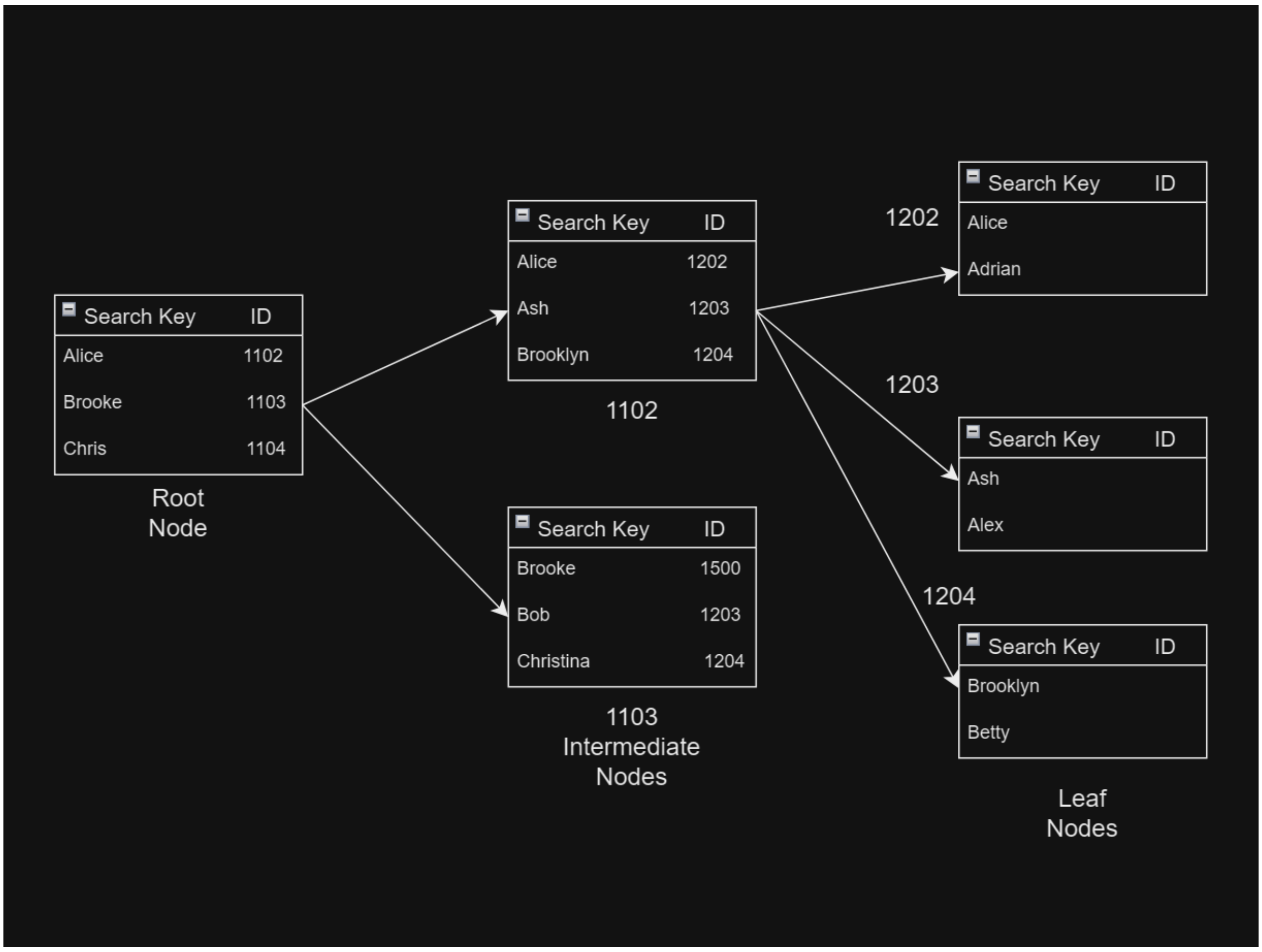
Variable Name	Function
Text	Creates text
Rectangle	Creates a rectangle
Canvas	Creates a canvas
Canvas.place	Places the relevant canvas
File	Opens a file
Button_image	Opens a button image through the use of the file variable
Button	Creates button
Button.place	Places button
Entry_image	Opens a entry image through the use of the file variable
Entry	Creates entry
Entry.place	Places entry

As many elements can be reused there is no need for many variables, keeping files clean and code structured both saving time in development and makes it easy for code to be modified.

Data Structures

Firstly, a static data structure is fixed, the content of the data structure can be modified but without changing the memory space allocated to it.

Design:



Here will be how a static data structure behaves in the database relevant to the application. It will handle root, intermediate and leaf nodes.

Data Types

Firstly, data types are the classification or categorization of data items and represent the kind of value that tells what operations can be performed on a particular data.

The common data types that will be used in the application include strings, integers and floats as the code will not be too different, lacking in other types to use such as Boolean and complex for example. Numeric data types will mainly be used to set the height, width, and area of objects while strings will be used to declare object names and paths. Names for variables will be appropriate easy to follow before the programming starts.

Components that should be tested - order

Testing components in order will be important as there are many systems being linked for the app such as the database and other files.

The first component that should be tested is the home page to make sure that the tools being used will work efficiently or at all. The home page will be linked to the dashboard file as well as the database for users, the file will also include minimal features taking less time to test.

The second component to test would be the dashboard as the respective file will include health reports, widgets and other important content. It is necessary to test if these features work as intended as to be ready for deployment and/or fix existing bugs.

The third component to test will be the air quality window as it will include the quality of air and visual representations of data. It is specifically important to check if the visual representations of data (graphs) will work as intended, if not then it can be fixed to work as intended.

Lastly, the component that should be checked last would be the database as it is not as important as the app, it has already been decided that the database will be serverless due to time constraints thus making the database easier to set up. If there are any issues, then they can be fixed quickly to be used in conjunction with the application.

Type of tests required

Firstly, before getting into the types of testing it is important to note that testing is the process of executing a program to find errors. To make software perform well it should be error-free. If testing is done successfully it will remove all the errors from the software.

Principles:

- All tests should meet customer requirements.
- Testing should be performed by a third party.
- Exhaustive testing is not possible, optimal amount of testing is needing as it will be based on the risk assessment of the application.
- All tests to be conducted should be planned before implementation.
- Start by testing small parts and extend it to large parts.

Types:

- Unit testing: type of software testing that focuses on individual units or components of a software system.
- Integration testing: the process of testing the interface between two software units or modules.
- System testing: a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution.
- Functional testing: a type of software testing in which the system is tested against the functional requirements and specifications.
- Acceptance testing: is a method of software testing where a system is tested for acceptability.
- Smoke testing: is a type of software testing that is typically performed at the beginning of the development process to ensure that the most critical functions of a software application are working correctly.
- Regression testing: is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made.
- Security testing: is a type of software testing that uncovers vulnerabilities in the system and determines that the data and resources of the system are protected from possible intruders.
- User Acceptance testing: is a testing methodology where clients/end users participate in product testing to validate the product against their requirements.

Required

The types of testing required include unit testing as it would make it easier to check each component or aspect of the project to fix errors or add missing features.

Secondly, integration testing as the application will include files being interlinked with each other, the application will also have to be tested on multiple different devices.

Thirdly, system testing as to give a good overview of the application to see what went well, what needs to be improved and/or updated.

Functional testing will also have to be considered as the client provided their set of functional requirements for the application, end users (customers) will also have expectations for the app. Smoke testing will not be needed due to the implementation of functional testing as well as how the plan and designs have been made, making it easier to go back to previous code in order to change and implement features.

Regression testing may not be needed due to the small size of the application as well as other tools providing the ability to test modified code and go back to previous versions anyway. As the application will not include sensitive information, security testing will not be a priority.

Lastly, user acceptance testing plays an important part of the project as the client and end users will need to participate in the testing of the application in order to give validation of requirements.