Bloom x labs

SOFTWARE 2 // GROUP 2 // CFG Nanodegree Group Project

Laura Cullen, Laura Depla-Phang, Sarah Dowd, Bethan Hoare, Xanthe Sharp, Aimee Szkolar

INTRODUCTION

Bloom x labs is a gardening program designed to help newcomers and amateur gardeners. It will assist them to keep track of what they have planted and manage their plant care within a personalised plant logbook. They can also access advice on particular plant care, ideas of suitable plants for their plot and receive handy weather, pollen count and UV level information. The aim is to provide a comprehensive solution to many of the questions an amateur gardener would have, and automate the process of monitoring plant progress.

BACKGROUND

Login page: To access the program, the user must create an account. The user is given the choice to either login or create a new account. The username they provide is stored in a MySQL database, along with a user ID assigned to that user. There are error messages provided to handle an incorrect login, or a case where a user tries to create a new account with a username that already exists. A password is not required for this iteration of the program, only a username.

Homepage: Once the user has successfully logged in, they will be taken to the homepage and view a “Welcome” message with ascii art. From here, they will be offered a choice of options and can input their selection. The user can choose from numerous options: Search for Plants; View today’s weather; Get a Random Quote; Do a Plant Quiz or Exit page, the Plant Logbook page and the Weather Page. The user makes their selection by entering the corresponding number. This will navigate the user to the appropriate page.

Search and Store Page (Plant Search and Logbook): On the Search and Store menu, the user will be able to do do the following:

- **Search and add plant to logbook:** the user can search for plants by common name. The SQL database will be queried for any plant names that match or partially match that search. The user can then select a plant to add to their logbook, or exit out of the search function.
- **Remove plant from logbook:** The user can remove a plant from their logbook. They will be prompted to input the Plan tID number for the plant they want to remove. This will then be removed from the local logbook dictionary used whilst the program is running, as well as the logbook table on the database.
- **View logbook:** The user can view their current logbook with all of their logbook entries. The user will view the plant ID number, name, latin name, all requirements the plant has and the date the user planted it. IF the user does not have a logbook, it will return a message stating that no logbook is found for that user.
- **Exit:** The user can then exit out of the Search and Store menu to the Homepage menu.

Random Quote: If the user selects this option, they will see some decorative ascii art, followed by a Quote of the Day, generated by a function that prints a gardening-related quote from a custom API. We originally designed this feature to read from a text file (appendix) but it was felt that it would be easier for the end user, if this was an API rather than additional files that would need to be downloaded.

Plant Quiz: Acknowledging that the target audience of the program are newcomer/amateur gardeners who may feel overwhelmed with gardening information and options, the plant quiz is a fun addition to the program which utilises the same custom API as the plant search. Questions are posed to the user and, through their answers, a selection of plants is filtered down. Of those that match the user’s requirements, a single random plant is drawn and presented to the user with key information on its requirements (appendix).

Screenshot showing excerpt of quiz questions as seen by the user

```
Question 1:
First things first, what type of plant are you looking for?
a) Fruit
b) Flower
c) Shrub
d) Vegetable
e) Don't mind
Choose a, b, c, d or e
e
Question 2:
So thinking a bit about the planting location now, how much light does it get?
a) Full sun
b) Partial shade
c) Shade
d) Skip
Choose a, b, c or d
a
```

Screenshot showing the culmination of the quiz outputting a selected plant (chosen randomly from the user's filtered selection)

```
Your perfect plant pal is:
Geranium (Pelargonium). It's a flower, preferring full sun and well drained soil.
The best months for planting are March, April and May. It needs watering every 2 days.

Thanks for playing, hope this gave you some inspiration! Happy Planting :)
```

Weather Page: The user is prompted to enter their postcode and will then receive an overview of the current climate for the day, including:

- A description of the weather (sunny, raining etc) and the temperature
- UV levels
- Pollen count for trees, weeds and grass
- Advisory messages relating to the conditions, for example a warning to stay indoors if peak UV levels will be high or take precautions such as antihistamines if pollen levels are high.

The data from this section is retrieved from external APIs:

- The OpenWeatherMap API which retrieves the weather details.
- The Postcodes.io API which takes a postcode as input and returns the latitude and longitude of the location. This is not displayed to the user, but provides an input for the pollen count and UV level API requests.
- The Ambee API which returns the pollen count details.
- The OpenUV API, which returns the UV levels.

The user may enter 'm' to return to the main menu, or any other key to check the weather again.

SPECIFICATIONS AND DESIGN

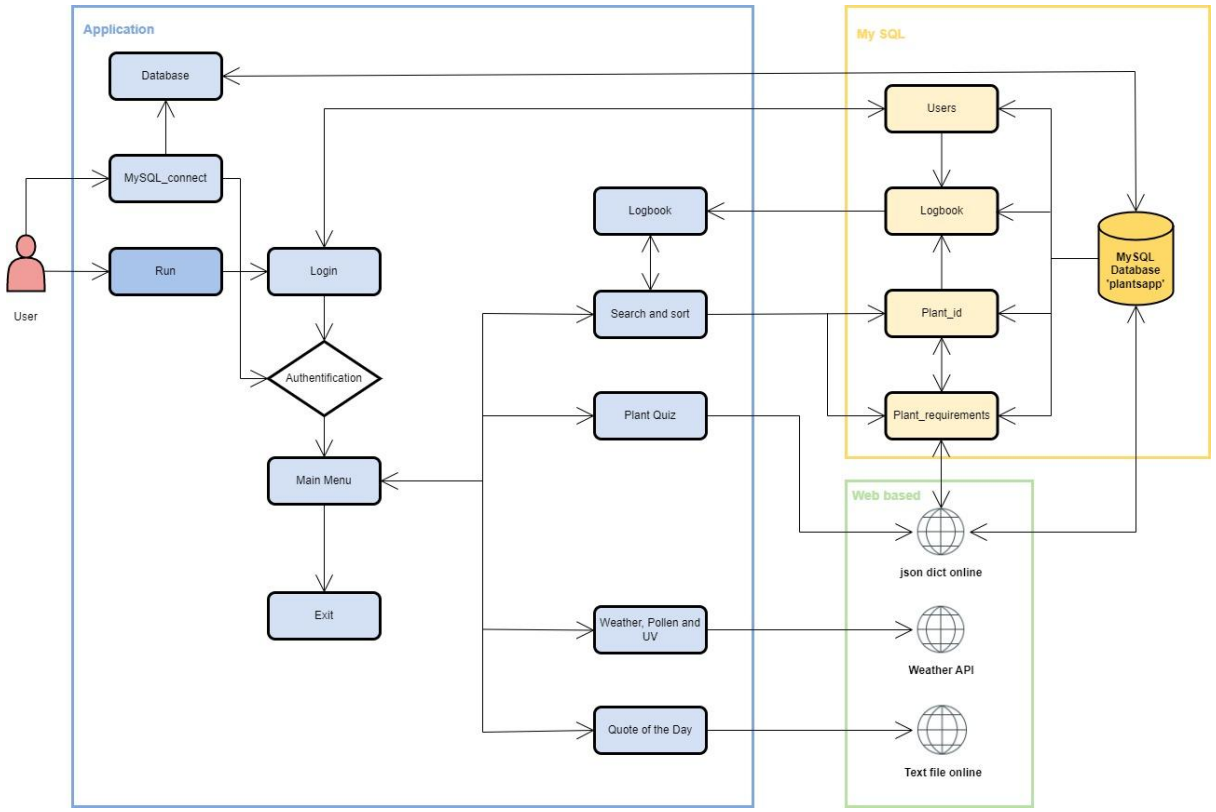
Non-Technical Requirements:

- Allows user to create an account with a password, and login and out of that account
- Saves user info in a database
- Saves plant details such as name, care guide and toxicity in a logbook once user selects this plant for inclusion
- Capability for user to search for plant information and get recommendations for plant types through a choice of quiz or standard search
- User can check the weather, pollen and UV for that day, based on location
- User can view a gardening-related quote of the day
- Visually engaging and friendly in tone

Technical Requirements

- MySQL database with logical tables
- Python code for program logic
- Custom API to get results from plant data
- Custom API to get random quote for ‘Quote of the Day’
- Plant data stored as json
- External APIs used for weather, pollen, latitude/longitude and UV data - requires API keys
- SQL queries used to fetch data
- Functions to be tested
- Errors such as incorrect inputs to be handled

Diagram to show the Architecture of the Application



IMPLEMENTATION AND EXECUTION : Timeframes

Task No.	Description	April		May				June
		1	2	3	4	5	6	7
1	**Initial Ideas**							
1.1	SWOT analysis for team members							
1.2	Decide on project idea							
2	**Project Statement**							
2.1	Write up objectives							
2.2	Write key features with must and wants							
2.3	Analysis and diagrams							
	Submit proposal			S				
3	**Coding**							
3.1	Create database on MySQL							
3.2	Find API's							
3.3	Base code							
3.4	Folder and file structuring							
3.5	Classes/ functions							
3.4	Divide code into sections							
3.5	Search/ logbook							
3.6	Homepage / login							
3.7	Weather							
3.8	Plant quiz/ quotations							
3.9	Database connect, create and populate							
4	**Test Code**							
4.1	Write test code for each section							
4.2	Run test and amend code							
5	**Project Work Document**							
5.1	Introduction							
5.2	Key section summaries							
5.3	Specifications and design							
5.4	Gantt Chart							
5.5	Program architecture							
5.6	Implementation and execution							
5.7	Testing and evaluation							
5.8	Diagrams							
	Submit project and report						S	
6	**Presentation**							
7.1	Create							
7.2	Present							

DEVELOPMENT APPROACH AND TEAM MEMBER ROLES

We worked initially by finding the requirements for the program, determining what we wanted it to do and completing individual flow diagrams to compare and contrast ideas. This produced a series of tasks that individuals in the team took on. We each populated the plant search API by focusing on a particular class of plant, such as vegetables, and using ChatGPT to generate json data based on this. All team members took on the role of Developer, focusing on different areas such as code to run the database queries, code to create the plant search, code for the weather search, code for the quiz and so on. Bethan took the role of Team Leader from weeks 2 to 3, and then Laura Cullen took the Team Leader role from week 3 onwards. Team members took responsibility for testing and exception handling for their own code, but as some functions were more challenging to test (such as those which interacted with a database), members with more coding confidence wrote those tests. With all roles, team members were ready to step in and help each other when required.

Laura Cullen

- Took Sarah's base code put code into smaller functions and classes
- Created a folder and file structure for the project
- Worked on the database connection file and database code
- Homepage(main menu) and login code
- Testing for the above
- Program architecture and gantt charts for report
- PM - Reviewing others code and git hub management

Laura Depla-Phang

- Made GitHub group repository
- Created plant data API
- Create quotes API
- Created plant quiz and random quote files
- Testing for the above
- In lieu of a GUI, sourced ASCII art titles and images to use throughout the program to enhance the user experience

Sarah Dowd

- Generated basic base code for the program.
- Wrote ChatGPT prompts for filling the .json file with plant information.
- Contributed towards the weather API section writing base code alongside Xanthe and combining the code from each base code files.
- Put Laura DP's plant quiz code into classes and changed the quotation python code to accept a URL .txt file.
- Contributed to work on the logbook and database connections (including testing for the logbook.py file).
- Helped with troubleshooting code and making sure the individual .py files all communicated correctly.
- Testing for login.py

Bethan Hoare

- Took Sarah's base code and sectioned it out and put it into a series of functions. Commented it out so we could understand which section is doing what.
- PM: tried different ways of recording tasks (trello, Slack). Kept a record of decisions made/questions we wanted to ask/tasks we'd assigned to each other. Admin around organising meetings.
- Worked on the initial plant search code, created functions for searching my name and by requirement. Separated into .py files.
- Worked on Aimee's logbook code to debug and add in features: recording date planted, updating local logbook and database logbook

- Helped troubleshooting teammate's code

Xanthe Sharp

- Conceived the original idea for the app and created a flow diagram to explore this
- Contributed towards plant data for the API
- Wrote code to explore how to create the database, including a numPy option
- Researched and tested external APIs for use in the program
- Wrote base code for the Weather page, to which Sarah added her code
- Refactored the combined weather code into smaller functions
- Wrote unit tests for the weather code
- Worked on the final report

Aimee Szkolar

- Contributing to logo design
- Helping for testing with both logbook and login
- Helping troubleshooting code

TOOLS AND LIBRARIES

We used the following libraries:

- Requests - to query external APIs
- Datetime - to put UV related data into readable format and for use in Plant Quiz alongside Calendar to obtain the current month
- PyMySQL - to connect to the mysql database
- Subprocess - to run multiple processes within the application
- Random - in both the Plant Quiz and Random Quote files to select random item from either a results file or an API
- Time - used in the Plant Quiz to give pauses between some printed output to enhance the user experience
- StringIO - used in testing to create object to store program's output to then see if 'x' was within it
- Unittest including TestCase, Mock, Patch - used in testing for mocking user input

We used classes and functions to organise the code and also decorators.

- Classes - Parent and child classes -for the search functionality and database connective, parent classes were used to avoid repetition within the child classes.
- Function - All code is broken down into functions to reduce repetition and increase useability and readability. This also helped with testing, as the code is easier to debug.
- Decorators - to enhance the features of functions without modifying them.

IMPLEMENTATION PROCESS

The initial gardening program idea was based on using an external API to get plant data from. It was a stretch goal to implement our own plant data API, and a learning curve for the team, so to do this was a real achievement. We found collaborating using gitHub a challenge, as there were times when several of us were working on the same code and creating numerous pull requests, which got confusing. Laura Cullen took responsibility for sorting the repo and arranging it into an organised file structure. We also became better at working on specific, agreed areas of the project so as to avoid duplication of work. Given more time, we would have liked to use Flask to create a front-end user interface, but incorporating the ascii art means that we have still created a pleasing and novel user experience.

AGILE DEVELOPMENT

We used an iterative approach to development, aiming to create a minimum viable product that would include a plant search with a log book and a weather search. After the first week which involved planning and investigating how to produce the plant search API, we were comfortably able to deliver a working product within the next two weeks. This meant that we could then focus on implementing classes, bug fixing, testing and adding nice-to-haves such as the plant quiz, quote generator and ascii art. We held frequent meetings both after classes and on weekends. Whilst it was sometimes hard to find a time that everyone could make, Bethan recorded meeting notes to share on Slack so everyone was up to date with progress. We used Slack for daily communication, to allocate tasks and to share code. We reviewed each other's code and at times, did some pair programming, splitting the team to work on different sections. We trialled using Trello as a scrum board, but found that with the small scale of our project, Slack was adequate for our needs and felt more intuitive.

TESTING AND EVALUATION

Our approach with testing was to unit test every function within the different sections, testing for a case where the function runs successfully, a case where the function fails, and ideally a corner case. We would then test the 'Run' functions to ensure that end-to-end testing was completed. We asked family members and friends to try out the program to ensure it is user friendly, whatever someone's level of technical ability.

CONCLUSION

This was an enjoyable yet challenging project that created a useful program. The Bloom X Labs application solves several gardening conundrums in one product, while providing an entertaining user experience. Given more time, we would aim to implement a front end. We would also increase the amount of data stored in the plant database, to ensure that plenty of data is available for users to query. Further ideas to develop areas such as the plant quiz, include taking the final output (a plant chosen randomly from the user's filtered results) and having the ability to add this to a new list, perhaps a 'wish list' or a 'shopping list' distinct from the logbook.

APPENDICES

Appendix A: Use of libraries such as pymysql, requests, also use of classes

```
import pymysql
import requests
from databases.mysql_connect import MYSQL_HOST, MYSQL_USER, MYSQL_PASSWORD


# Database management
------------------------------------------------------------------------
----------------------
class DatabaseManagement:
    def __init__(self, host, user, password, database=None):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connection = None
        self.cursor = None
```

Appendix B: Use of decorator and classes, with try/except block

```
class PlantDataManager:
    @staticmethod
    def connect_to_plantapp():
        try:
            connection = pymysql.connect(
                host=MYSQL_HOST,
                user=MYSQL_USER,
                password=MYSQL_PASSWORD,
                database='plantsapp'
            )
            return connection
        except Exception as exc:
            print(f'Error in connecting to db: {exc}')
```

Appendix C: Use of external APIs

```
def get_weather_data(api_key, latitude, longitude):
    url =
f'http://api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longit
ude}&appid={api_key}'
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        weather_data = {
            'description': data['weather'][0]['description'],
            'temperature': data['main']['temp'] - 273.15,  # Convert from
Kelvin to Celsius
            'humidity': data['main']['humidity']
        }
```

```
        return weather_data
    else:
        raise Exception(f"Error fetching weather data:
{response.status_code}")
```

Appendix D: Example of testing

```
from unittest import TestCase, mock, main
from todays_weather import get_lat_lng_from_postcode, get_weather_data,
get_uv_data, \
    get_pollen_count, generate_temp_advice, generate_uv_advice,
generate_pollen_advice, \
    get_all_details, weather_information, uv_information,
pollen_information, print_summary

import todays_weather

class TestGetLatLngFromPostcode(TestCase):

    def test_valid_postcode(self):
        expected = (53.402636, -2.222864)
        result = get_lat_lng_from_postcode('M20 5NT')
        self.assertEqual(expected, result)

    def test_invalid_postcode(self):
        with self.assertRaises(Exception):
            get_lat_lng_from_postcode('230 MGG')
```

Appendix E: Excerpt of the Plant Data json

```
{
 "F001": {
    "plant_name": "Rose",
    "latin_name": "Rosa",
    "class_name": "Flower",
    "light_conditions": "Full sun",
    "frequency_watering": 2,
    "soil_type": "Well drained",
    "minerals_required": [
      "Nitrogen",
      "Phosphorus",
      "Potassium"
    ],
    "planting_season": [
      "March",
      "April",
      "May"
    ],
    "growing_time": 90,
    "toxicity": "None"
 },
 "F002": {
    "plant_name": "Lily",
```

```
  "latin_name": "Lilium",
  "class_name": "Flower",
  "light_conditions": "Partial shade",
  "frequency_watering": 3,
  "soil_type": "Well drained",
  "minerals_required": [
    "Phosphorus",
    "Potassium"
  ],
  "planting_season": [
    "March",
    "April",
    "May"
  ],
  "growing_time": 60,
  "toxicity": "Animals"
},
```