Plant Recommendation Program

Computer Science NEA

Name: *Thomas Knight*
Candidate Number: *1266*

Centre Name: Barton Peveril College
Centre Number: 58231

*****Note - the section and subheadings below are given as a guide only. It may be more appropriate for your project to use different subheadings. All projects must have the same 5 chapter headings: Analysis, Design, Technical Solution, Testing, Evaluation*****

*****Your final report should not contain these red notices. Make sure that you delete them*****

- [x] ~~13th December Testing video due~~
- [ ] 6th January deadline for full write-up draft due for feedback
  Analysis, Design, Testing and Evaluation
- [ ] 9th February FINAL DEADLINE ALL COURSEWORK IN
  This will include/be a printed version of all the work

0 Contents

1 Analysis

1.1 Statement of Problem

I intend to solve the confusing and frustrating choice plant enthusiasts have when it comes to purchasing and maintaining a new plant. I wish to design, prototype, troubleshoot and finally create a smooth and functional Plant Recommendation Program.

By providing a recommendation program that considers the majority of factors affecting plant health and user satisfaction (main 3 being: soil moisture, sunlight, temperature), it should solve the main difficulties at the moment, where people wishing to purchase a plant may look up their conditions in order to narrow down the plant or go to a garden centre to browse their plant collection to find their perfect plant. However, due to the inconsistencies across websites and recommendations, it is common for a plant to be purchased and disappoint the customer, wasting precious time and money.

Additionally, some customers may not have the time to take care of their plants and are more just looking for an aesthetically pleasing alternative to the time-consuming plant a "green-fingered" enthusiast may want. Therefore, by providing a recommendation program that takes into account all of this, the problem can be solved and customer satisfaction can increase as they will have the information needed to grow a plant to be green and healthy their entire life.

This program is aimed at general consumers who are interested in purchasing plants, however it could be used at businesses to aid workers trying to sell their plants or just find the right plant to keep in stock based on the data of the environment in which the store is located.

1.2 Background

There are about 390 thousand plant species out there, so the chances of someone finding their 'perfect' plant is very low, however with a program designed to find the optimal plant for them, it could recommend a list of the top n number of plants that suits them, greatly increasing the chances they find a plant they like first time, every time.
Clearly it is extremely difficult to take into account every single plant known to humankind, so the plant api I am using will include as many as is possible for it. I believe that this number is a little over 3000 of the most popular and most common plants, as I'm confident that this will be capable of providing sufficient options for the user.

1.3 End User

My end user is ___. I believe they are a suitable end user, due to their extensive experience in hobbyist plant-growth. Their consistent interaction with plants would be significantly beneficial to the program's development as this means feedback and support would come from a knowledgeable source.

Additionally due to my close contact with the end user, I would be capable of immediate responses and accelerated development due to the instant recommendations.

//NEED TO DO MORE

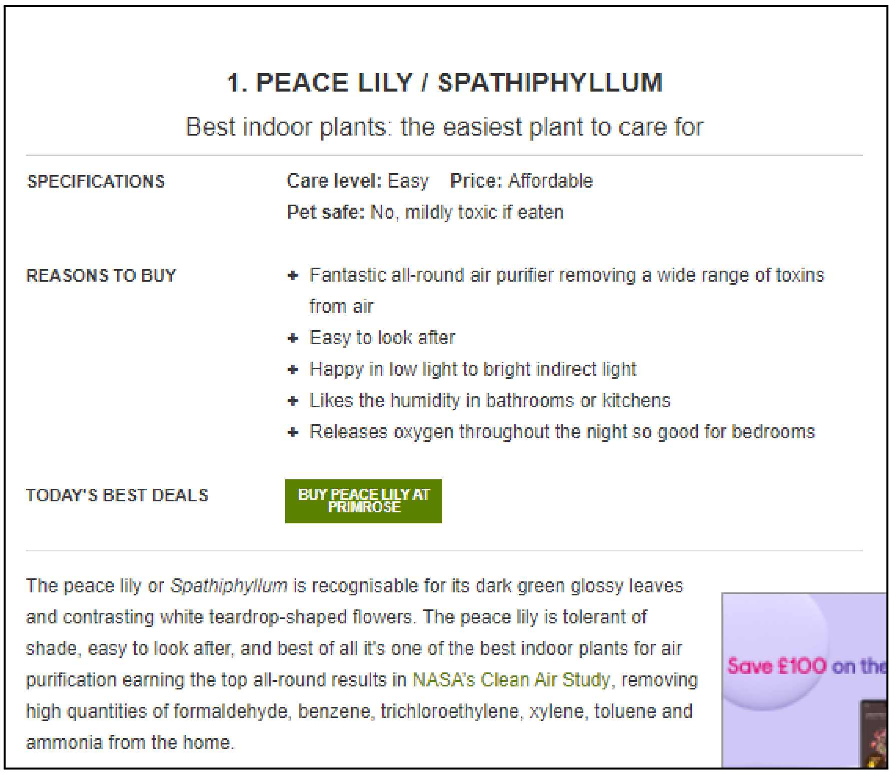
1.4 Initial Research

1.4.1 Existing, similar programs

Usually when looking for recommendations for plants, people end up reading through websites / forums. I couldn't find any programs designed for indoor plant recommendations. It appeared that agricultural recommendation programs were available to help with harvest times and various variables relating to that, however these programs could not be considered similar enough to be comparable to my project. I could only find websites.

Of the websites I found, there were none that required entering any personalised environment information, so the recommendations found are all general - even more so given my searching in an in-private browsing tab. A few of these websites are shown below:
Search query was "indoor house plant recommendations"
The first 3 sites (non-ad) [07/06/2022 - 11:30AM, microsoft edge in-private browsing tab]:

1. https://www.gardeningetc.com/buying-guides/best-indoor-plants



1. PEACE LILY / SPATHIPHYLLUM

Best indoor plants: the easiest plant to care for

SPECIFICATIONS	**Care level:** Easy **Price:** Affordable **Pet safe:** No, mildly toxic if eaten
REASONS TO BUY	+ Fantastic all-round air purifier removing a wide range of toxins from air + Easy to look after + Happy in low light to bright indirect light + Likes the humidity in bathrooms or kitchens + Releases oxygen throughout the night so good for bedrooms
TODAY'S BEST DEALS	BUY PEACE LILY AT PRIMROSE

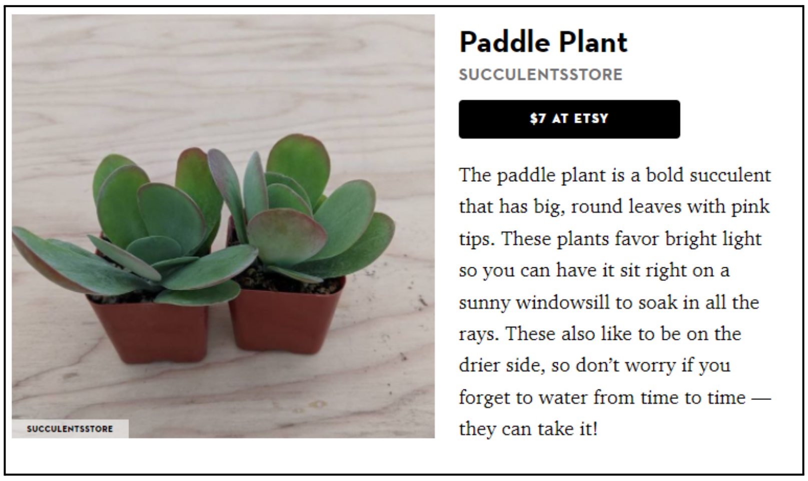
The peace lily or *Spathiphyllum* is recognisable for its dark green glossy leaves and contrasting white teardrop-shaped flowers. The peace lily is tolerant of shade, easy to look after, and best of all it's one of the best indoor plants for air purification earning the top all-round results in NASA's Clean Air Study, removing high quantities of formaldehyde, benzene, trichloroethylene, xylene, toluene and ammonia from the home.

Figure 1

The image above shows the top recommendation from this website. This recommendation requires no input on the user's side so it is not specific to the user in any way, however it provides extra information on the plant that would help satisfy the user. The site gives reasonable detail in terms of the plant's needs, such as humidity and sunlight, however there is no unit available to actually comprehend what that means, just a simple "likes the humidity in bathrooms or kitchens" - this can leave quite significant ambiguity for the user to try and determine.

Other than this, the website provides quite reasonable information on the plant, including care level, which is very useful for people looking for plants.

2. https://www.goodhousekeeping.com/home/gardening/advice/g1285/hard-to-kill-plants/



Paddle Plant

SUCCULENTSSTORE

$7 AT ETSY

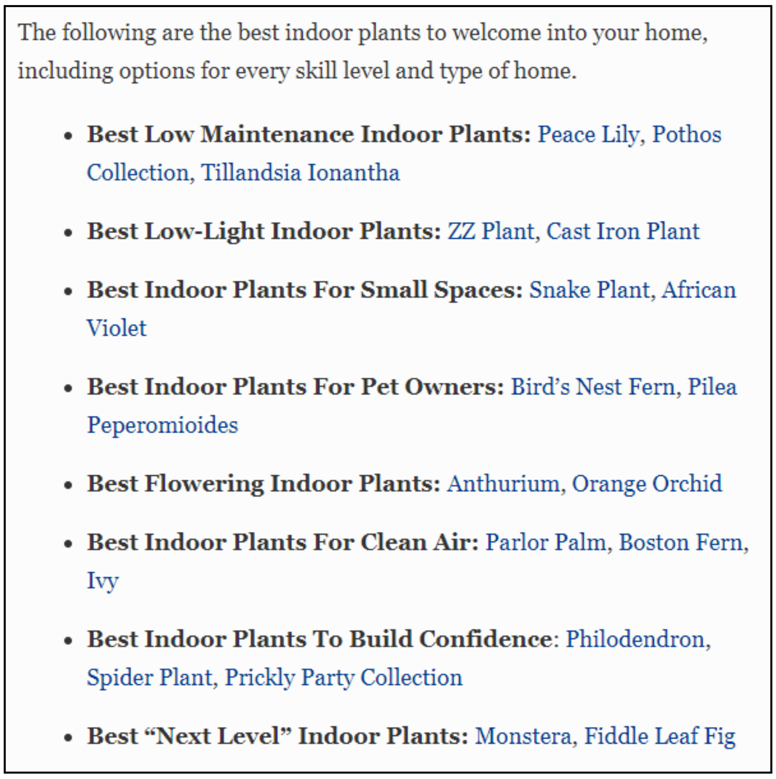
The paddle plant is a bold succulent that has big, round leaves with pink tips. These plants favor bright light so you can have it sit right on a sunny windowsill to soak in all the rays. These also like to be on the drier side, so don't worry if you forget to water from time to time — they can take it!

Figure 2

This recommendation site provides even less detailed information than the previous site. However, due to this simplification it makes it very easy to understand and keeps the site available for beginners or people who are unfamiliar with plant growth or any technical terminology. On the other hand, this lack of information would mean that any more skilled people looking for advice or additional information on a plant will not find any support from this site, there is not even a “more” section or “advanced” tab in order to open this up and be available to all skill levels.

This shows me that in my project, in order to make it accessible to all users, I will need to make sure that all data or information displayed to the user includes specific and advanced information, however also a simplified version of this information.

3. https://www.forbes.com/sites/forbes-personal-shopper/2022/02/27/best-indoor-plants/ & https://www.plants.com/p/peace-lily-plant-157654?afsrc=1&clickid=VLUUpqzijxyNT0H2N%3ASXpxqkUkDWOuURnWEy100&irgwc=1

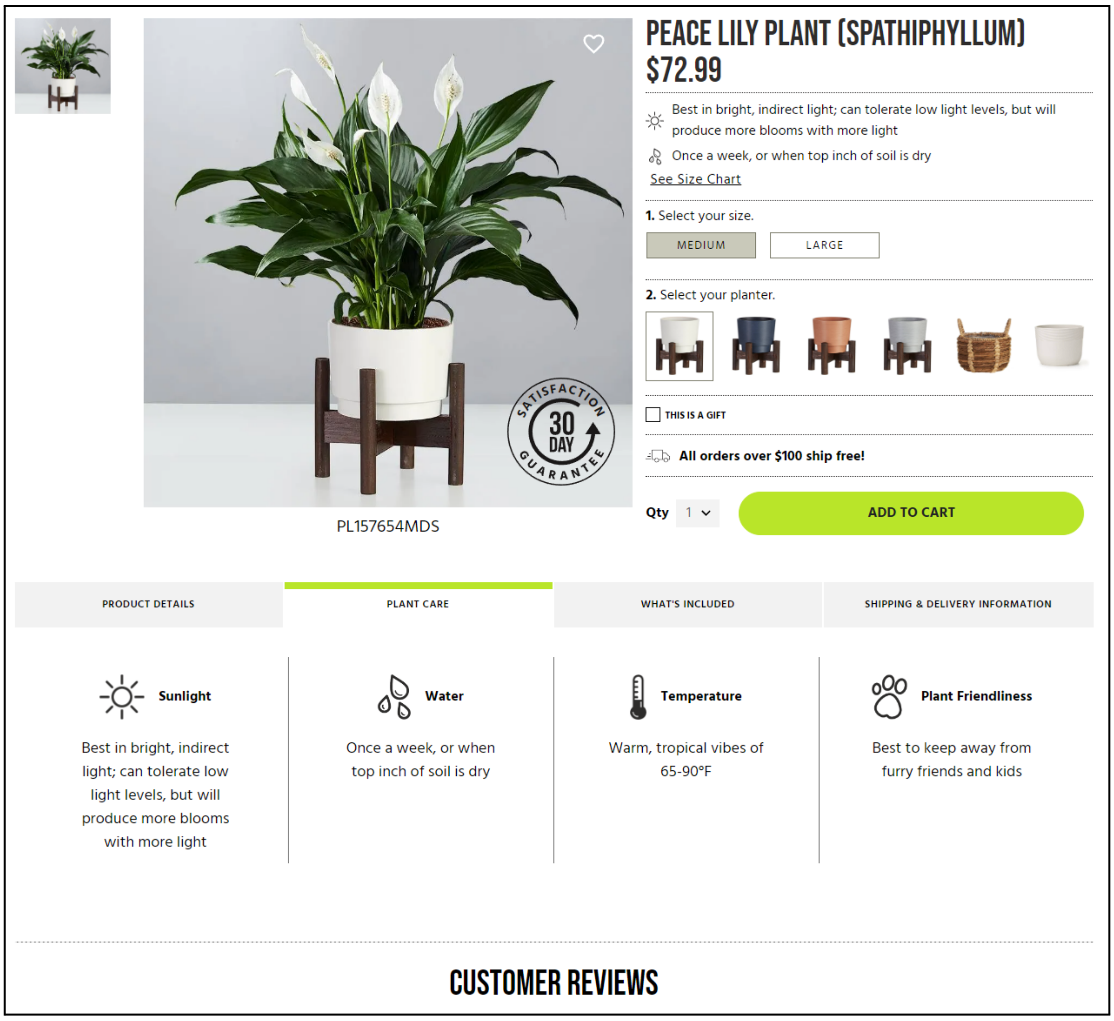

The following are the best indoor plants to welcome into your home, including options for every skill level and type of home.

- **Best Low Maintenance Indoor Plants:** Peace Lily, Pothos Collection, Tillandsia Ionantha
- **Best Low-Light Indoor Plants:** ZZ Plant, Cast Iron Plant
- **Best Indoor Plants For Small Spaces:** Snake Plant, African Violet
- **Best Indoor Plants For Pet Owners:** Bird’s Nest Fern, Pilea Peperomioides
- **Best Flowering Indoor Plants:** Anthurium, Orange Orchid
- **Best Indoor Plants For Clean Air:** Parlor Palm, Boston Fern, Ivy
- **Best Indoor Plants To Build Confidence**: Philodendron, Spider Plant, Prickly Party Collection
- **Best “Next Level” Indoor Plants:** Monstera, Fiddle Leaf Fig

Figure 3

This site seems to provide recommendations based on common categories, and then links to a different site, this site being “plants.com”. As a recommendation site, I believe this does a good job of giving a few options for the most popular categories, however once again there are no customization options and so no recommendations can be specific to the user, and are instead based on generalisations of the customers and that is how they provide up to date recommendations.

In terms of plant details, this is now down to the site they seem to be partnered with, this being “plants.com”. The figure below shows a page for one of their plants.



PEACE LILY PLANT (SPATHIPHYLLUM)
$72.99

Best in bright, indirect light; can tolerate low light levels, but will produce more blooms with more light

Once a week, or when top inch of soil is dry

See Size Chart

1. Select your size.

MEDIUM | LARGE

2. Select your planter.

THIS IS A GIFT

All orders over $100 ship free!

Qty 1 | ADD TO CART

PL157654MDS

PRODUCT DETAILS | PLANT CARE | WHAT'S INCLUDED | SHIPPING & DELIVERY INFORMATION

Sunlight	Water	Temperature	Plant Friendliness
Best in bright, indirect light; can tolerate low light levels, but will produce more blooms with more light	Once a week, or when top inch of soil is dry	Warm, tropical vibes of 65-90°F	Best to keep away from furry friends and kids

CUSTOMER REVIEWS

Figure 4

This website seems to provide quite adequate information, once again quite general however they have included some details as to what the different light levels may actually mean for the plant. Additionally they give a time period for how often to water it, or how to tell when to water it. This can be quite useful for users so they can base their watering habits off of this information and can keep an eye out for it.

However once again there are no units, no comparisons, no perspective as to how any of this information can actually be interpreted. There are units to the temperature, which is very important and useful. And they’ve included a range as clearly the plant can withstand a range of temperatures.

Candidate Number: **1266**
Centre Number: **58231**

Review
In conclusion, by comparing the similarities and differences of these, I have decided to add the following objective(s) to my project so that I am confident it will perform as a solution to the initial problem:

- Must include information in detail and in simplified forms
- Must provide usable information on the 3 main components of plant growth

1.4.2 Potential abstract data types / algorithms

All of the following vector formulae work with any vector of n dimensions, I only went up to 3 dimensions, however the project will be using far more, as each dimension will be a different property of a plant (watering, sunlight, humidity, etc.)

So in the context of my project, I will be utilising vectors as a data structure to store components and variables for the plants. They will be needed in order to keep the data easy to handle, and allow the angle between them to be calculated, as it can be imagined as a coordinate system in whichever dimension is a coordinate. This can consist of limitless dimensions, however we shouldn't need more than 3 dimensions in this instance.

Vectors - Vectors can be imagined as 1x ***n*** matrices. A matrix can be used to store a grid of values, and a matrix can have Arithmetic operations carried out on them, such as addition, subtraction, and multiplication. Matrices can be very easily noted if you consider them much like ***n*** dimensional arrays. Vectors are commonly annotated as below:

$$\vec{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Vectors (dot product):

The dot product of two vectors consists of systematic multiplication. This is a different order of operations to the more commonly found multiplication of matrices and vectors.

In normal matrix multiplication, the elements are multiplied row element by column element, going from the top row to the bottom row, and it requires a certain order. An example of this can be seen below:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} \{ae + bg\} & \{af + bh\} \\ \{ce + dg\} & \{cf + ch\} \end{bmatrix}$$

However, we are trying to find the dot product of two vectors. This requires a different method of operations, and the result will be a single value instead of a resultant vector. We would normally expect a resultant matrix if we are conducting matrix multiplication, so this differs from that.

Below, you can see how the dot product of two vectors, ***a*** and ***b*** can be calculated.

$$\vec{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\vec{a} \cdot \vec{b} = a1 * b1 + a2 * b2 + a3 * b3$$

As you can see, finding the dot product of two vectors is very straightforward, and is simply the sum of each element in one vector multiplied by the corresponding element in another vector. Due to this fact, the dot product calculation of two vectors can only be performed if both vectors are of the same number of dimensions.

Despite it being such a simple calculation, the dot product of two vectors is a significant and important part of calculating the angle between two vectors, and so it is vital that the final program can complete this calculation flawlessly and efficiently, as this calculation will be performed on thousands of entries.

Vectors (magnitude):

The magnitude of a vector is simply the direct displacement of a coordinate after a translation by the aforementioned vector. In order to calculate this, we can simply take the origin as the initial coordinate, and then the final coordinate can simply take the values of the vector.

In the dimensions we are going to be working in, the magnitude of each vector can simply be calculated by the use of pythagoras' theorem. In two dimensions this is as follows:
$a^2 + b^2 = c^2$
Where a and b are the two magnitudes of the dimensions of x and y, and c is the resultant hypotenuse of the right-angled triangle formed - or in other words, the **magnitude**.

This can easily be done in three dimensions, as the third dimension can be added with no other change to the equation or the calculation. So in three dimensions, the new equation can be considered as
$a^2 + b^2 + c^2 \ = d^2$
Where a, b, and c are the three dimensions of the vector, and thus d becomes the magnitude.

The magnitude of a vector is annotated as follows:
$$|\vec{a}|$$

So finally, the magnitude of a three dimensional vector (the equation that will be utilised in the project), will be as shown below.

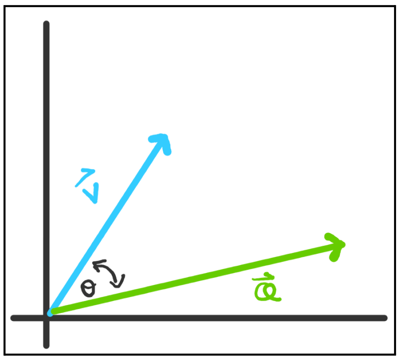
$$|\vec{a}| = \sqrt{(a_1)^2 + (a_2)^2 + (a_3)^2}$$

Angle between vectors (Dot product cos(θ) rule):

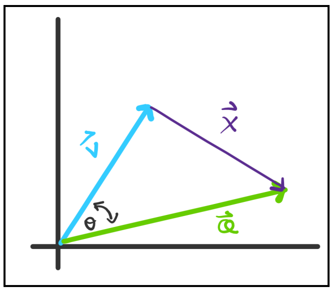
When working with vectors, the angle between two vectors is information that is highly needed, and is important for numerous applications of vectors and vector analysis. An equation has been found for θ in terms of vectors ***a*** and ***b*** and their magnitudes. This can be seen below

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}|\left|\vec{b}\right|}$$

Proof
This equation can be used no matter the number of dimensions of vectors ***a*** and ***b***, so to simplify this proof I will be working with vectors in two dimensions.

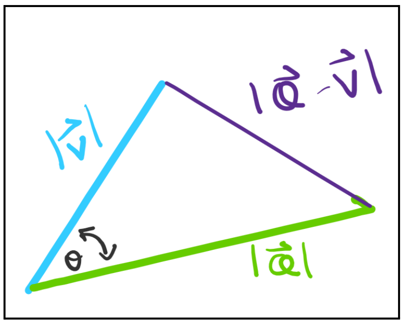


So we take our two vectors in two dimensional space, and we can plot them on a graph. Labelling θ as the angle between them.



Then we can plot a joining vector, ***X***, and determine that ***X*** has a value of ***Q-V***

$$\vec{V} + \vec{X} = \vec{Q}$$
$$\vec{X} = \vec{Q} - \vec{V}$$



Next we can plot the vectors as a triangle, using their magnitudes as the lengths of each corresponding side.

Then we use this triangle to find the size of theta, using the regular triangle cosine rule.

Cosine Rule

$$a^2 = b^2 + c^2 - 2ab \cdot \cos(A)$$

Strictly for this triangle

$$\left|\vec{Q} - \vec{V}\right|^2 = \left|\vec{V}\right|^2 + \left|\vec{Q}\right|^2 - 2\left|\vec{V}\right|\left|\vec{Q}\right| \cos(\theta)$$

Proof

$$\vec{V} \cdot \vec{V} = \begin{pmatrix} x \\ y \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = x^2 + y^2 = \left|\vec{V}\right|^2$$

So…

Candidate Number: **1266**
Centre Number: **58231**

$$\begin{aligned}\left|\vec{Q}-\vec{V}\right|^2 &= \left|\vec{Q}-\vec{V}\right|\cdot\left|\vec{Q}-\vec{V}\right| \\ &= \vec{V}\cdot\vec{V}+\vec{Q}\cdot\vec{Q}-2\left|\vec{V}\right|\left|\vec{Q}\right|\cos(\theta)\end{aligned}$$

$$\cancel{\vec{Q}\cdot\vec{Q}}-\vec{V}\cdot\vec{Q}-\vec{V}\cdot\vec{Q}+\cancel{\vec{V}\cdot\vec{V}} = \cancel{\vec{V}\cdot\vec{V}}+\cancel{\vec{Q}\cdot\vec{Q}}-2\left|\vec{V}\right|\left|\vec{Q}\right|\cos(\theta)$$

$$\cancel{-2}\left(\vec{V}\cdot\vec{Q}\right) = \cancel{-2}\left|\vec{V}\right|\left|\vec{Q}\right|\cos(\theta)$$

$$\vec{V}\cdot\vec{Q} = \left|\vec{V}\right|\left|\vec{Q}\right|\cos(\theta)$$

$$\cos(\theta) = \frac{\vec{V}\cdot\vec{Q}}{\left|\vec{V}\right|\left|\vec{Q}\right|}$$

1.4.3 First interview

This first interview is going to be useful in providing an insight into what the end user(s) are most interested in, and what obstacles they believe may limit their usage of the program. Their responses to the following questions (as shown below each question) will allow me to narrow down my objectives and focus on what is most significant and ensure my prototype iterations are in-line with the user's requirements.

Q: Why do you enjoy looking after plants?

"*There's a sense of achievement when you get a seed to germinate. And the nurturing element of getting a plant to grow. They help to improve the air quality in the house. I find something satisfying about being part of a system or a process that keeps a plant alive, there's a sense of responsibility - so it's quite upsetting if a plant doesn't survive due to environmental needs I was unaware of or was uncontrollable but it was too late if I had already purchased the plant.*"

Q: What are the main issues you run into when trying to look after a plant?

"*Maintaining the correct environmental conditions that are possibly not so simple to control. So sunlight, heat, and water. As they can be very temperamental. I believe that sunlight is incredibly important as it is something that is completely uncontrollable when I'm not directly with the plant. If I'm out and about I cannot move it away from a window or make sure it's in shade. Heat is also very similar to that, as it's so important for a plant but, unless you have a controlled enclosure for it, there is little to slim chance of being able to alter the temperature of your entire house or room just for one plant. It would take a lot of effort, energy and cost to keep the house at a steady temperature no matter the time of year. I think water is a lot easier to control, however even if it is easy to control it doesn't mean it* *doesn't become an issue, as sunlight and temperature then affects how much water it needs regularly, as a plant in direct sunlight will dry out significantly one that stays in the shade the entire time - however this is rarely reflected in recommendations.*"

Q: Expanding on the previous question, what information do you wish this program to provide to support you when deciding on a plant to purchase?

"*It would help if it could provide the lowest temperature it can survive at, a temperature range it enjoys or thrives in, details of the sunlight requirements, and whether it likes a wet or dry environment, maybe again a band or a kind of moisture scale so you know where you can aim to keep it, otherwise you'll be drip feeding it water when in fact you could've just watered once a week for example.*"

Q: What would you say would be the most important feature of a plant recommendation program?

"*I think temperature information is the key bit. You can feed it or shade it, but the temperature is very 'out of my hands', and its variance can really impact the plant's health so it's quite important in my opinion.*"

Q: How frequently do you purchase a plant or attempt to grow a plant and it dies due to improper care / an unsuitable environment?

"*Probably once a month on average, there's usually growing periods where I'd purchase more than one a week, but over the entire year it would be nearer once a month I have a new plant on the go.*"

Q: Is there anything you would like to add that you believe would be helpful for the design and development of the program?

"*It would be good if you had categories for indoor or outdoor plants, or at least a way of organising plants by what they would like in terms of their environment. Probably need to split that so it's easier to identify whether it's an indoor or outdoor plant and assign them to separate categories inside the program if that's possible.*"

Review

The first interview was very informative, it gave an overview of the requirements of the program, and possible features I could implement. I believe in the prototype I will make sure to find and use an API that could provide these details, and then make sure this information is easily accessible or viewable by the user.

1.4.4 Key components

After my initial research and following the interview, I have identified the following main components that are key to the functionality and success of the plant recommendation program:

ANGLE BETWEEN VECTORS (dot product cosine rule)
This will be a key component of the program, as this is how the recommendation system will be functioning and be implemented. The calculations made for this will need to be completed efficiently in order to complete the comparison and calculation on over 3000 entries every time a recommendation is calculated.

API USAGE
- Interactions through the API to provide plant information
- Search for plant by name

USER INTERFACE
- For user input and user experience
- Provides details required and states clearly the outputs

1.5 Further Research

1.5.1 Prototype

My prototype will test the API usage and the vector dot product calculations in order to determine the most suitable plant. This will give me insight whether the project can be completed and how complicated it would be to create.

1.5.1a Information

Below is a table of contents for the modelling section, that should give more detail for what the prototype covered and how this supports the completion of the final project.

Name	*Type*	*Overview*
GetAngle	function, returns double	Retrieves the value of θ (in rad) from the equation found in **Section 1.4.2**
GetToken	function, async Task<string>	Retrieves the token needed to access the api. A token is requested for every instance of the program as it needs a new token and the tokens expire after a set time.
GetSearchResult	function, returns PlantSearch	This function retrieves a list of Plants that fit the search query through the API, and this is returned as a PlantSearch object.
GetPlantDetailsByPID	function, returns PlantDetails	This function retrieves all the information available for a PlantDetails object through the API.
PlantDetails	class	Used as a custom data structure. Contains all plant details for a single plant after the details that are provided by the plantAPI when a request is sent has been processed. This information is only available when the plant details are retrieved by PID
PlantSearchResult	class	Used as a custom data structure. Contains the details that are provided after a general plantsearch. These details are only plant_pid, display_pid, and the plant’s alias.
PlantSearch	class	Used as a custom data structure. Contains every result from searching for a plant. This means it contains a list of PlantSearchResult and the count of the results. In addition to this it temporarily contains the next url and the previous url that is/was used to make a request to, as the lists can be thousands long and only 100 are available in each request so it is split over several requests.

1.5.1b Modelling

GetAngle

This function will practically determine the majority of the project. This function will be run thousands of times and is the key component for whether the program can provide accurate recommendations. Errors in this function will render the program useless, so this was a very important function to prototype.

This function should return the value of θ from the equation from **Section 1.4.2**
(as shown below)

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}|\left|\vec{b}\right|}$$

Code

```
private double GetAngle(Vector v1, Vector v2)
        {

            double angle;
            double dotproduct = v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
            double magv1 = Math.Sqrt(Math.Pow(v1.x, 2) + Math.Pow(v1.y, 2) + Math.Pow(v1.z, 2));
            double magv2 = Math.Sqrt(Math.Pow(v2.x, 2) + Math.Pow(v2.y, 2) + Math.Pow(v2.z, 2));
            angle = Math.Acos(dotproduct / (magv1 * magv2)); // in rad
            return angle;
        }
```

Breakdown
As it can be seen in the code, each line of this function can be seen in the equation and is named appropriately to show this clear correlation.

Dot product

```
double dotproduct = v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
```

$\vec{a}\cdot\vec{b}$ from the equation in this case is represented by $\vec{v_1}\cdot\vec{v_2}$

$$\vec{v_1} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \vec{v_2} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Where the data for these dimensions are such that

$\vec{v1}_x = v1.x$

So in conclusion, this line of code is equivalent to the following maths calculation

$$\vec{v_1} \cdot \vec{v_2} = \begin{bmatrix} v1_x \\ v1_y \\ v1_z \end{bmatrix} \cdot \begin{bmatrix} v2_x \\ v2_y \\ v2_z \end{bmatrix} = v1_x \cdot v2_x + v1_y \cdot v2_y + v1_z \cdot v2_z$$

Magnitude of Vectors

```
double magv1 = Math.Sqrt(Math.Pow(v1.x, 2) + Math.Pow(v1.y, 2) + Math.Pow(v1.z, 2));
```

These two lines are a substitute for the following equation

$|\vec{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2}$

where vec(a) is equal to v1 and v2

As `Math.Pow(v1.x, 2)` is the equivalent of

$(v1.x)^2$

Final angle

```
double angle = Math.Acos(dotproduct / (magv1 * magv2)); // in rad
```

The line above shows the final working to get the angle between two inputted vectors. This line is a combination of all of the previous workings, and is directly equal to the original equation we were looking for:

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}|\left|\vec{b}\right|}$$

GetToken

GetToken is an asynchronous function which means it will run synchronously until the program reaches its await expression, where it will then pause the program until the synchronous task has been completed, at which point it will continue. GetToken is used to retrieve the temporary access token for the plantbook API.

This was the first function I wrote that would interact with the API, so it was pivotal that it functioned as intended before I could start work on the final solution. The code below shows the final method I ended up using for the prototype.

Code

```
public static async Task<string> GetToken() // async Task<string>
    {
        Console.WriteLine("Generating token...");
        HttpResponseMessage response; // string result;
        using (HttpClient httpClient = new HttpClient())
        {
            using (HttpRequestMessage request = new HttpRequestMessage(new
HttpMethod("POST"), "https://open.plantbook.io/api/v1/token/"))
            {
                MultipartFormDataContent multipartContent = new MultipartFormDataContent();
                multipartContent.Add(new StringContent("client_credentials"), "grant_type");
                multipartContent.Add(new
StringContent("DNlEeLNQOpfYW3bTeMfY8v9b1mrMKGYfi6VGTfZC"), "client_id");
                multipartContent.Add(new
StringContent("KylrYac2J7OwCLfDChw8twiNoscP9wBtPzHeUKyYeIRnafrYfY2lPZ5yrubgxsYw
asv7yno1B9ZkbAXvdIL534lhvq6TMNf0bEh8EHerbM3iFj3OD31ZoHpR8JDLtfqv"),
"client_secret");

                request.Content = multipartContent;

                response = await httpClient.SendAsync(request);
            }
        }
        return response.Content.ReadAsStringAsync().Result.Substring(18, 30);
    }
```

Breakdown

This function uses the library System.Net.Http to interact with the API I am using, as http requests need to be made.
The code below is the blank request being instantiated

```
HttpClient httpClient = new HttpClient()
HttpRequestMessage request = new HttpRequestMessage(new HttpMethod("POST"),
"https://open.plantbook.io/api/v1/token/")
```

This request message is currently blank, it is simply allocated the method and the intended address.

In order to interact with the API successfully, this Http request requires certain contents for it to be processed correctly server-side. For the plantbook API, it requires an account so that it can generate a token, including the client's ID and client's secret. In Layman's terms, it is the username and password for the client.

In order to add this information to the message, multipart content is created, with a header and details for each part. I used MultipartFormDataContent and instantiated this object, then added the required details individually. Then, I set the original request content as the new multipart data content I just created, as seen below.

```
MultipartFormDataContent multipartContent = new MultipartFormDataContent();
multipartContent.Add(new StringContent("client_credentials"), "grant_type");
multipartContent.Add(new StringContent(id), "client_id");
multipartContent.Add(new StringContent(secret), "client_secret");

request.Content = multipartContent;
```

Now the final step for this message was to simply send it. The code I used to send and then process the received message can be seen below.

```
response = await httpClient.SendAsync(request);
return response.Content.ReadAsStringAsync().Result.Substring(18, 30);
```

The response that was received from this request was not just the client's temporary token, there were extra parts. However, I am not interested in these extra parts, so the result was trimmed down so I just received the 30 digit token.

```
.Substring(18, 30)
```

As seen by the substring function, it trimmed the received string starting at the 18th character, and then returned the next 30 characters as the result.

GetSearchResult

The code below will show the function I used in the prototype that allowed a string to be entered and this would be searched across the API. It would return a list of possible plants that match the search query. As a result, this function returned the object **SearchResult** (see **1.5.1a Information** for an overview). This was a complex data structure created specifically for this function to allow the result to be processed far more easily.

This interaction with the API is very similar to the **GetToken** function, however the **GetSearchResult** function uses a json deserializer to process the data returned by the API.

This means that the request being sent is very similar, however the way the result is processed is very different.

In addition to this, due to limitations on the API side, search results are returned in lists of maximum 200. This means that for certain species of plant, this limit is breached and so several requests will be sent in order to retrieve all the lists from different pages.

Code
Due to the size of this method, the code will be analysed in the **breakdown** (below). However, the code in its entirety will be available as seen in section **5.1.1 Program.cs**

Breakdown
GetSearchResult is very similar to GetToken in the way that it interacts with the API. However due to the way that it interacts with the API, it is required to receive and deserialize the object received.

So, as we know the format that the data is sent back in, we can directly input this into a usable data structure. In this case we use **PlantSearchResult** and **PlantSearch**.

As an example, let's say we wanted to search for "acer", we would need to use the "GET" method, and send it to the open.plantbook.io following address

```
GET /api/v1/plant/search?alias=acer&limit=10&offset=20
```

So for this search, we receive the following from the API…

```
{
  "count": 38,
  "next": "https://open.plantbook.io:443/api/v1/plant/search?alias=acer&limit=10&offset=30",
  "previous": "https://open.plantbook.io:443/api/v1/plant/search?alias=acer&limit=10&offset=10",
  "results": [
    {
      "pid": "acer macrophyllum",
      "display_pid": "Acer macrophyllum",
      "alias": "acer macrophyllum"
    },
    {
      "pid": "acer maximowiczianum",
      "display_pid": "Acer maximowiczianum",
      "alias": "acer maximowiczianum"
    },
    {
      "pid": "acer miyabei",
      "display_pid": "Acer miyabei",
      "alias": "acer miyabei"
    },
    {
            "pid": "acer mono",
            "display_pid": "Acer mono",
            "alias": "acer mono"
        },
        {
            "pid": "acer monspessulanum",
            "display_pid": "Acer monspessulanum",
            "alias": "acer monspessulanum"
        },
        {
            "pid": "acer negundo",
            "display_pid": "Acer negundo",
            "alias": "acer negundo"
        },
        {
            "pid": "acer palmatum",
            "display_pid": "Acer palmatum",
            "alias": "acer palmatum"
        },
        {
            "pid": "acer pensylvanicum",
            "display_pid": "Acer pensylvanicum",
            "alias": "acer pensylvanicum"
        },
        {
            "pid": "acer platanoides",
            "display_pid": "Acer platanoides",
            "alias": "acer platanoides"
        },
        {
            "pid": "acer pseudoplatanus",
            "display_pid": "Acer pseudoplatanus",
            "alias": "acer pseudoplatanus"
        }
    ]
}
```

[from https://open.plantbook.io/api/v1/plant/search?alias=acer&limit=10&offset=20]

This data can be processed rather simply using a json deserializer and the {get; set;} methods in the object setup. A json deserializer takes in all of these individual values and then immediately allocates them to the variable that is needed.

So if we look at the response from the API, we can see that we are given an int variable called "count", followed by two strings, and then an array called "results" containing an unknown number of elements (in this case 20).

So to process this, in the code, we include the following

```
string result = streamReader.ReadToEnd();
json = result;
```

Candidate Number: **1266**
Centre Number: **58231**

So the result of the response is now put into a string called “result”, and then the json string is immediately assigned the value of result.

```
JsonConvert.DeserializeObject<PlantSearch>(json);
```

This line of code is all that is needed to convert this initial response into the **PlantSearch** data format. And this is all that is needed to convert this complicated response from the API into a usable data structure.

GetPlantDetailsByPID

This function is what allows specific advanced details of a plant to be accessed individually. This is fetched from the API once again using a very similar request system, however the response is stored in a different data structure this time as the results sent back are once again different. We store this data in the **PlantDetails** data structure.

The result from the API for a request such as this, can be seen below

```
{
   "pid": "acer pseudoplatanus",
   "display_pid": "Acer pseudoplatanus",
   "alias": "acer pseudoplatanus",
   "max_light_mmol": 7200,
   "min_light_mmol": 3000,
   "max_light_lux": 75000,
   "min_light_lux": 2800,
   "max_temp": 35,
   "min_temp": 5,
   "max_env_humid": 80,
   "min_env_humid": 30,
   "max_soil_moist": 60,
   "min_soil_moist": 15,
   "max_soil_ec": 2000,
   "min_soil_ec": 350,
   "image_url": "https://opb-img.plantbook.io/acer%20pseudoplatanus.jpg"
}
```

Code

```
public static PlantDetails GetPlantDetailsByPID(string token, string instr)
        {
            string urlStr = "";
            for (int i = 0; i < instr.Length; i++)
            {
                if (instr[i] == ' ')
                {
                    urlStr += "%20";
                }
                else
                {
                    urlStr += instr[i];
                }
            }

            string url = $"https://open.plantbook.io/api/v1/plant/detail/{urlStr}/";
            HttpWebRequest httpRequest = (HttpWebRequest)WebRequest.Create(url);

            httpRequest.Headers["Authorization"] = "Bearer " + token;
            string json;
            HttpWebResponse httpResponse = (HttpWebResponse)httpRequest.GetResponse();
            using (StreamReader streamReader = new
StreamReader(httpResponse.GetResponseStream()))
            {
                string result = streamReader.ReadToEnd();
                //Console.WriteLine(result);
                // debugging
                json = result;
            }
            PlantDetails newplant = JsonConvert.DeserializeObject<PlantDetails>(json);
            newplant.opt_humid = (newplant.max_env_humid + newplant.min_env_humid) / 2;
            newplant.opt_light_lux = (newplant.max_light_lux + newplant.min_light_lux) / 2;
            newplant.opt_temp = (newplant.max_temp + newplant.min_temp) / 2;
            return newplant;
        }
```

Breakdown
This code should seem very familiar, as it is practically identical to the **GetToken** and **GetSearchResult** functions. This is because they are all http requests so are all in a very similar format.

The only significantly different section of this function is the 4 line calculation that can be seen towards the end.

```
    PlantDetails newplant = JsonConvert.DeserializeObject<PlantDetails>(json);
    newplant.opt_humid = (newplant.max_env_humid + newplant.min_env_humid) / 2;
    newplant.opt_light_lux = (newplant.max_light_lux + newplant.min_light_lux) / 2;
    newplant.opt_temp = (newplant.max_temp + newplant.min_temp) / 2;
```

As can be seen, the first line of this is just the json deserializer being used again to convert the response into the **PlantDetails** format. However, the 3 lines below this show the optimum humidity, lux, and temperature of the plant.

This is very useful, as it is difficult to calculate an accurate recommendation when you are given a range, especially if the range is rather large.

So to counter this, the optimum values are used, which is simply the mean value of the maximum and minimum values provided by the API. This then means that the recommendation is based off of what the plants most want, and then the ranges should be checked at the end in the final project, just to make sure that a recommendation isn't being based off of the optimum if it means actually being outside the functioning range.

Custom data structures

PlantDetails
Contains all the details of a single, specific plant. Also contains Display() subroutine.

```
public string pid { get; set; }
public string display_pid { get; set; }
public string alias { get; set; }
public int max_light_mmol { get; set; }
public int min_light_mmol { get; set; }
public int max_light_lux { get; set; }
public int min_light_lux { get; set; }
public int opt_light_lux;
public int max_temp { get; set; }
public int min_temp { get; set; }
public int opt_temp;
public int max_env_humid { get; set; }
public int min_env_humid { get; set; }
public int opt_humid;
public int max_soil_moist { get; set; }
public int min_soil_moist { get; set; }
public int max_soil_ec { get; set; }
public int min_soil_ec { get; set; }
public string image_url { get; set; }
public void Display()  { }
```

PlantSearchResult
Contains the details provided for each plant in the list that is returned by the *GetSearchResult* function. Also contains Display() subroutine.

```
public string pid { get; set; }
public string display_pid { get; set; }
public string alias { get; set; }
public void Display() { }
```

PlantSearch
Contains the results of a search using the API. The most significant being the list of *PlantSearchResult*. This does not contain a Display() subroutine.

```
public int count { get; set; }
public string next { get; set; }
public string previous { get; set; }
public List<PlantSearchResult> results { get; set; }
```

1.5.2 Second interview

Now that I have started work on the design and prototype, I believe that reconsulting with my end user will give me a further advantage before I start my solution. They have already been shown the first few iterations of my prototype, and so they should now be able to gauge and imagine what I am aiming for with the final solution. This second interview should help me with finite changes and alterations or implementation of new features if they believe it would be beneficial to the user's experience.

Q: Do you believe a plant saving feature or a 'garden' feature would be more enjoyable for the user?
"I like that! I think that if you could save plants you are searching for or even save the environment conditions where you want to keep the plant you're looking for?"

Q: Would you like to see a preset feature then? Something that would allow you to save the details of an 'optimum' plant as it were?

"I think that would work. And maybe include a way to save any plants that you recommend from these? I think the 'garden' feature mentioned earlier would be a great way to save it. They could also see the date and time of creation so that it's easier to figure out when you were creating the environment?"

Q: Would a user system be useful for this?
"I think that a user system would be useful for not only that but also if you save a plant it would be cool to see what other people are saving and looking at"

Q:

1.6 Final Objectives

1 User Interface

1. Menu / GUI / Choices
    - ➢ Menu options
      This is a list of options that should be available on the main menu
        - ○ Search for plant
        - ○ Enter plant details
        - ○ [EXT] Options
        - ○ [EXT] Information (sources, api's, extra details)
        - ○ Exit option (when appropriate)
    - ➢ Navigation using arrow keys
        - ○ Indicator to show currently 'selected' option
        - ○ Left/Right/Up/Down for appropriate navigation
    - ➢ [EXT] Selection using number keys (or enter)
        - ○ Ignores current selection option
        - ○ Press a key to immediately choose an option (each option next to an allocated number)
    - ➢ Exit option available to exit { mode / section currently in }
        - ○ **[A] force returns from subroutine**
          This will require good stack moderation and ensuring subroutines etc. are called in the correct order.
        - ○ ~~[B] force closes immediately~~
          (e.g. Environment.Exit(0);)
2. Display results
    - ➢ Display advanced and simplified details of plants
    - ➢ Allow plants to be stored ("favourited") for later visits
        - ○ "TheGarden" implementation (user gardens)
3. User Gardens
    - ➢ Users must be able to login to their accounts
    - ➢ Suggested plants capable of being saved to a garden
    - ➢ Each garden must be owned by a single user
    - ➢ Each user can own several gardens
    - ➢ Every garden can contain numerous plants
    - ➢ A single plant entry in "TheGarden" can only be associated with one garden
    - ➢ Every garden can be deleted, however only by their owner
    - ➢ Plants from TheGarden can only be deleted by their owner
    - ➢ Guests should not be able to save plants to TheGarden
    - ➢ TheGarden should be capable of being filtered by
        - ○ Plant Name
        - ○ Owner of the plants
        - ○ Garden they are in

4. Take in user input
    - ➢ Integers when requested
    - ➢ Strings that are to be sanitised if necessary
    - ➢ Multiple choice input [refer to **User input** >> 1]
5. Sanitisation
    - ➢ Check the data entered is of the correct format
    - ➢ Do not continue unless data is correct
    - ➢ SQL data needs to be sanitised before queries are executed otherwise it will fail to execute properly

2 Data handling

1. The API must be interacted with correctly
    - ➢ Send valid requests
        - ○ Sanitise / insure valid URL
          Where to send the request
        - ○ Add appropriate headers
          Data to add to the request depending on request
        - ○ Gather and then use valid token for each instance
          Token is needed for authorisation (token expires after a set time, so request a new token every launch of the program)
    - ➢ Handle Responses
        - ○ Json deserializing
          JSON is a format that encodes other variables into a string.
          Serialization - object** to **string
          Deserialization - string** to **object
            - ● Create custom objects and assign data from the response to them (for plant data) [refer to **Data handling** >> 2 >> Custom objects]
2. Must be capable of handling data in different structures
    - ➢ Custom data structures
      Should include (but not limited to):
        - ○ **PlantDetails**
          The details of a single plant
          To include subroutine: Display()
          To include variables: pid, alias, maxlight, minlight, opt_light, maxtemp, mintemp, opt_temp, maxhumidity, minhumidity
          where "opt-<var>" is determined by the mean of maximum and minimum values provided for <var>.
        - ○ **PlantSearch**
          The object that is returned as the result of a search
          To include variables:
    - ➢ Separate class for user recommendations
3. Must be capable of scraping / caching
   Scraping is required to reduce huge delays while the code is running. As fetching data on the go is very time-consuming as the response result is required for calculations.

- ➢ Requires creation and writing of sql database for later use
- ➢ Initial scraping
    - ○ Scrapes entire database if not already completed
    - ○ If scrape has been completed, provide option to confirm every entry is valid before referring to it
- ➢ Caching may be required

  e.g. If there are other API interactions that are taking far too long to be practical, and are causing significant delays to the user's experience

  Caching search results and other time-consuming results would be logical in these situations

3 Providing Recommendations

1. Calculating minimum angle between vectors
    - ➢ Cosine dot product rule

    $$\cos\theta = \frac{\vec{a} \bullet \vec{b}}{|\vec{a}||\vec{b}|}$$

    Set ***a*** as optimum plant for user environment vector

    Set ***b*** as environment vector of current plant (iterate through every plant)

    When θ is minimum, that's the plant closest to the optimum
2. Provide sorted list of plants with best recommendation
    - ➢ Ordered (ascending order)
3. Search function available for the user

2 Technical Design

2.1 Programming Languages and Libraries

The primary language of the program will be C#; the reason for this being that it is the language I am most comfortable with, as I have spent time learning and familiarising myself with it during lessons at college in addition to using it for projects at home. I decided it would be best to go with the obvious choice as other languages would have required far more time spent learning features instead of working on the solution.

The other language I will use is SQL. It is required for my database interactions and it will be helpful when interfaced with my program. As I know basic SQL commands, the user's logins, their saved plants and their gardens could all be saved to tables in the database I am intending to use.

Find below the API and all of the libraries I used:

2.1.1 Plantbook API

The open source API I am using is named "plantbook" (https://open.plantbook.io/) and states that its purpose is:

"... to help the community with collecting and storing STRUCTURED information about plants care and potentially plants in general."

The API, when interfaced and interacted with properly, can be used to access data stored on thousands of plants, and also features a **search option (3.3)**. In the solution, the API is called upon for the search option, in addition to requesting more information on a given plant from a search query. Not only does my program do these requests during run-time, however to populate the plant database that is used to calculate and provide recommendations I scraped all information on any plant in the plantbook database.

2.1.2 List of implemented libraries

System.Net.Http;
System.Threading;
System.Threading.Tasks;
Newtonsoft.Json;

2.1.2.1 Http

In my solution, due to the requests I make to the API throughout run-time, the System.Net and System.Net.Http libraries are required in order to allow me to make such requests.

2.1.2.2 Tasks

System.Threading and System.Threading.Tasks are required in my program for making asynchronous requests to the API, and checking for timeout on the API if the user is not connected to the internet.

2.1.2.3 Newtonsoft Json

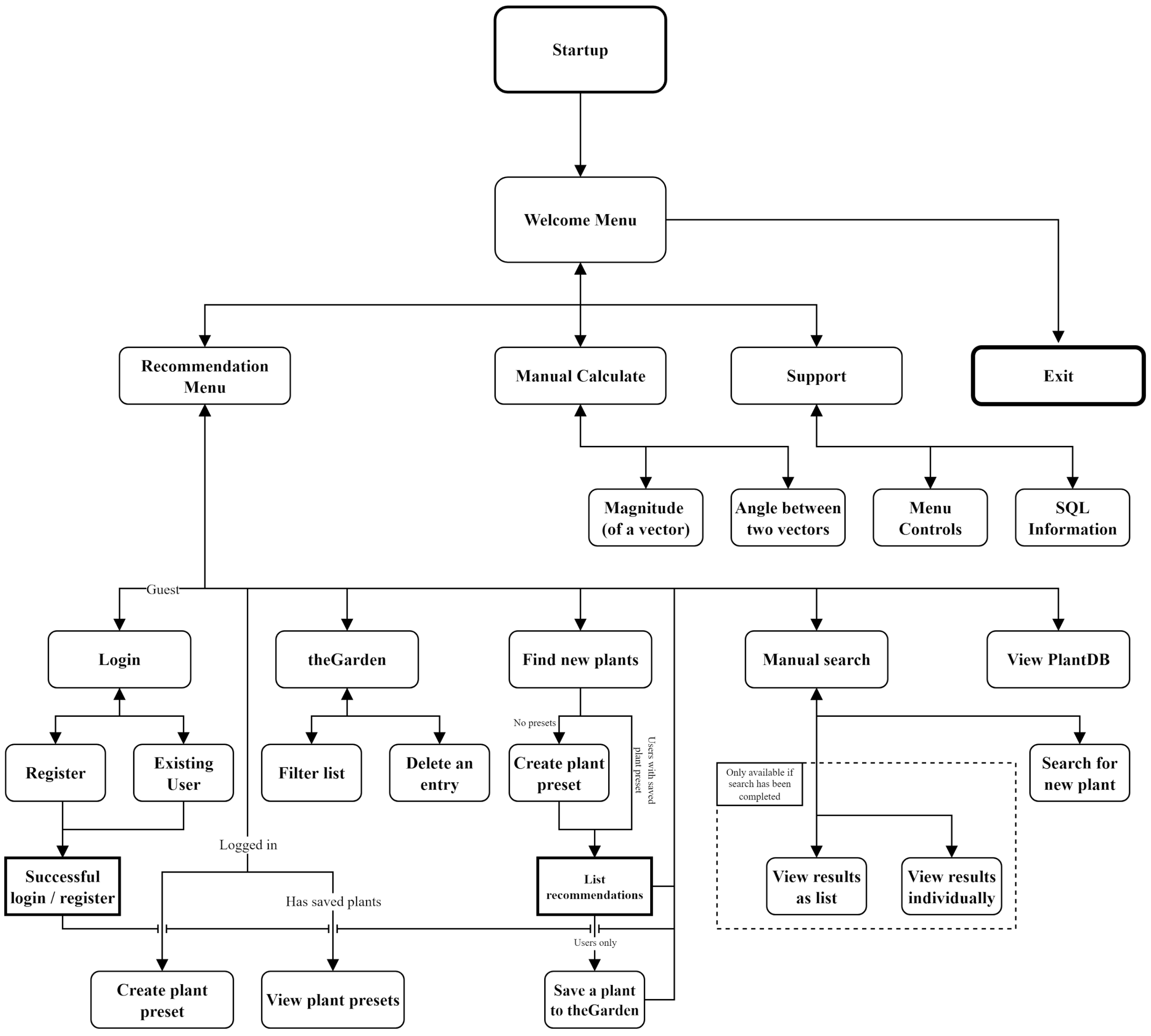
For interactions with the API, I used a Json deserialiser in order to process the response and convert it into usable custom data structures I designed. This library includes the deserialiser I used.

2.2 High-level Overview

The purpose of this project is to allow a user to gather information on their own plants, or search for new plants using a custom recommendation algorithm where they may then save these plants for later reference, including a named room or “garden” they wish to save to.

On the next page is the interface interaction hierarchy of the program, in which you can see what the user would be faced with and how they should be able to interact with the program. The round boxes are menus / options, and the hard square boxes are operations, actions, or events that occur.

The user can traverse the arrows as shown. They are directed, however almost all arrows are double ended, which will allow the user to traverse through menus and operations with ease, without having to close the instance of the program and reopen it just to reach an option on the menu before, for example.



The menu/interaction hierarchy of the program

2.3 Design Techniques

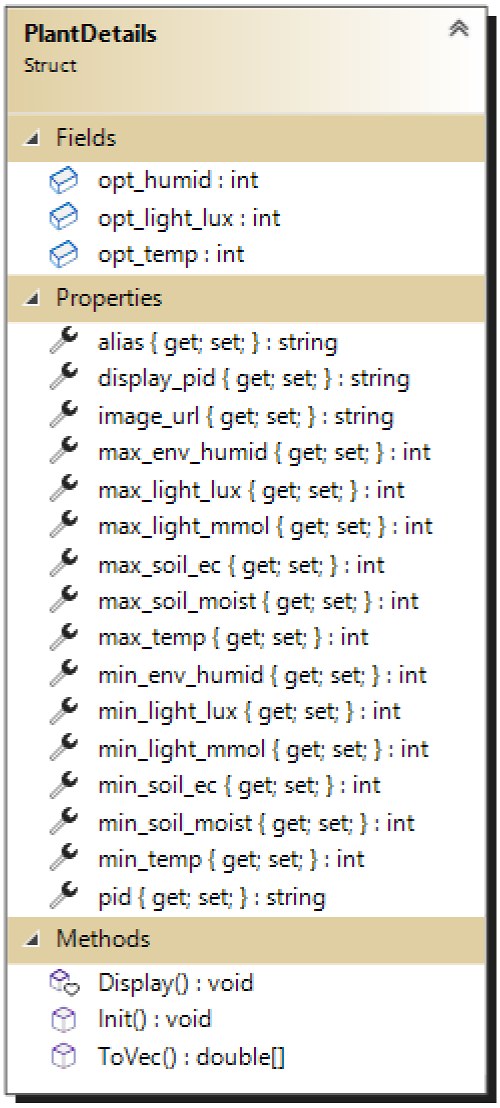
2.3.1 Custom Data Structures

Listed in this section will be every data structure I had to create for my solution to function as intended. You may note that several of these structures are all based around the API interactions and may be curious as to whether that was how the API was intended to be implemented. Unfortunately, the API in use did not come with any documentation.

Given the lack of documentation, I had to experiment with the interactions and look for features and learn how to use the features in question. Primarily I had to work out how the responses were configured in order to ensure I could request and receive the data as required.

2.3.1.1 PlantDetails

When testing with the API, I needed to create a custom data structure to store the information the API returns, in addition to anything I may need during the recommendation process. All of the variables defined as "properties" in the image below are all of the values directly returned by the API. the min/max of all the parameters (light, moisture, humidity, temp etc.). A json deserialiser is used to directly convert the API's responses into this custom structure (**2.1.2.3 Newtonsoft Json**).


PlantDetails
Struct

Fields
- opt_humid : int
- opt_light_lux : int
- opt_temp : int

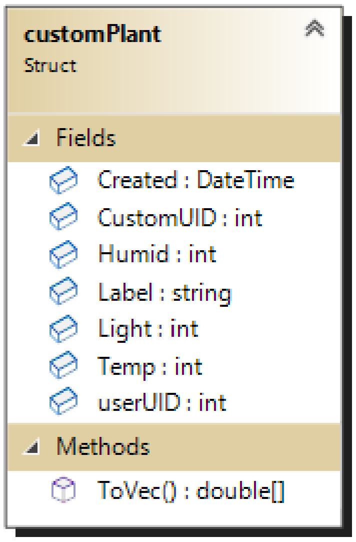
Properties
- alias { get; set; } : string
- display_pid { get; set; } : string
- image_url { get; set; } : string
- max_env_humid { get; set; } : int
- max_light_lux { get; set; } : int
- max_light_mmol { get; set; } : int
- max_soil_ec { get; set; } : int
- max_soil_moist { get; set; } : int
- max_temp { get; set; } : int
- min_env_humid { get; set; } : int
- min_light_lux { get; set; } : int
- min_light_mmol { get; set; } : int
- min_soil_ec { get; set; } : int
- min_soil_moist { get; set; } : int
- min_temp { get; set; } : int
- pid { get; set; } : string

Methods
- Display() : void
- Init() : void
- ToVec() : double[]

2.3.1.2 customPlant

In order to provide a recommendation to the user, a plant with ideal environment requirements had to be artificially created and stored in such a way that it could be compared to other plants. For my interactions with the API during prototype testing I realised that I would need to store the plant data from the API in a custom data structure ((**2.3.1.1 PlantDetails**) as well, so I ensured the parameters of the artificial *customPlant* lined up with them.

Additionally, during my second interview, the end user recommended the use of datetime stamps in order to identify creation dates, and also include names / labels so that they could be distinguished from one another without having to painstakingly browse through the parameter values.

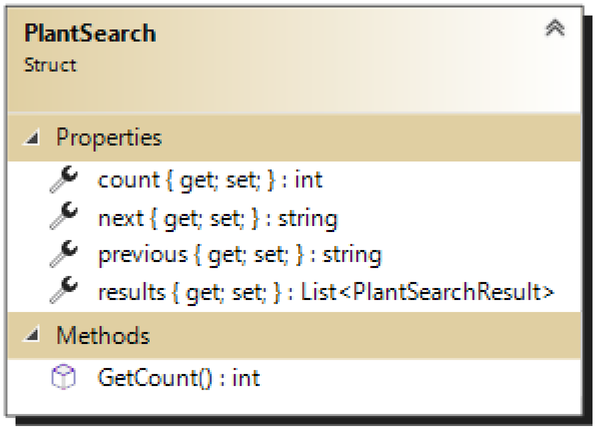


2.3.1.3 PlantSearch

This is another custom data structure required due to the API interactions that are happening. This variable is the response of a manual plant search through the API (as the name suggests).

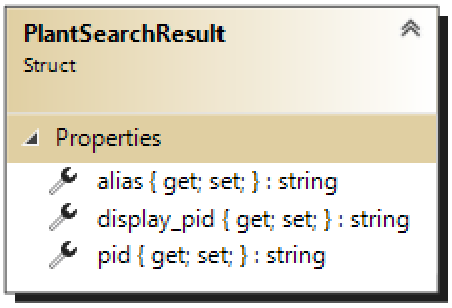
While designing this structure, it appeared that the API responded with: a total count of all search results for the query at hand (*count : int*), the url of the next page of the search (*next*), the url of the previous page of the search (*previous*), in addition to the results themselves (*results*) - which I parsed as a list.

It appeared that the number of elements per page of search results could be configured by alterations in the url the request was sent to. I experimented with this to try and find a maximum so that each request was as efficient as possible for each request. The maximum appeared to be 200 so this was the value I used for every search request.


PlantSearch
Struct
Properties
count { get; set; } : int
next { get; set; } : string
previous { get; set; } : string
results { get; set; } : List<PlantSearchResult>
Methods
GetCount() : int

2.3.1.4 PlantSearchResult

As mentioned above in the plantsearch (**2.3.1.3 PlantSearch**) breakdown, a list is required for each result, and the structure below shows what was included in terms of details for each entry.

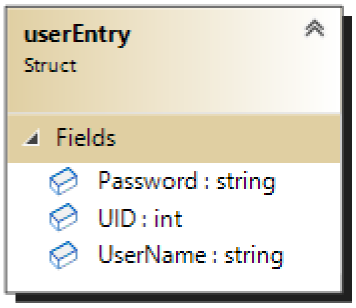


PlantSearchResult
Struct

Properties
- alias { get; set; } : string
- display_pid { get; set; } : string
- pid { get; set; } : string

2.3.1.5 userEntry

This structure is primarily for data stored in the table of users in the database, and is for identifying the current user, and allows actions to be made by the user and minimises SQL interaction during runtime. For example, it makes it very easy for their UID to be uploaded with a new custom plant as they can call upon [userEntry].UID at any point.

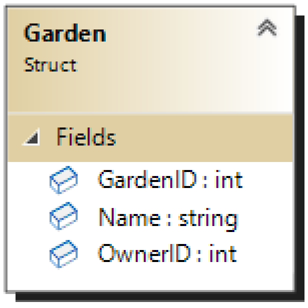


userEntry
Struct

Fields
- Password : string
- UID : int
- UserName : string

2.3.1.6 Garden

This structure is simply what I used to store the data from the solution's SQL database, as it stores what would be equivalent to an entry of a garden in the relevant table.

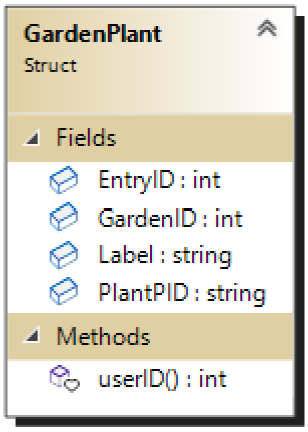


Garden
Struct

Fields
- GardenID : int
- Name : string
- OwnerID : int

2.3.1.7 GardenPlant

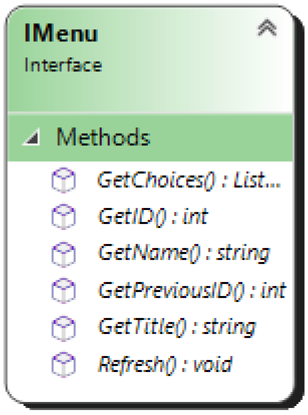
This structure (similarly to previous structures) is just a way of handling the data that is equivalent to a plant stored in theGarden table in the database.



2.3.2 Custom Interfaces

2.3.2.1 IMenu

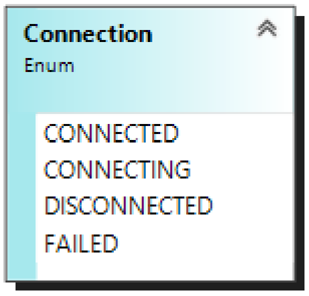
IMenu is an interface I made in order to allow for my menu system to function. The menu system was designed in order to allow backtracking and ensure a fluid user experience with an intuitive and pleasing interface.



*see more (**2.3.4 Menus**)*

2.3.3 CONNECTION Enum

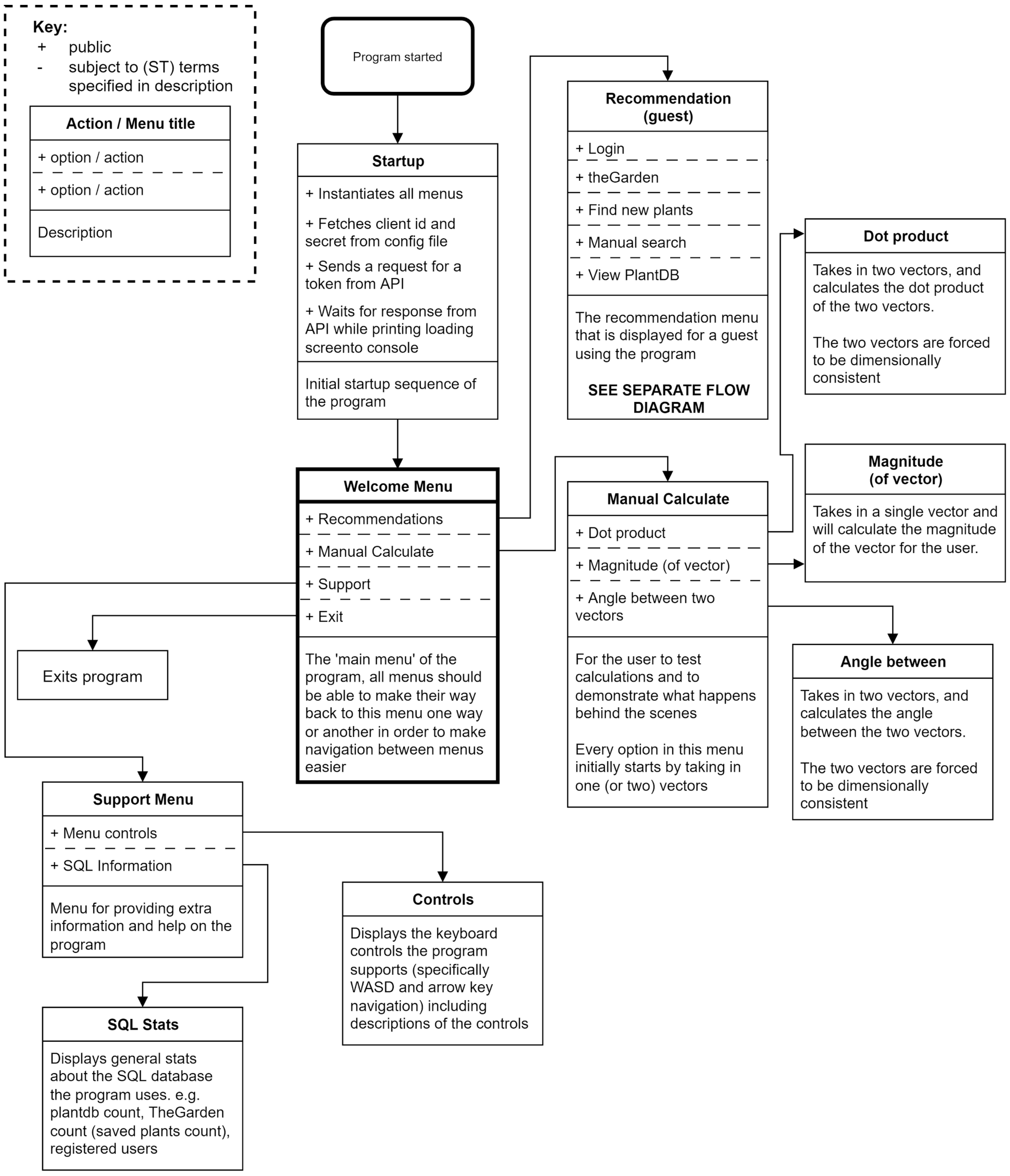
The connection enum is used to identify the current status of the API's connection. A custom enum allowed more easy understanding of the situation and gives more than the binary true or false of a bool connection indicator. With the enum I am able to identify what the status is, and then act accordingly. So if the API's connection was CONNECTING, I would wait until it had switched to CONNECTED or FAILED before attempting to resume any interactions. If it is FAILED I may try again, however not repeatedly attempt to connect as it is a FAILED attempt and not just DISCONNECTED - which may suggest a network connection issue or a failure server-side instead of simply a lack of connection.



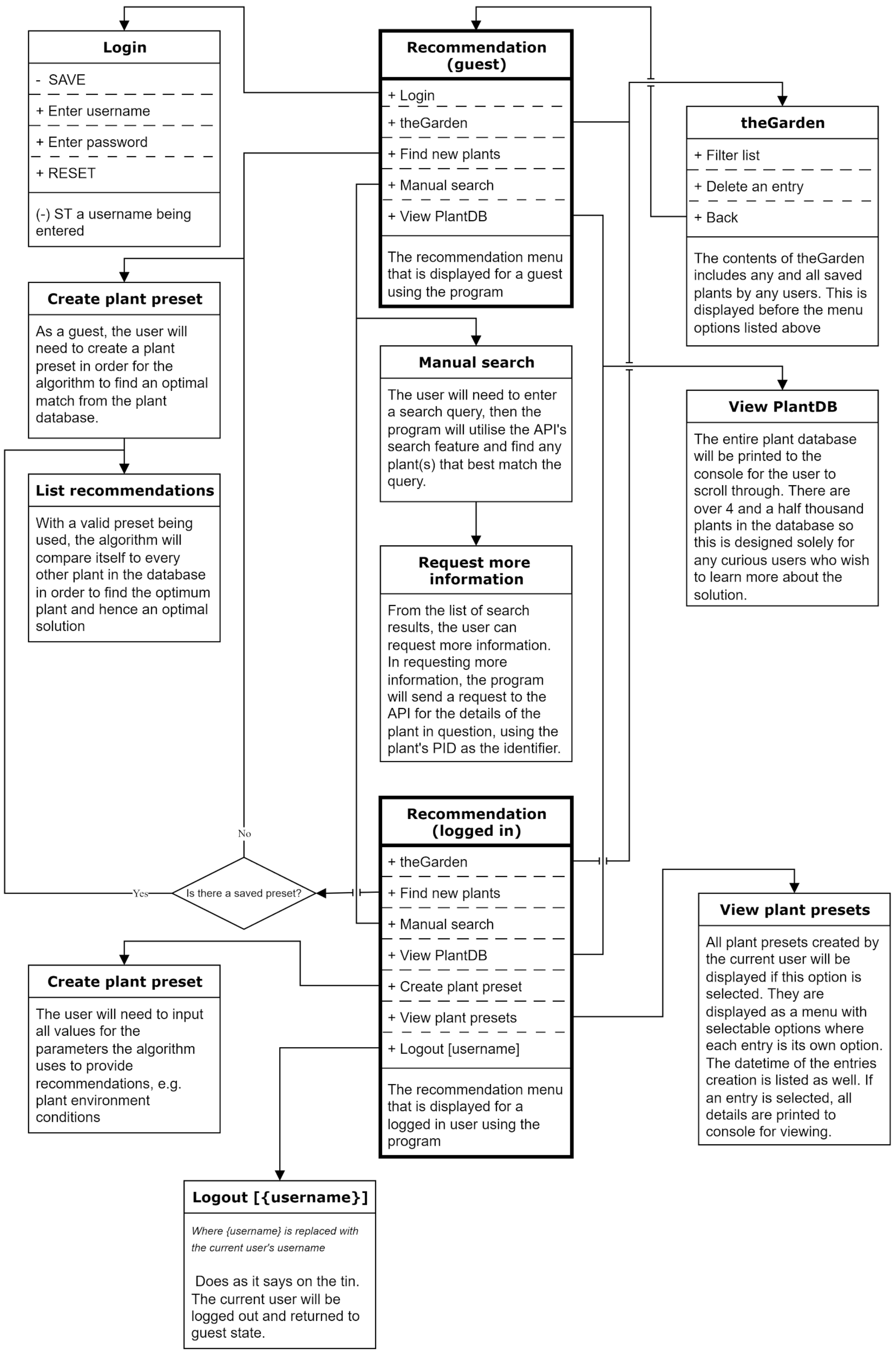
2.3.4 Menus

This section will break down the design and thought process for the menu system and the organisation of the menus. As a menu is the first thing the user will interact with in respect to my program, I believe that a crisp, clean, unbreakable and uncomplicated menu system would be in order.

The menu and menu option selection diagram is shown on the following pages. This should accurately represent the menu and options that are available to the user and could be considered as a more detailed flow diagram showing what will happen if the user selects a given option. The diagram gives simplified descriptions of each menu, simply to make it easier to understand the intention of each menu if the menu title was seemingly ambiguous, without going into so much detail that it clustered the diagram.


Key:
+ public
- subject to (ST) terms specified in description
Action / Menu title
+ option / action
+ option / action
Description
Program started
Startup
+ Instantiates all menus
+ Fetches client id and secret from config file
+ Sends a request for a token from API
+ Waits for response from API while printing loading screento console
Initial startup sequence of the program
Recommendation (guest)
+ Login
+ theGarden
+ Find new plants
+ Manual search
+ View PlantDB
The recommendation menu that is displayed for a guest using the program
SEE SEPARATE FLOW DIAGRAM
Dot product
Takes in two vectors, and calculates the dot product of the two vectors.
The two vectors are forced to be dimensionally consistent
Magnitude (of vector)
Takes in a single vector and will calculate the magnitude of the vector for the user.
Welcome Menu
+ Recommendations
+ Manual Calculate
+ Support
+ Exit
The 'main menu' of the program, all menus should be able to make their way back to this menu one way or another in order to make navigation between menus easier
Manual Calculate
+ Dot product
+ Magnitude (of vector)
+ Angle between two vectors
For the user to test calculations and to demonstrate what happens behind the scenes
Every option in this menu initially starts by taking in one (or two) vectors
Angle between
Takes in two vectors, and calculates the angle between the two vectors.
The two vectors are forced to be dimensionally consistent
Exits program
Support Menu
+ Menu controls
+ SQL Information
Menu for providing extra information and help on the program
Controls
Displays the keyboard controls the program supports (specifically WASD and arrow key navigation) including descriptions of the controls
SQL Stats
Displays general stats about the SQL database the program uses. e.g. plantdb count, TheGarden count (saved plants count), registered users

Menu breakdown of the solution



Login
- SAVE
+ Enter username
+ Enter password
+ RESET
(-) ST a username being entered
Recommendation (guest)
+ Login
+ theGarden
+ Find new plants
+ Manual search
+ View PlantDB
The recommendation menu that is displayed for a guest using the program
theGarden
+ Filter list
+ Delete an entry
+ Back
The contents of theGarden includes any and all saved plants by any users. This is displayed before the menu options listed above
Create plant preset
As a guest, the user will need to create a plant preset in order for the algorithm to find an optimal match from the plant database.
Manual search
The user will need to enter a search query, then the program will utilise the API's search feature and find any plant(s) that best match the query.
View PlantDB
The entire plant database will be printed to the console for the user to scroll through. There are over 4 and a half thousand plants in the database so this is designed solely for any curious users who wish to learn more about the solution.
List recommendations
With a valid preset being used, the algorithm will compare itself to every other plant in the database in order to find the optimum plant and hence an optimal solution
Request more information
From the list of search results, the user can request more information. In requesting more information, the program will send a request to the API for the details of the plant in question, using the plant's PID as the identifier.
Recommendation (logged in)
+ theGarden
+ Find new plants
+ Manual search
+ View PlantDB
+ Create plant preset
+ View plant presets
+ Logout [username]
The recommendation menu that is displayed for a logged in user using the program
No
Yes
Is there a saved preset?
Create plant preset
The user will need to input all values for the parameters the algorithm uses to provide recommendations, e.g. plant environment conditions
View plant presets
All plant presets created by the current user will be displayed if this option is selected. They are displayed as a menu with selectable options where each entry is its own option. The datetime of the entries creation is listed as well. If an entry is selected, all details are printed to console for viewing.
Logout [{username}]
Where {username} is replaced with the current user's username
Does as it says on the tin. The current user will be logged out and returned to guest state.

Recommendation menu flow diagram

Candidate Number: **1266**
Centre Number: **58231**

As can be seen by the first flow diagram (*Menu breakdown of the solution*), the **Welcome Menu** seems to be the main focus of the menu system, which makes sense given how it is acting as a main menu, from which all other menus can be accessed. This design choice also allows all menus to be backtracked to this central menu.

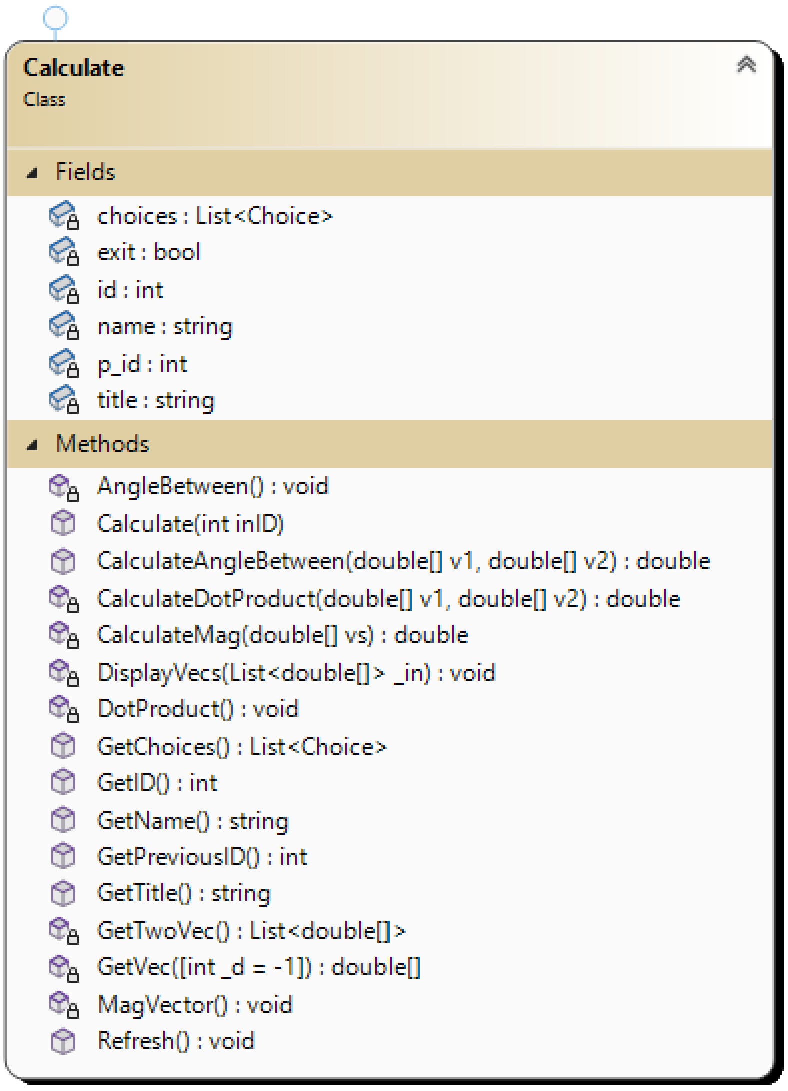
Additionally from the diagram, you can see that there is a startup menu which cannot be visited again aside from when a new instance is created. This startup is due to the API token request that is required before anything can be done with the API (see **2.3.5 API**).

2.3.4.1 Recommendation Menu

2.3.4.2 Calculation Menu

Otherwise known as "Manual Calculate"

2.3.4.2.1 Class



Calculate
Class

Fields
- choices : List<Choice>
- exit : bool
- id : int
- name : string
- p_id : int
- title : string

Methods
- AngleBetween() : void
- Calculate(int inID)
- CalculateAngleBetween(double[] v1, double[] v2) : double
- CalculateDotProduct(double[] v1, double[] v2) : double
- CalculateMag(double[] vs) : double
- DisplayVecs(List<double[]> _in) : void
- DotProduct() : void
- GetChoices() : List<Choice>
- GetID() : int
- GetName() : string
- GetPreviousID() : int
- GetTitle() : string
- GetTwoVec() : List<double[]>
- GetVec([int _d = -1]) : double[]
- MagVector() : void
- Refresh() : void

2.3.4.2.2 Dot product

2.3.4.2.2.1 Stack Frame

2.3.4.2.2.2 Overview

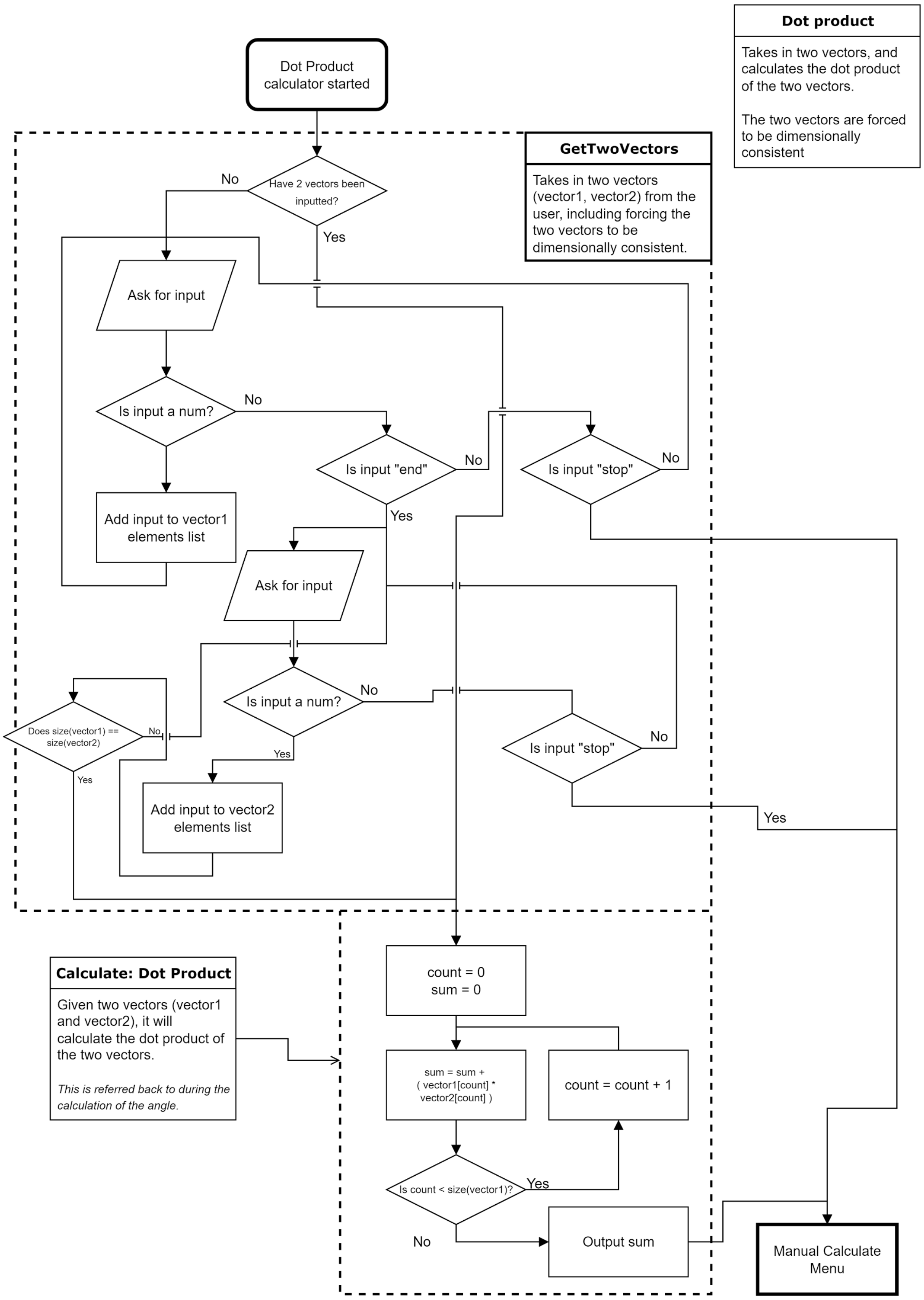
Selecting this option will allow the user to enter two vectors of a dimension of their choice. The first vector inputted will set the dimension limit, then the second vector will be forced to be dimensionally consistent.

Not only is this option interactive for the user for this menu, this function can also be used during the other calculations made - such as the angle between two vectors calculation that is used for my recommendation algorithm.

Below you will find a flow diagram of and also pseudocode for the dot product calculator. On this flow chart there are annotations to indicate the functions I am referring to above, in addition to this, please take note that some of these functions within the algorithm are used in following flow charts, however a reference will always be made when appropriate.

2.3.4.2.2.3 Pseudo

2.3.4.2.2.4 Flow chart


Dot product
Takes in two vectors, and calculates the dot product of the two vectors.
The two vectors are forced to be dimensionally consistent
Dot Product calculator started
GetTwoVectors
Takes in two vectors (vector1, vector2) from the user, including forcing the two vectors to be dimensionally consistent.
Have 2 vectors been inputted?
No
Yes
Ask for input
Is input a num?
No
Add input to vector1 elements list
Is input "end"
No
Yes
Is input "stop"
No
Ask for input
Is input a num?
No
Yes
Add input to vector2 elements list
Does size(vector1) == size(vector2)
No
Yes
Is input "stop"
No
Yes
Calculate: Dot Product
Given two vectors (vector1 and vector2), it will calculate the dot product of the two vectors.
This is referred back to during the calculation of the angle.
count = 0
sum = 0
sum = sum + (vector1[count] * vector2[count])
count = count + 1
Is count < size(vector1)?
Yes
No
Output sum
Manual Calculate Menu

Dot product algorithm flow diagram

Candidate Number: **1266**
Centre Number: **58231**

2.3.4.2.3 Magnitude (of vector)

2.3.4.2.3.1 Stack Frame

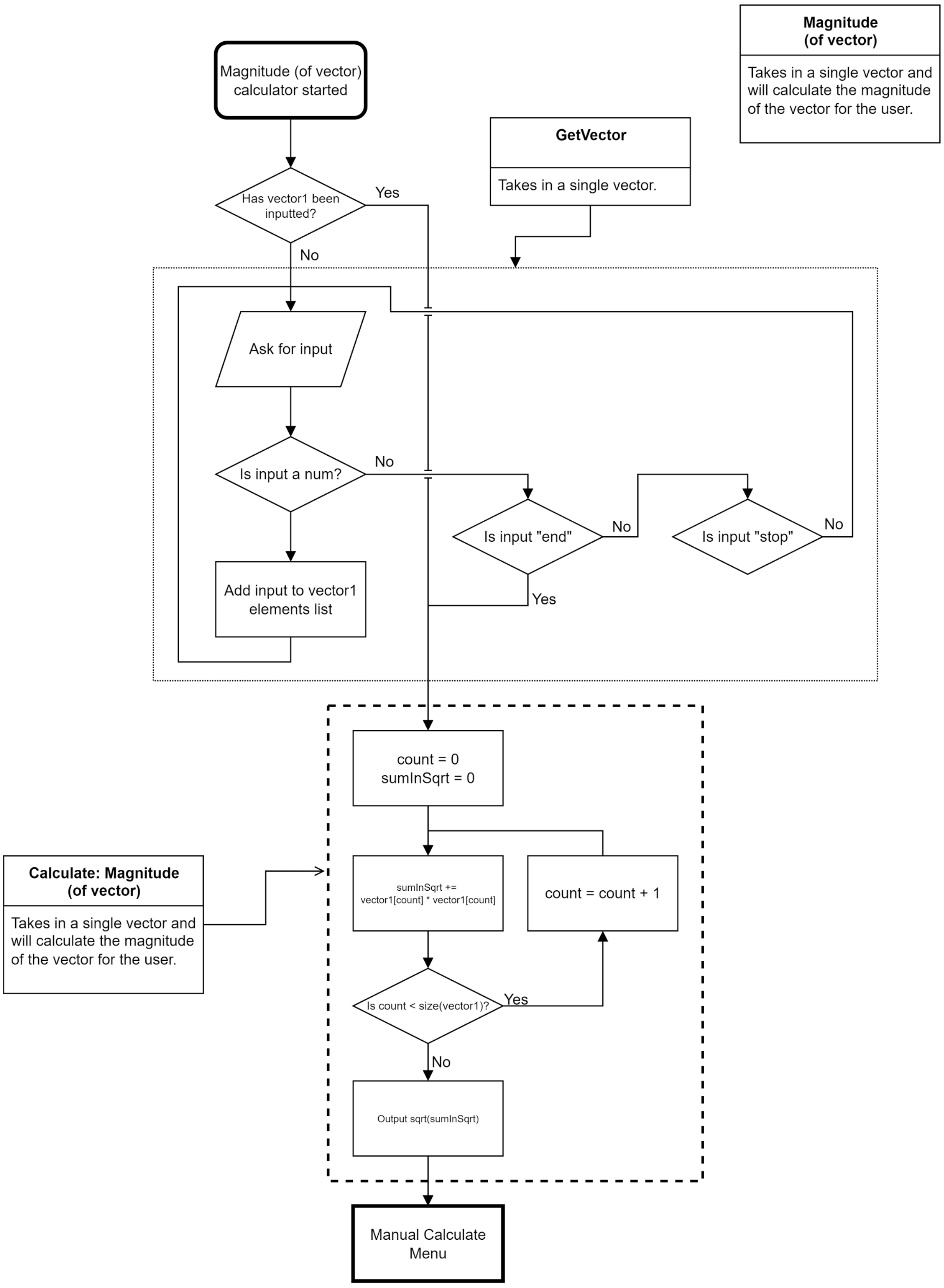
2.3.4.2.3.2 Overview

Similarly to the description of the option above, this option can be utilised as a standalone function, inputting the vector which you wish to calculate the magnitude of. However it does accept a valid user input when this option is selected through the menu.

Such as seen previously, there is pseudocode and a flow chart for this seen below. There are annotations once again on the flow chart in order to provide some more details where necessary. This is an abstraction of the actual algorithm, as it is intended to be as simplistic as possible for easy following, however the key points still get across and are addressed.

2.3.4.2.3.3 Pseudo

2.3.4.2.3.4 Flow chart


Magnitude (of vector)
Takes in a single vector and will calculate the magnitude of the vector for the user.
Magnitude (of vector) calculator started
GetVector
Takes in a single vector.
Has vector1 been inputted?
Yes
No
Ask for input
Is input a num?
No
Is input "end"
No
Is input "stop"
No
Yes
Add input to vector1 elements list
count = 0
sumInSqrt = 0
Calculate: Magnitude (of vector)
Takes in a single vector and will calculate the magnitude of the vector for the user.
sumInSqrt += vector1[count] * vector1[count]
count = count + 1
Is count < size(vector1)?
Yes
No
Output sqrt(sumInSqrt)
Manual Calculate Menu

Magnitude of a vector algorithm flow diagram

Candidate Number: **1266**
Centre Number: **58231**

2.3.4.2.4 Angle between two vectors

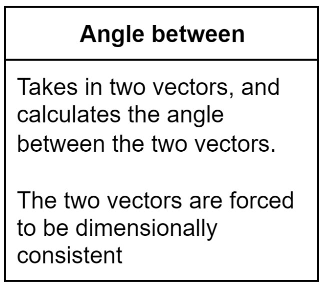
2.3.4.2.4.1 Stack Frame

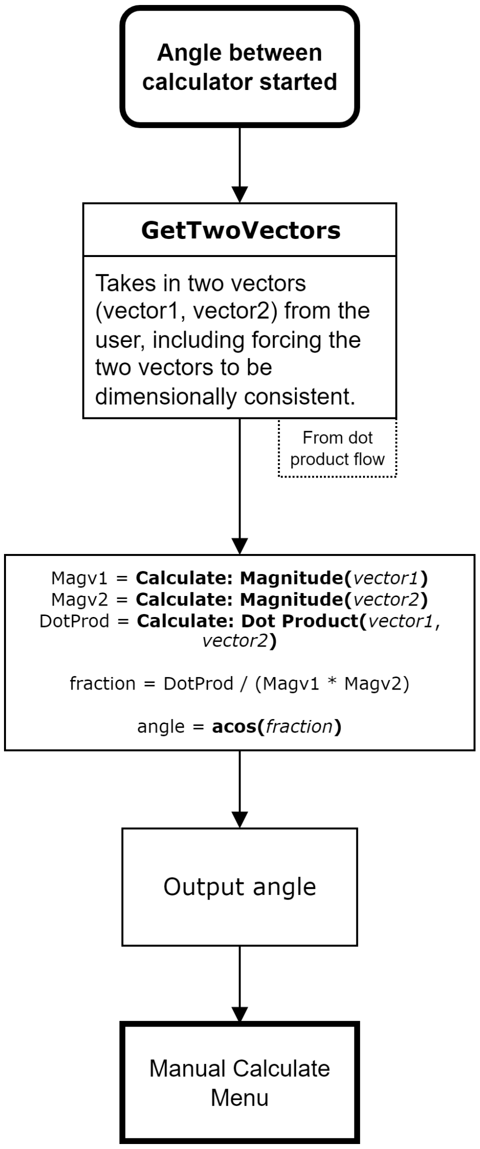
2.3.4.2.4.2 Overview

The flow diagram for this algorithm seems incredibly simple, however I believe this is due to the previously defined functions in the **Dot product algorithm flow diagram** in addition to the other functions in the **Product between two vectors algorithm flow diagram**. Each of these functions are quite significant segments of the previous flow diagrams, so if I had not continued with such abstraction this diagram would be far more complicated and less easy to follow.

2.3.4.2.4.3 Pseudo

2.3.4.2.4.4 Flow chart

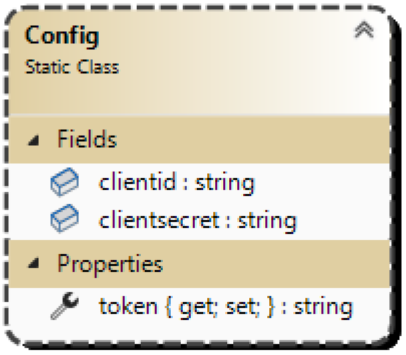

Angle between
Takes in two vectors, and calculates the angle between the two vectors.
The two vectors are forced to be dimensionally consistent


Angle between calculator started
GetTwoVectors
Takes in two vectors (vector1, vector2) from the user, including forcing the two vectors to be dimensionally consistent.
From dot product flow
Magv1 = Calculate: Magnitude(vector1)
Magv2 = Calculate: Magnitude(vector2)
DotProd = Calculate: Dot Product(vector1, vector2)
fraction = DotProd / (Magv1 * Magv2)
angle = acos(fraction)
Output angle
Manual Calculate Menu

Angle between two vectors algorithm flow diagram

2.3.5 API

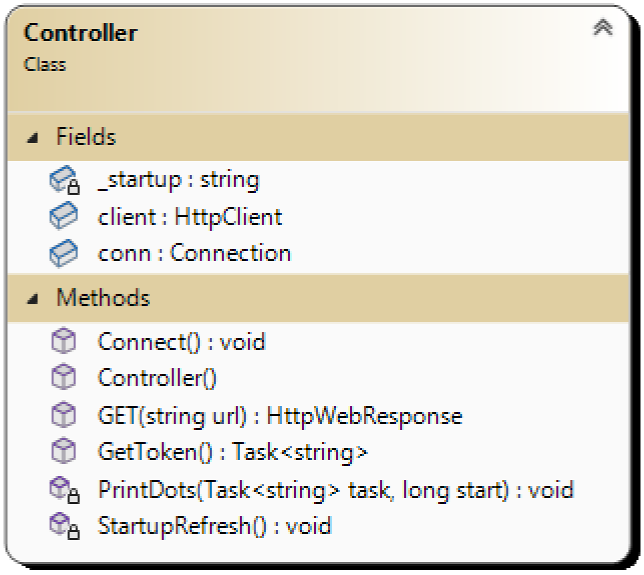
2.3.5.1 Configuration



Interactions with the API are initiated as soon as the program is started. The first thing the program does is instantiate the menus in which the client requests its token that will be used for the duration of that instance of the program.

There is a separate config file that contains the details needed for API interactions within the solution files that allows it to be changed without having to hardcode it into any other class, and is also clearly labelled and easily accessible for maintenance reasons as well. This config file holds the clientid and also clientsecret, which are both required to request the token from the API. Once this token has been received, it will be used in any request to the API from that moment onwards, and (from the limited documentation the API creators provided) I am lead to believe that this token will expire 30 days from when it was requested. I have not tested this - and I do not intend to - however for ease of use and to avoid any unnecessary complication, this token is requested for every instance of the program created (even if the previous instance was created a mere 17 seconds beforehand and the old token would still be valid).

2.3.5.2 Controller



```
_startup : string
client : HttpClient
conn : Connection
Connect() : void
Controller()
GET(string url) : HttpWebResponse
GetToken() : Task<string>
PrintDots(Task<string> task, long start) : void
StartupRefresh() : void
```

The API controller I set up consists of the client and then the connection. The string **_startup** is simply the ASCII art I intend to add for any major processes that may take some time to insure that the user is on the same page as the program and understands what is occuring.

2.3.5.2.1 Connect

The connect function does as its name suggests, and "connects" the controller to the API. In actual fact, this should just request the token and process the response, then update the config file with this new token in order for it to be referenced in GET requests. The **Connect()** function should start the initiation sequence of the connection, which includes calling upon the **PrintDots** subroutine (for the user's benefit) and the **GetToken** function which will both run asynchronously due to the use of multithreading.

2.3.5.2.2 GET

This is by far the most important function of this class. It will be used for every interaction past the initialisation of the program, and it is important that it can use the client credentials correctly.

3 Testing

3.1 Testing Table

TestID	Objective/Description	Test Data / Input	Expected Output	Pass/Fail	Evidence
Objective 1: User Interface / Interaction					
1.1 Menu / GUI / Choices					
1	The main menu displays options in a clear, structured way	-	Main Menu being displayed	Pass	0:05
2	Menus can be navigated through the use of the arrow keys	Arrow keys	Pointer navigates up and down the menu	Pass	0:09 0:56
3	Menus can be navigated through the user of the wasd keys	WASD keys	Pointer navigates up and down the menu	Pass	0:25
4	Cannot navigate beyond the bounds of the menu e.g. cannot move pointer to non-existent option	Up/Down arrow keys	Pointer stays on first or last option	Pass	0:14
5a	A choice in the menu can be selected through [ENTER]	ENTER key	Selects the current menu option	Pass	0:33
5b	A choice in the menu can be selected through [-->]	RIGHT ARROW key	Selects the current menu option	Pass	0:41
6	Filter menu options can be selected directly by pressing specific keys	[ENTER]	Opens the filter options list	Pass	0:43

7	Filter menu will simply refresh if an incorrect key is pressed	A string of random keys		Pass	1:11
8	When in a menu can exit back to the previous menu (including by option or arrow keys)	-		Pass	0:56
9	Capable of navigating back to the main menu (through previous menus) from any menu	-		Pass	0:56
10	Able to exit the program from a selectable option on the main menu	Selecting the option	Console should close as program is exited	Pass	1:31
11	Menus can be returned to without having progress reset due to switching menus (OOP implemented)	Half-register then browse other menus before revisitting and checking progress	Progress should be saved and can pick up from where the user left off	Pass	1:36
1.2 Display results					
12	When viewing a suggested plant, view basic information (e.g. name) in addition to the more complex information (e.g. moisture and light requirements)	-	-	Pass	2:43
13	Any plants suggested will also be provided with the algorithmic data reasoning for the suggestion	-	-	Pass	2:43

14	Displayed results are displayed with the option to save the plant to a garden	-	-	Pass	2:43
15	Elements displayed from a manual search are given the option to request more information from the API	-	-	Pass	2:32
1.3 User Gardens / Login					
16a	Allows registration of a new user with no password	Enters string for only username then saves	No errors thrown, new user appears in the list as they are added to the database	Pass	2:04
16b	Allows registration of a new user with a password	Enters string for username and password, then saves	No errors thrown, new user appears in the list as they are added to the database, including [LOCKED] as they are password protected	Pass	3:07
16c	Allows user to edit usernames during registration	-	Previously entered username is shown and can enter a new username	Pass	1:57
17a	Allows selection of available users from the database	-	List of usernames in the database	Pass	2:06
17b	Displays whether users	-	List of	Pass	2:06

	are locked (password protected)		usernames in the database with [LOCKED] next to users with passwords		
18	Locked users can only be logged into by using the associated password	-	-	Pass	3:30-3:58
18a	-	Incorrect password entered for the user	Asks if they wish to re-enter the password (does not allow access)	Pass	3:37
18b	-	Correct password entered for the user	Completes successful login of the user	Pass	3:49
19	Users may log out from the recommendation menu	Selection of the “logout” option in the menu	User is logged out, menus update as they are now a “guest”	Pass	2:14
20	Suggested plants can be saved to a garden of the user’s choice	-	User is given a choice of the gardens that are available to them	Pass	5:57
21	Plants can only be saved to a garden owned by the user	-	User is given only a choice of their own gardens	Pass	5:57

22	Only one of a plant can be saved to a garden	Attempt to save a plant that has already been saved to the garden	Program does not allow it, and states what the plant is saved as (label) so user can identify it	Pass	5:38
23	Plants can only be deleted from TheGarden by their owners	-	-	Pass	4:36
23a	A plant not owned by the user is attempted to be deleted	Inputs the ID of a plant that is not owned by the current user	Plant is not deleted. Program states if the plant is found at all and whether it is owned by the user	Pass	4:36
23b	A plant owned by the user is deleted	Inputs the ID of a plant that is owned by the current user	Plant is removed from TheGarden	Pass	4:58
24	Gardens can only be deleted by their owners				
25	Guests cannot save any plants to TheGarden	Attempt to save a plant while not logged in	The program will not save anything to TheGarden, and will state the reason why	Pass	5:14
26	TheGarden can be filtered by 3 options to limit the displayed	See below			

	results (to avoid future clutter when there may be hundreds of users and hundreds of entries)				
26a	Can filter TheGarden by plant (PlantPID)	-	Gives options of which PlantPID to filter by then filters by them	Pass	1:25
26b	Can filter TheGarden by Owner (OwnerID)	-	Gives options of which OwnerID to filter by then filters by them	Pass	0:47
26c	Can filter TheGarden by personal gardens (GardenID)	Selects the ID of a garden from a list provided	List of gardenID's provided, and then the list is successfully filtered by the user's choice of gardenID	Pass	6:22
27a	Users can create a plant preset in order to find new plants more easily	Informati on for the plant preset	A new plant preset is created for the user	Pass	3:58
27b	Users can view plant presets they have previously created	-	All their previously created plant presets should be displayed	Pass	4:15
1.4-5 User Input, Sanitisation					
28	Inputs from the user are only accepted when of	-	-	Pass	Found through-out

	the format required, such as: Integers Certain explicitly defined keys				entire video *Specific example* ***4:57***
28a	Entering string when asked for an ID (int)	String entered	Not accepted, does not continue	Pass	4:57
28b	Entering valid int	valid integer entered	Accepted, and proceeds as normal	Pass	5:06
29	SQL sanitisation of user's input must occur before the query is executed				
Objective 2: Data Handling					
2.1-2 API Interaction, handling responses					
30	The API configuration details (secret and ID) is read correctly for requests	(In config files of program)	No errors thrown due to lack of permission to make a request from the API	Pass	0:01 / 2:19
31	The API is called on initial startup in order to generate a valid token for the program to use		valid token received	Pass	0:04
32a	The API can be used to search for plants by a query inputted by the user	"aloe"	List of plants with names containing / relating to "aloe"	Pass	2:19
32b	The user's input is sanitised before being	(implied sanitisatio	Request made	Pass	(2:19)

	requested from the API	n or else would fail to make the request)	without errors		
2.3 Scraping & Caching					
33	The program populates an SQL database with plant entries from the API if one is not already present	-	A database is created / is checked that it exists in which it contains all of the plants expected	Pass	0:05 (When the main menu is instantiated it would have triggered the check)
34	The local database is accessed for recommendations in order to increase speeds of fetching plant data	-	(implied due to the speed of the recommendations, API interaction wouldn't allow such speeds)	Pass	2:43
Objective 3: Providing Recommendations					
3.1-2 Angle between two vectors, provide sorted list of plants					
35	Can calculate the angle between two inputted vectors	-	-		
35a	2D vectors inputted				6:58
35b	3D vectors inputted				7:22
35c	4D vectors inputted				7:45
35d	5D vectors inputted				8:22
36	Can calculate the dot product of two inputted vectors of the same dimensions	-	-		

36a	2D vectors inputted				9:02
36b	3D vectors inputted				9:23
36c	5D vectors inputted				9:33
37	Forces inputted vectors to be of the same dimension				START 6:58 END 10:45
38	Can calculate the magnitude of a vector of any dimension				9:54

Total # of tests: 54

Testing video

https://youtu.be/JtSuG8q5Y1w

4 Evaluation

☐

5 Code Base

5.1 Prototype

Program.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Newtonsoft.Json;

namespace prototype
{
    public class Program
    {
        public class PlantDetails
        {
            public string pid { get; set; }
            public string display_pid { get; set; }
            public string alias { get; set; }
            public int max_light_mmol { get; set; }
            public int min_light_mmol { get; set; }
            public int max_light_lux { get; set; }
            public int min_light_lux { get; set; }
            public int opt_light_lux;
            public int max_temp { get; set; }
            public int min_temp { get; set; }
            public int opt_temp;
            public int max_env_humid { get; set; }
            public int min_env_humid { get; set; }
            public int opt_humid;
            public int max_soil_moist { get; set; }
            public int min_soil_moist { get; set; }
            public int max_soil_ec { get; set; }
            public int min_soil_ec { get; set; }
            public string image_url { get; set; }
    public void Display()
    {
        Console.WriteLine("    pid: " + pid);
        Console.WriteLine("    display_pid: " + display_pid);
        Console.WriteLine("    alias: " + alias);
        Console.WriteLine("    max_light_mmol: " + max_light_mmol);
        Console.WriteLine("    min_light_mmol: " + min_light_mmol);
        Console.WriteLine("    max_light_lux: " + max_light_lux);
        Console.WriteLine("    min_light_lux: " + min_light_lux);
        Console.WriteLine("    max_temp: " + max_temp);
        Console.WriteLine("    min_temp: " + min_temp);
        Console.WriteLine("    max_env_humid: " + max_env_humid);
        Console.WriteLine("    min_env_humid: " + min_env_humid);
        Console.WriteLine("    max_soil_moist: " + max_soil_moist);
        Console.WriteLine("    min_soil_moist: " + min_soil_moist);
        Console.WriteLine("    max_soil_ec: " + max_soil_ec);
        Console.WriteLine("    min_soil_ec: " + min_soil_ec);
        Console.WriteLine("    image_url: " + image_url);

        Console.WriteLine("\n    opt_lux: " + opt_light_lux);
        Console.WriteLine("    opt_temp: " + opt_temp);
        Console.WriteLine("    opt_humid: " + opt_humid);
    }
}

public class PlantSearchResult
{
    public string pid { get; set; }
    public string display_pid { get; set; }
    public string alias { get; set; }
    public void Display()
    {
        Console.WriteLine("    pid: " + pid);
        Console.WriteLine("    display_pid: " + display_pid);
        Console.WriteLine("    alias: " + alias);
    }
}

public class PlantSearch
{
    public int count { get; set; }
    public string next { get; set; }
    public string previous { get; set; }
    public List<PlantSearchResult> results { get; set; }
```

```
        }

        protected static void Refresh(string token = "N/A")
        {
            Console.Clear();
            Console.WriteLine($"[PROTOTYPE] Plant Recommendation Program\nTOKEN -
    {token}\n");
        }

        public static async Task<string> GetToken() // async Task<string>
        {
            Console.WriteLine("Generating token...");
            HttpResponseMessage response; // string result;
            using (HttpClient httpClient = new HttpClient())
            {
                using (HttpRequestMessage request = new HttpRequestMessage(new
    HttpMethod("POST"), "https://open.plantbook.io/api/v1/token/"))
                {
                    MultipartFormDataContent multipartContent = new MultipartFormDataContent();
                    multipartContent.Add(new StringContent("client_credentials"), "grant_type");
                    multipartContent.Add(new
    StringContent("DNlEeLNQOpfYW3bTeMfY8v9b1mrMKGYfi6VGTfZC"), "client_id");
                    multipartContent.Add(new
    StringContent("KylrYac2J7OwCLfDChw8twiNoscP9wBtPzHeUKyYeIRnafrYfY2lPZ5yrubgxsYw
    asv7yno1B9ZkbAXvdIL534lhvq6TMNf0bEh8EHerbM3iFj3OD31ZoHpR8JDLtfqv"),
    "client_secret");

                    request.Content = multipartContent;

                    response = await httpClient.SendAsync(request);
                }
            }
            return response.Content.ReadAsStringAsync().Result.Substring(18, 30);
        }

        static void Main(string[] args)
        {
            Console.SetWindowSize(Console.WindowWidth + 16, Console.WindowHeight + 9);
            string token;
            Console.Title = "[PROTOTYPE] Plant Recommendation Program";
            Task<string> tokenTask = GetToken();
            tokenTask.Wait();
            token = tokenTask.Result;
            if (tokenTask.IsCompleted)
            {
```

```
            GetChoice(token);
        }
    }

    public static void PrintMenu(int n)
    {
        Console.WriteLine("Please choose an option:");
        Console.WriteLine("    [1]   Search for plant\n    [2]   Enter details for plant\n    [3]   Extra 
information\n    [ESC] Exit");
        Console.CursorLeft = 2;
        for (int i = 5; i < 9; i++)
        {
            Console.CursorTop = i;
            Console.Write(" ");
            Console.CursorLeft = 2;
        }
        Console.CursorTop = n;
        Console.Write(">");
        Console.CursorTop = 8;
        Console.CursorLeft = 0;
    }

    public static void GetChoice(string token)
    {
        ConsoleKey keyIn; int choice = 4;
        do
        {
            Refresh(token);
            PrintMenu(choice);
            keyIn = Console.ReadKey(true).Key;
            Console.WriteLine();
            switch (keyIn)
            {
                case (ConsoleKey.D1):
                    MakeChoice(0, token);
                    break;
                case (ConsoleKey.D2):
                    MakeChoice(1, token);
                    break;
                case (ConsoleKey.D3):
                    MakeChoice(2, token);
                    break;
                case (ConsoleKey.DownArrow):
                    if (choice < 7) choice++;
                    break;
                case (ConsoleKey.UpArrow):
                    if (choice > 4) choice--;
                    break;
                case (ConsoleKey.RightArrow):
                    MakeChoice(choice - 4, token);
                    break;
                case (ConsoleKey.Enter):
                    MakeChoice(choice - 4, token);
                    break;
                default:
                    Console.WriteLine("Please enter a valid input");
                    break;
            }
        } while (keyIn != ConsoleKey.Escape);
    }

    public static void MakeChoice(int choice, string token)
    {
        switch (choice)
        {
            case 0:
                SearchForPlant(token);
                break;
            case 1:
                UserInput(token);
                break;
            case 2:
                ShowExtraInfo();
                Console.WriteLine("\nPress anything to return to the main menu...");
                Console.ReadKey(true);
                break;
            case 3:
                Environment.Exit(0);
                break;
        }
    }

    public static void UserInput(string token)
    {
        userRecommendation userRec = new userRecommendation(token);
        userRec.GetInfo();
        Refresh(token);
        userRec.userPlant.Display();
        List<PlantDetails> userBest = userRec.GetOptPlant();
        double angle = userRec.angle;
```

```
            DisplayListPlantDetails(token, userBest, userRec, angle);
        }

        private static void DisplayListPlantDetails(string token, List<PlantDetails> plants,
userRecommendation userRec, double angle)
        {
            bool browsing = true;
            int current = 0;
            while (browsing)
            {
                Refresh(token);
                if (current < 1) { current = 1; }
                if (current >= plants.Count) { current = plants.Count - 1; }
                Console.WriteLine($"searchResult {current} of {plants.Count-1}:");
                plants[current].Display();
                Console.WriteLine("");
                userRec.userPlant.Display();
                Console.WriteLine($"angle: {angle}");
                if(angle == 0) { Console.WriteLine("Perfect match!"); }
                Console.WriteLine("\n\n - Use the left and right arrow keys to navigate through the list" +
                    //"\n - use the up arrow to view details on this plant" +
                    "\n - Use the down arrow to exit the list");
                ConsoleKey keyIn = Console.ReadKey(true).Key;
                switch (keyIn)
                {
                    case ConsoleKey.LeftArrow:
                        current--;
                        break;
                    case ConsoleKey.RightArrow:
                        current++;
                        break;
                    case ConsoleKey.DownArrow:
                        browsing = false;
                        break;
                }
            }
        }

        private static void ShowExtraInfo()
        {
            Refresh();
            Console.WriteLine("Units:");
            Console.WriteLine("light intensity:\n    Mmol = 602E15 photons per watt\n    When
measuring micromoles, you're counting how many photons are emitted per watt.\n    lux = lumen /
sq metre");
```

```
        Console.WriteLine("\nNatural light condition  -  typical lux");
        Console.WriteLine("    Direct Sunlight - 32,000 to 100,000");
        Console.WriteLine("    Ambient Daylight - 10,000 to 25,000");
        Console.WriteLine("    Overcast Daylight - 1000");
        Console.WriteLine("    Sunset & Sunrise - 400");
        Console.WriteLine("    Moonlight (Full moon) - 1");
        Console.WriteLine("    Night (No moon) - <0.01");
        Console.WriteLine("temp:\n    Celsius (C)"); // = (Fahrenheit - 32) * 5/9");
        Console.WriteLine("humidity:\n    Relative humidity (RH)");
    }

    public static void SearchForPlant(string token)
    {
        string searchTerm;
        do
        {
            Refresh(token);
            Console.Write("Search term needs to be greater than 3 chars\nEnter search term: ");
            searchTerm = Console.ReadLine();
        } while (searchTerm.Length < 4);
        Console.WriteLine("Searching...");
        PlantSearch searchResult = GetSearchResult(token, searchTerm);
        Refresh(token);
        if (searchResult.count < 1)
        {
            Console.WriteLine("searchTerm: " + searchTerm);
            Console.WriteLine("count: " + searchResult.results.Count);
            Console.WriteLine("No results to display");
        }
        if (searchResult.count == 1)
        {
            Refresh(token);
            Console.WriteLine("Fetching info...");
            PlantDetails temp = GetPlantDetailsByPID(token, searchResult.results[0].pid);
            Refresh(token);
            temp.Display();
            Console.WriteLine("\nPress anything to continue...");
            Console.ReadKey();
        }
        else
        {
            ConsoleKey keyIn;
            int choice;
            do
            {
                Console.WriteLine("searchTerm: " + searchTerm);
                Console.WriteLine("count: " + searchResult.results.Count);
                Console.WriteLine("View as:\n    [1]   List\n    [2]   Scrollable list\n    [ESC] Exit");
                keyIn = Console.ReadKey(true).Key;
            } while (keyIn != ConsoleKey.D1 && keyIn != ConsoleKey.D2 && keyIn !=
ConsoleKey.Escape);
            if (keyIn == ConsoleKey.D1) { choice = 1; }
            else { choice = 2; }
            if (keyIn == ConsoleKey.Escape) { Environment.Exit(0); }
            DisplaySearchResult(searchTerm, token, searchResult, choice);
        }
    }

    private static void DisplaySearchResult(string searchTerm, string token, PlantSearch
SearchResult, int choice)
    {
        if (choice == 1)
        {
            Refresh(token);
            Console.WriteLine("searchTerm: " + searchTerm);
            for (int i = 0; i < SearchResult.results.Count; i++)
            {
                Console.WriteLine($"{i + 1}:");
                SearchResult.results[i].Display();
            }
            if (SearchResult.results.Count == 0)
            {
                Console.WriteLine("Nothing found!");
            }
            Console.WriteLine("\n    [1] View as scrollable list\n    [ANY] exit");
            ConsoleKey keyIn = Console.ReadKey(true).Key;
            if (keyIn == ConsoleKey.D1)
            {
                choice = 2;
            }
        }
        if (choice == 2)
        {
            bool browsing = true;
            int current = 0;
            while (browsing)
            {
                Refresh(token);
                if (current < 0) { current = 0; }
                if (current >= SearchResult.results.Count) { current = SearchResult.results.Count - 1; }
                Console.WriteLine("searchTerm: " + searchTerm);
                Console.WriteLine($"searchResult {current + 1}:");
                SearchResult.results[current].Display();
                Console.WriteLine("\n\n - Use the arrow keys to navigate through the list" +
                    "\n - use the up arrow to view details on this plant" +
                    "\n - use the down arrow to exit the search");
                ConsoleKey keyIn = Console.ReadKey(true).Key;
                switch(keyIn)
                {
                    case ConsoleKey.LeftArrow:
                        current--;
                        break;
                    case ConsoleKey.RightArrow:
                        current++;
                        break;
                    case ConsoleKey.UpArrow:
                        Refresh(token);
                        Console.WriteLine("Fetching info...");
                        PlantDetails temp = GetPlantDetailsByPID(token,
SearchResult.results[current].pid);
                        Refresh(token);
                        temp.Display();
                        Console.WriteLine("\nPress anything to continue scrolling...");
                        Console.ReadKey(true);
                        break;
                    case ConsoleKey.DownArrow:
                        browsing = false;
                        break;
                }
            }
        }
        if (choice == 3)
        {
            Environment.Exit(0);
        }
    }

    public static PlantDetails GetPlantDetailsByPID(string token, string instr)
    {
        string urlStr = "";
        for (int i = 0; i < instr.Length; i++)
        {
            if (instr[i] == ' ')
            {
                urlStr += "%20";
            }
            else
            {
                urlStr += instr[i];
            }
        }

        string url = $"https://open.plantbook.io/api/v1/plant/detail/{urlStr}/";
        HttpWebRequest httpRequest = (HttpWebRequest)WebRequest.Create(url);

        httpRequest.Headers["Authorization"] = "Bearer " + token;
        string json;
        HttpWebResponse httpResponse = (HttpWebResponse)httpRequest.GetResponse();
        using (StreamReader streamReader = new
StreamReader(httpResponse.GetResponseStream()))
        {
            string result = streamReader.ReadToEnd();
            //Console.WriteLine(result);
            // debugging
            json = result;
        }
        PlantDetails newplant = JsonConvert.DeserializeObject<PlantDetails>(json);
        newplant.opt_humid = (newplant.max_env_humid + newplant.min_env_humid) / 2;
        newplant.opt_light_lux = (newplant.max_light_lux + newplant.min_light_lux) / 2;
        newplant.opt_temp = (newplant.max_temp + newplant.min_temp) / 2;
        return newplant;
    }

    public static PlantSearch GetSearchResult(string token, string instr)
    {
        string alias = "";
        for(int i = 0; i < instr.Length; i++)
        {
            if (instr[i] == ' ')
            {
                alias += "%20";
            }
            else alias += instr[i];
        }
        string url = $"https://open.plantbook.io/api/v1/plant/search?alias={alias}&limit=200";
        PlantSearch search = new PlantSearch();
        List<PlantSearchResult> results = new List<PlantSearchResult>();
        PlantSearch newplants = new PlantSearch();
        newplants.results = new List<PlantSearchResult>();
        bool run = true;
        do
        {
            HttpWebRequest httpRequest = (HttpWebRequest)WebRequest.Create(url);

            httpRequest.Headers["Authorization"] = "Bearer " + token;

            string json;
            HttpWebResponse httpResponse = (HttpWebResponse)httpRequest.GetResponse();
            using (StreamReader streamReader = new
StreamReader(httpResponse.GetResponseStream()))
            {
                string result = streamReader.ReadToEnd();
                //Console.WriteLine(result);
                // debugging
                json = result;
            }
            newplants = JsonConvert.DeserializeObject<PlantSearch>(json);

            for (int i = 0; i < newplants.results.Count; i++)
            {
                if (newplants.results[i] != null)
                {
                    results.Add(newplants.results[i]);
                    Console.WriteLine($"search: {results.Count}/{newplants.count} ({ 100 *
results.Count / newplants.count}%)");
                }
            }
            if (newplants.next == null) run = false;
            else { url = newplants.next; }
        } while (run);
        search.results = results;
        search.count = newplants.count;
        return search;
    }
}
}
```

userRecommendation.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Newtonsoft.Json;

namespace prototype
{

    public class userRecommendation : Program
    {
        public struct userOptPlant
        {
            public int lux;
            public int temp;
            public int humid;
            //public double soil_moist;
            //public double soil_ec;
            public void Display()
            {
                Console.WriteLine($"lux: {lux}\ntemp: {temp}\nhumid: {humid}");
            }
            public Vector vec;
        }
        public struct Vector
        {
            public double x; // lux
            public double y; // temp
            public double z; // humid
        }
        public struct neededPlantInfo
        {
            public string pid;
            public int lux;
            public int temp;
            public int humid;
        }
        public userOptPlant userPlant = new userOptPlant();
        public string recToken;
        public double angle;
        private PlantSearch allPlantSearch;
        public userRecommendation(string inToken) : base()
        {
            Refresh(inToken);
            Console.WriteLine("\nYou have started the user-specific plant recommendation program." +
                "\nThis will require information from you, and give you guidelines for the responses we
would expect." +
                "\nPlease be aware that this process may take some time depending on your system.");
            recToken = inToken;
            //allPlantSearch = GetSearchResult(recToken, "    ");
            allPlantSearch = getFullSearch();
            BinScrape();
            Console.WriteLine("Enter anything to continue...");
            #region debugging
            // debugging (checking)
            //for(int i = 0; i < families.Length; i++)
            //{
            //    Console.WriteLine(families[i]);
            //}
            //families = System.IO.File.ReadAllLines("plantGenus.txt");
            //string[] allLines = System.IO.File.ReadAllLines("wikiData.txt");
            //families = Format(allLines);
            //Console.WriteLine(families.Length);
            #endregion
            Console.ReadLine();
        }

        private PlantSearch getFullSearch()
        {
            PlantSearch allPlants = new PlantSearch();
            allPlants.results = new List<PlantSearchResult>();
            bool createFile = !File.Exists("allPlants.bin");
            Console.WriteLine(createFile);
            Console.ReadLine();
            if (!createFile)
            {
                try
                {
                    int aim;
                    string str = $"allPlants.bin";
                    using (BinaryReader reader = new BinaryReader(File.Open(str,
FileMode.OpenOrCreate)))
                    {
                        aim = reader.ReadInt32();
                        allPlants.count = aim;
                        reader.ReadString(); // reading off next and previous url (not needed)
                        reader.ReadString();
                        for (int i = 0; i < aim; i++)
                        {
```

Candidate Number: **1266**
Centre Number: **58231**

```
                PlantSearchResult temp = new PlantSearchResult();
                temp.pid = reader.ReadString();
                temp.display_pid = reader.ReadString();
                temp.alias = reader.ReadString();
                allPlants.results.Add(temp);
                Console.WriteLine($"Reading: {i}/{aim}");
            }
        }
        Console.ReadLine();
    }
    catch (Exception e)
    {
        Refresh();
        Console.WriteLine($"Caught an error:\n{e.Message}");
        Console.ReadLine();
    }
    if (allPlants.results.Count != 5534)
    {
        Console.WriteLine("Deleting file...");
        File.Delete("allPlants.bin");
        createFile = true;
        Console.ReadLine();
    }

}
if (createFile)
{
    PlantSearch activeSearch;
    Console.WriteLine("Creating file...");
    using (BinaryWriter writer = new BinaryWriter(File.Open("allPlants.bin",
FileMode.OpenOrCreate)))
    {
        activeSearch = GetSearchResult(recToken, "    ");
        int count = activeSearch.count;
        writer.Write(count);
        int i = 0;
        foreach (PlantSearchResult psr in activeSearch.results)
        {
            writer.Write(psr.pid);
            writer.Write(psr.display_pid);
            writer.Write(psr.alias);

            Console.WriteLine("writing: " + (i++) + "/" + (count - 1) + $" ({100 * i / (count -
1)}%)");
        }
```

```
            try
            {
                int aim;
                string str = $"allPlants.bin";
                using (BinaryReader reader = new BinaryReader(File.Open(str,
FileMode.OpenOrCreate)))
                {
                    aim = reader.ReadInt32();
                    allPlants.count = aim;
                    reader.ReadString(); // reading off next and previous url (not needed)
                    reader.ReadString();
                    for (int j = 0; j < aim; j++)
                    {
                        PlantSearchResult temp = new PlantSearchResult();
                        temp.pid = reader.ReadString();
                        temp.display_pid = reader.ReadString();
                        temp.alias = reader.ReadString();
                        allPlants.results.Add(temp);
                        Console.WriteLine($"Reading: {j}/{aim}");
                    }
                }
                Console.ReadLine();
            }
            catch (Exception e)
            {
                Refresh();
                Console.WriteLine($"Caught an error:\n{e.Message}");
            }
        }
        return allPlants;
    }

    return allPlants;
}

public List<PlantDetails> GetOptPlant()
{
    double smallestAngle = 2 * Math.PI;
    List<string> bestPID = new List<string>();
    int p = 0;
    #region commented out
    //for (int i = 0; i < families.Length; i++)
    //{
    //    Refresh(recToken);
    //    Console.WriteLine($"({i} * 100) / {families.Length} = " + (i / families.Length) * 100 +
"%");
            //    Console.WriteLine(i);
            //    Console.WriteLine(p);
            //    PlantSearch allPlantSearch = GetSearchResult(recToken, families[i]);
            //    foreach (PlantSearchResult psr in allPlantSearch.results)
            //    {
            //        p++;
            //        //psr.Display();
            //        PlantDetails currentPlant = GetPlantDetailsByPID(recToken, psr.pid);
            //        Vector currentVec = new Vector();
            //        currentVec.x = currentPlant.opt_light_lux;
            //        currentVec.y = currentPlant.opt_temp;
            //        currentVec.z = currentPlant.opt_humid;
            //        double currentAngle = GetAngle(userPlant.vec, currentVec);
            //        if (currentAngle < smallestAngle)
            //        {
            //            smallestAngle = currentAngle;
            //            bestPID = psr.pid;
            //        }

            //        #region DEBUGGING

            //        //Console.WriteLine($"\ncurrent: {psr.pid}\nangle: {currentAngle}\nsmallestAngle:
{smallestAngle}\n");

            //        #endregion

            //    }
            //}
            #endregion

            //PlantSearch allPlantSearch = GetSearchResult(recToken, "   ");
            List<neededPlantInfo> plants = BinRead();
            foreach (neededPlantInfo psr in plants)
            {
                //psr.Display();
                Vector currentVec = new Vector();
                currentVec.x = psr.lux;
                currentVec.y = psr.temp;
                currentVec.z = psr.humid;
                double currentAngle = GetAngle(userPlant.vec, currentVec);
                if (currentAngle < smallestAngle)
                {
                    smallestAngle = currentAngle;
                    bestPID.Clear();
                    bestPID.Add(psr.pid);
                }
                if (currentAngle == smallestAngle)
                {
                    bestPID.Add(psr.pid);
                }
                Console.SetCursorPosition(0,4);
                Console.WriteLine($"comparison: {p}/{plants.Count - 1} ({p++ * 100 / (plants.Count - 
1)}%)");
                #region DEBUGGING
                //Console.WriteLine($"\ncurrent: {psr.pid}\ncurrentbest: {bestPID}\nangle: 
{currentAngle}\nsmallestAngle: {smallestAngle}\n");

                #endregion

            }
            Console.WriteLine("Creating list...");
            List<PlantDetails> results = new List<PlantDetails>();
            for (int i = 0; i < bestPID.Count; i++)
            {
                Console.SetCursorPosition(0, 6);
                Console.WriteLine("                                        ");
                Console.SetCursorPosition(0, 6);
                Console.WriteLine($"progress: {i * 100 / bestPID.Count}%");
                results.Add(GetPlantDetailsByPID(recToken, bestPID[i]));
            }
            angle = smallestAngle;
            return results;
        }

        private double GetAngle(Vector v1, Vector v2)
        {

            double angle;
            double dotproduct = v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
            double magv1 = Math.Sqrt(Math.Pow(v1.x, 2) + Math.Pow(v1.y, 2) + Math.Pow(v1.z, 2));
            double magv2 = Math.Sqrt(Math.Pow(v2.x, 2) + Math.Pow(v2.y, 2) + Math.Pow(v2.z, 2));
            angle = Math.Acos(dotproduct / (magv1 * magv2)); // in rad
            //Console.WriteLine("Generating angle...");
            //Console.WriteLine($"v1: <{v1.x}, {v1.y}, {v1.z}>");
            //Console.WriteLine($"v2: <{v2.x}, {v2.y}, {v2.z}>");
            //Console.WriteLine("dotproduct: " + dotproduct);
            //Console.WriteLine("magv1: " + magv1);
            //Console.WriteLine("magv2: " + magv2);
            //Console.WriteLine("angle: " + angle);
```

```
            return angle;
        }

        public void GetInfo()
        {
            do
            {
                Refresh(recToken);
                Console.WriteLine("\nNatural light condition  -  typical lux");
                Console.WriteLine("    Direct Sunlight - 32,000 to 100,000");
                Console.WriteLine("    Ambient Daylight - 10,000 to 25,000");
                Console.WriteLine("    Overcast Daylight - 1000");
                Console.WriteLine("    Sunset & Sunrise - 400");
                Console.WriteLine("    Moonlight (Full moon) - 1");
                Console.WriteLine("    Night (No moon) - <0.01");

                Console.Write("Configuring lux:\n\nEnter an appropriate approx. average [LUX LEVEL]
for the plant: ");
            } while (!int.TryParse(Console.ReadLine(), out userPlant.lux) || !(userPlant.lux < 100000) ||
!(userPlant.lux > 0));

            do
            {
                Refresh(recToken);
                Console.WriteLine("\nRoom  -  recommended homeowner temperature");
                Console.WriteLine("    Bathroom - 22 to 24");
                Console.WriteLine("    Living Room - 20 to 22");
                Console.WriteLine("    Office Room - 20 to 22");
                Console.WriteLine("    Kitchen - 18 to 20");
                Console.WriteLine("    Bedroom - 16 to 20");
                Console.WriteLine("    Hallways etc - 15 to 18");

                Console.Write("\nConfiguring temp:\n\nEnter an appropriate approx. average [TEMP]
for the plant: ");
            } while (!int.TryParse(Console.ReadLine(), out userPlant.temp) || !(userPlant.temp < 30) ||
!(userPlant.temp > 5));

            do
            {
                Refresh(recToken);
                Console.WriteLine("\nContext of humidity  -  Relative Humidity (0-100%)");
                Console.WriteLine("    Average 'fair' house - 30 to 60");
                Console.WriteLine("    Mould occurs - >70");
                Console.WriteLine("    Poor low humidity - <25");
                Console.Write("\nConfiguring environment humidity:\n\nEnter an appropriate approx.
average [HUMIDITY] for the plant: ");
            } while (!int.TryParse(Console.ReadLine(), out userPlant.humid) || !(userPlant.humid > 0) ||
!(userPlant.humid < 100));

            userPlant.vec.x = userPlant.lux;
            userPlant.vec.y = userPlant.temp;
            userPlant.vec.z = userPlant.humid;
        }

        public void BinScrape(bool force = false, string instr = "scrape")
        {
            string filename = $"{instr}.bin";
            if (!force)
            {
                if (BinIsFull()) return;
                int actualNum = 0;
                if (File.Exists(filename))
                {
                    try
                    {
                        using (BinaryReader reader = new BinaryReader(File.Open(filename,
FileMode.OpenOrCreate)))
                        {
                            Refresh(recToken);
                            int count = reader.ReadInt32();

                            for (int i = 0; i < count; i++)
                            {
                                neededPlantInfo temp = new neededPlantInfo();
                                temp.pid = reader.ReadString();
                                temp.lux = reader.ReadInt32();
                                temp.temp = reader.ReadInt32();
                                temp.humid = reader.ReadInt32();
                                actualNum++;
                            }
                        }
                    }
                    catch (EndOfStreamException e)
                    {
                        Console.WriteLine(e.Message);
                        Console.WriteLine($"Writing failed.\nForced restarting initiating...");
                        BinScrape(true);
                    }
                }
            }

            Console.WriteLine("Creating file...");
            Refresh(recToken);
            using (BinaryWriter writer = new BinaryWriter(File.Open(filename,
FileMode.OpenOrCreate)))
            {
                int count = allPlantSearch.count;
                writer.Write(count);
                int i = 0;
                foreach (PlantSearchResult psr in allPlantSearch.results)
                {
                    PlantDetails currentPlant = GetPlantDetailsByPID(recToken, psr.pid);
                    writer.Write(currentPlant.pid);
                    writer.Write(currentPlant.opt_light_lux);
                    writer.Write(currentPlant.opt_temp);
                    writer.Write(currentPlant.opt_humid);

                    Console.WriteLine("writing: " + (i++) + "/" + (count - 1) + $" ({100 * i / (count -
1)}%)");
                }
            }
        }

        private bool BinIsFull(string instr = "scrape")
        {
            try
            {
                int aim;
                string str = $"{instr}.bin";
                using (BinaryReader reader = new BinaryReader(File.Open(str,
FileMode.OpenOrCreate)))
                {
                    aim = reader.ReadInt32();
                }
                if (BinRead().Count == allPlantSearch.count)
                {
                    return true;
                }
                return false;
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                return false;
            }
        }

        public List<neededPlantInfo> BinRead()
        {
            string str = "scrape.bin";
            List<neededPlantInfo> plants = new List<neededPlantInfo>();
            try
            {
                using (BinaryReader reader = new BinaryReader(File.Open(str,
FileMode.OpenOrCreate)))
                {
                    Refresh(recToken);
                    int count = reader.ReadInt32();
                    for (int i = 0; i < count; i++)
                    {
                        neededPlantInfo temp = new neededPlantInfo();
                        temp.pid = reader.ReadString();
                        temp.lux = reader.ReadInt32();
                        temp.temp = reader.ReadInt32();
                        temp.humid = reader.ReadInt32();
                        plants.Add(temp);
                        Console.SetCursorPosition(0, 3);
                        Console.WriteLine($"reading: {i}/{count - 1} ({100 * i / (count - 1)}%)");
                    }
                }
            }
            catch (EndOfStreamException e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("Rewriting...");
                BinScrape(true);
            }
            return plants;
        }
    }
}
```

5.2 Final Solution

References

[06-2022] - API being used: https://open.plantbook.io/
[06-2022] - https://reqbin.com/
[10-07-2022] - https://greenbusinesslight.com/resources/lighting-lux-lumens-watts/
[10-09-2022] -
https://www.vaillant.co.uk/homeowners/advice-and-knowledge/ideal-room-temperature/#:~:text=The%20average%20room%20temperature%20is,be%20heated%20to%20specific%20temperatures.
[10-09-2022] - https://www.youtube.com/watch?v=bbBGgHDhmVg
[16-09-2022] -
https://www.plants.com/p/peace-lily-plant-157654?afsrc=1&clickid=VLUUpqzijxyNT0H2N%3ASXpxqkUkDWOuURnWEy100&irgwc=1