INTRODUCTION

The primary aim of this project is to create a user-friendly website that helps dog owners find dog-friendly locations. This platform will address the growing need for reliable information about places where dogs are welcome, allowing owners to include their furry friends in their daily activities and travels.

Objective and Aims

Our main objective is to create a search functionality that allows users to find dog-friendly locations based on a given location. We will integrate the Yelp API to provide the dog-friendly establishments with additional information such as reviews, opening hours and contact hours. If time permits, we will then aim to integrate Google Maps API to provide the user with directions to their chosen location.

We aim to address the issue of dogs being left behind, rehomed or abandoned by providing a resource that encourages owners to bring their dogs along on outings. Ultimately, we hope to support the wellbeing of dogs by helping owners find places their dogs are welcome to join and promote dog-inclusive lifestyles.

The Team

Adele, Chloe, Yasmin, Jessica, Ellie

BACKGROUND

The inspiration for this project stems from the noticeable increase in dog ownership during the COVID-19 pandemic. With many individuals working from home or being furloughed, people looked towards dog adoption for companionship. However, more recently the world has returned to normalcy and lots of people have busy schedules. This has resulted in a struggle between balancing dog care with daily responsibilities. There is a growing need for solutions that enable dog owners to integrate their dogs into their lifestyle.

Our website aims to provide a solution to this problem by providing users with dog-friendly locations. The website will have a simple user interface that is easy to use to ensure a seamless user experience. This solution encourages responsible dog ownership as well as mitigating issues related to dog abandonment by making it easier for dog owners to include their dogs in their activities.

SPECIFICATIONS AND DESIGN

Functional, non-functional and extended requirements

Functional

1. Sign In
    - Users can create an account.
    - Users can sign in to the account.
    - Users can sign out of their account.

2. Search
    - Users can search dog friendly establishments by location.
    - Users can search by using their current location provided by the browser.

3. Information
    - Website will display important information related to the restaurant such as the address and contact information.
    - Information is saved for the user.

Non-functional

1. Performance
    - Website must load within 5 seconds of accessing the page.
    - Load search results within 5 seconds of request.

2. Usability
    - The design will be user-friendly, intuitive and pleasant to look at.
    - Should be accessible on desktop.

3. Scalability
    - Designed to support additional features if desired.

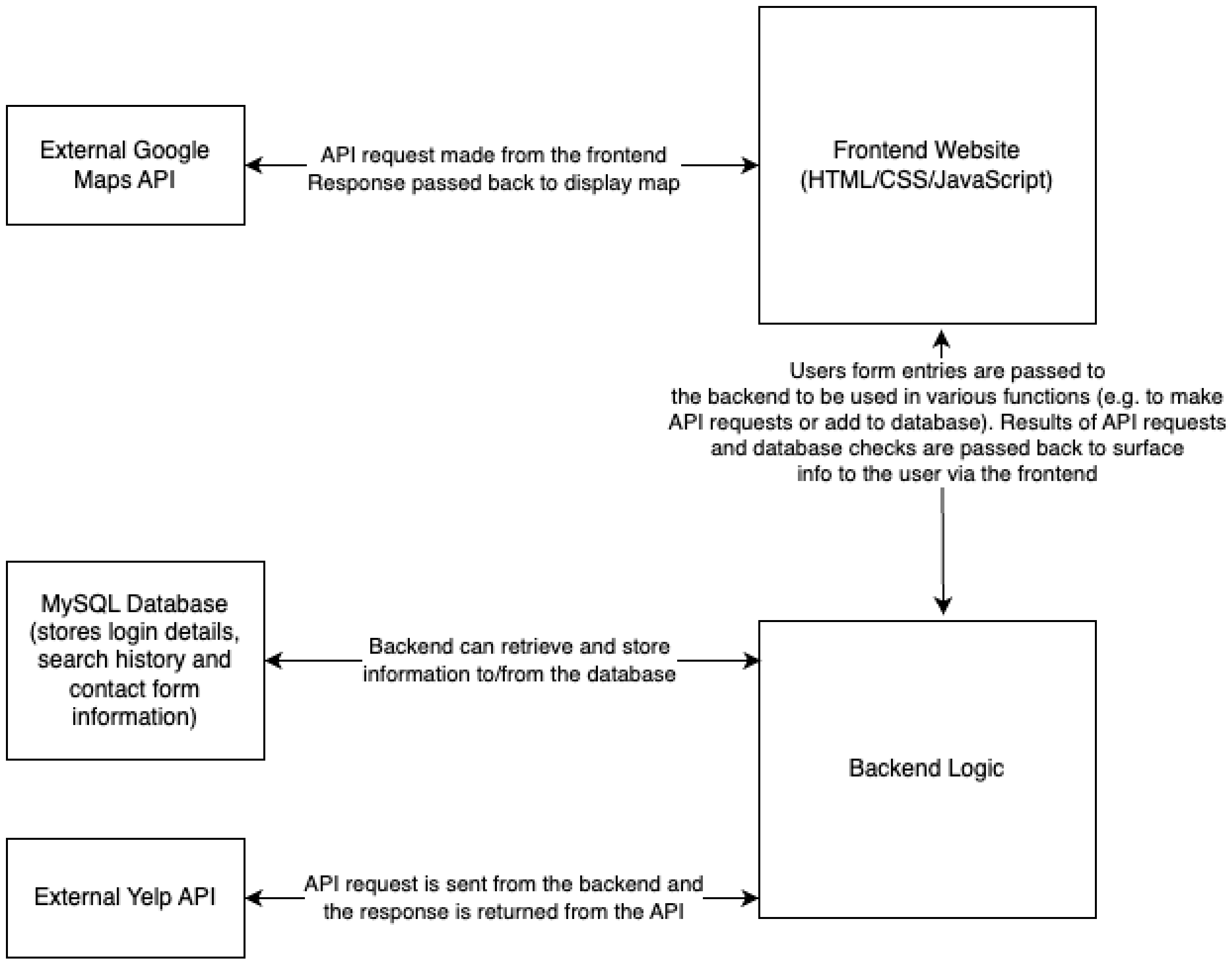
4. Maintainability
    - Best code practices so it is easy to maintain, update and scale.
    - Good documentation.

The functional and non-functional requirements were our minimal viable product, so were prioritized before any of the extended requirements.

Extended Requirements

1. Should be accessible on mobile.
2. Map integration - the establishment's location will be shown on a map.
3. Users can filter the search results by cuisine, store type (restaurant, bar, cafe).
4. Display an image from the business.
5. Display establishment reviews.
6. Users have the ability to delete their account.

Design and architecture


External Google Maps API
API request made from the frontend
Response passed back to display map
Frontend Website (HTML/CSS/JavaScript)
Users form entries are passed to the backend to be used in various functions (e.g. to make API requests or add to database). Results of API requests and database checks are passed back to surface info to the user via the frontend
MySQL Database (stores login details, search history and contact form information)
Backend can retrieve and store information to/from the database
Backend Logic
External Yelp API
API request is sent from the backend and the response is returned from the API

IMPLEMENTATION AND EXECUTION

Implementation process

As a team we created a JIRA board to distribute work and allocate tasks to each team member. We delegated tasks based on each team member's strengths to ensure everyone is happy with their workload. We broke down the workload into smaller tasks each week to complete individually. Throughout the week we communicated via slack and shared code regularly to ensure everyone can discuss any problems they encounter. This way we solved issues when they arose as opposed to later down the line. Despite working individually on tasks, we took a group approach to solving any issues. Each team member used JIRA to update the team when a task was completed. We created a team GitHub repository, and we all cloned this and coded locally in our development environment on individual branches. We reviewed code via pull requests and put in time each week for a team meeting to talk about problems, updates and new tasks that need allocating. We tested our system by individually carrying out unit tests on our work for each task. This tested to see if our individual code was working successfully. After combining code with others, we then performed integration testing to ensure that person A's code is still working alongside person B's code. We performed both functional and non-functional tests regularly through the project.

Tools and Libraries

1. Front End
    - HTML, CSS, JS
    - Python using the flask framework

2. Backend
    - DB in MySQL

3. Information
    - Yelp API
    - Google API

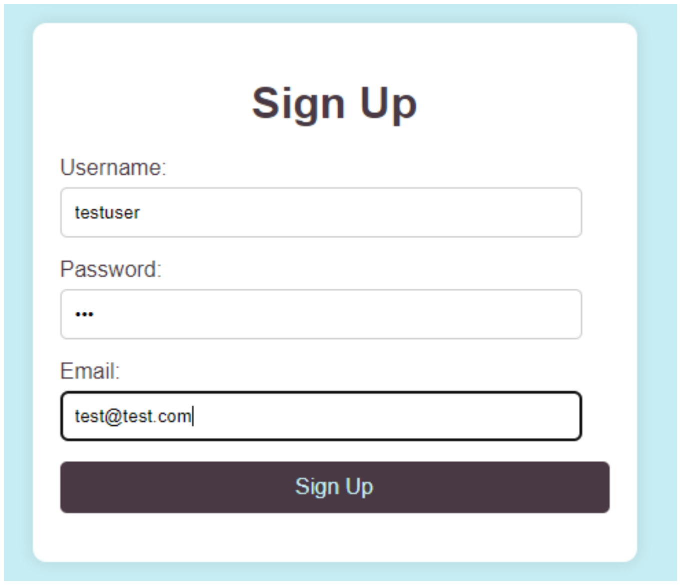
TESTING AND EVALUATION

Testing Strategy

The testing strategy for our project aimed to ensure that the system met all functional and nonfunctional requirements, identify defects so we could fix them, and validate the performance under various conditions. We employed several types of testing: unit testing using frameworks to verify endpoint responses, integration testing with Postman to ensure different parts of the application worked together, and system testing through manual verification of frontend and API interactions. Database testing was also conducted to ensure data integrity during login and sign-in processes. Manual testing played a crucial role in verifying that the frontend correctly interfaced with the backend, while automated testing with Postman efficiently validated API endpoints.

Functional and User Testing

Functional and user testing included scenarios such as user registration, login, data retrieval, and submission, ensuring that user inputs were accurately processed and stored. Below is an example of manually testing using the scenario of user sign up, a test users details are submitted using a form.



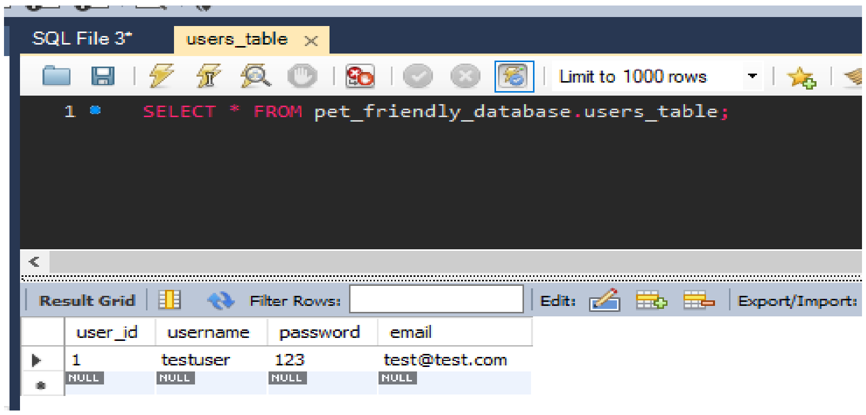
Sign Up

Username: testuser

Password: •••

Email: test@test.com

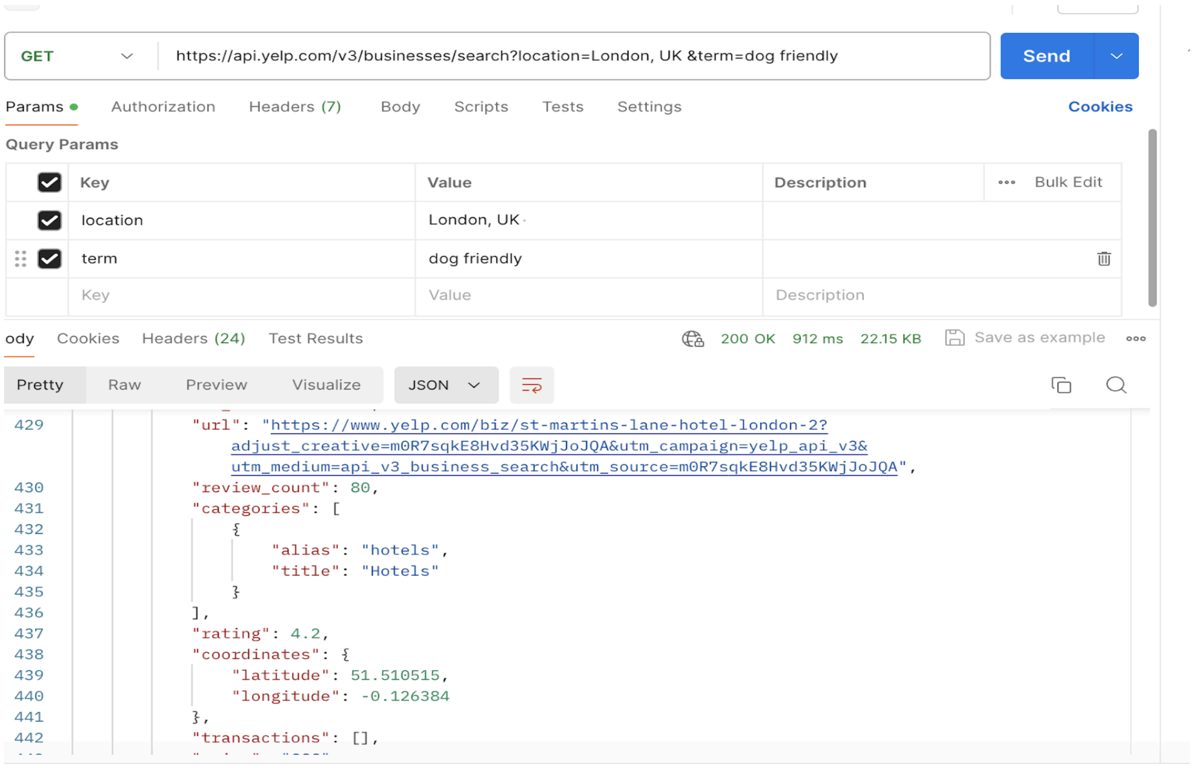
And the below sql query has been run to prove that the user has been added to the database



```
SELECT * FROM pet_friendly_database.users_table;
```

user_id	username	password	email
1	testuser	123	test@test.com
NULL	NULL	NULL	NULL

User testing involved manual checks to confirm the application's correct functionality and database interactions. This process verified that the frontend behaved as expected when interacting with the APIs and that data was correctly stored in the database. Below is an example of testing API interaction using postman, this allows us to retrieve the response from the API and quickly test different scenarios i.e. happy path and unhappy path.



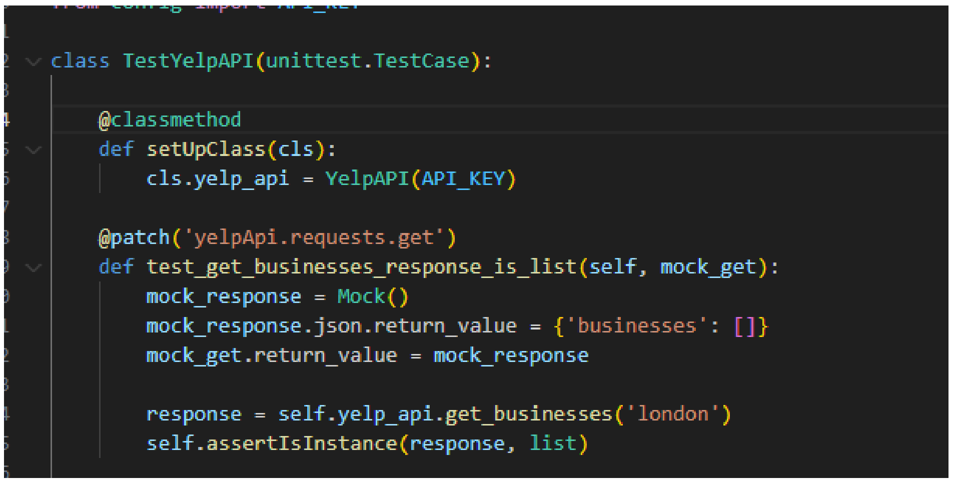
```
GET https://api.yelp.com/v3/businesses/search?location=London, UK &term=dog friendly
```

Query Params

	Key	Value	Description
✓	location	London, UK	
✓	term	dog friendly	
	Key	Value	Description

```
                "url": "https://www.yelp.com/biz/st-martins-lane-hotel-london-2?
                    adjust_creative=m0R7sqkE8Hvd35KWjJoJQA&utm_campaign=yelp_api_v3&
                    utm_medium=api_v3_business_search&utm_source=m0R7sqkE8Hvd35KWjJoJQA",
                "review_count": 80,
                "categories": [
                    {
                        "alias": "hotels",
                        "title": "Hotels"
                    }
                ],
                "rating": 4.2,
                "coordinates": {
                    "latitude": 51.510515,
                    "longitude": -0.126384
                },
                "transactions": [],
```

Basic unit tests for each of the API functions were added to test the responses from the functions, below is an example of testing the get_businesses function returning a list when a valid request is called.



```python
from config import API_KEY

class TestYelpAPI(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.yelp_api = YelpAPI(API_KEY)

    @patch('yelpApi.requests.get')
    def test_get_businesses_response_is_list(self, mock_get):
        mock_response = Mock()
        mock_response.json.return_value = {'businesses': []}
        mock_get.return_value = mock_response

        response = self.yelp_api.get_businesses('london')
        self.assertIsInstance(response, list)
```

For future improvements component or integration tests could be added in code, to cover more aspects of testing. I.e a component test could be added to create an in memory database and test the happy path of user registration, this would remove the need of manually testing through the frontend and checking in the database. A integration test could be added to call the YelpAPI endpoints to prove that the dependencies work and a response is received, this would remove the need of using postman to test the endpoint calls.

System Limitations

Known issues, such as minor UI inconsistencies, were documented. These inconsistencies did not impact the core functionality of the system. Future improvements were suggested to enhance user interface responsiveness and add more detailed error handling in subsequent updates.

CONCLUSION

To execute our project, we first determined what the most important features we wanted to include would be. We aimed for a minimum viable product which met all of the project criteria and ensured a fully functioning website that would allow the user to search dog-friendly locations and create an account to see their most recently searched locations. For the front-end, we wanted to ensure the website was straightforward and easy to navigate. For the back-end, our goal was to write readable code, using object-oriented programming (OOP) where possible, and also to establish a connection to a MySQL database for managing user information and search history.

Moving forward, there are several improvements we plan to make to enhance the user experience and to improve our back-end code. One such improvement would be to add advanced search filters to create a more practical website, allowing users to refine their search depending on the type of location they want to visit. We would also implement community features and social media integration so users can connect with fellow pet lovers in the area and share their experiences after visiting recommended locations.

Further improvements could be made by ensuring the website is accessible on various devices, including mobile phones and tablets. Additionally, we could include reviewing options for the users, so they can leave reviews and photos of locations they have visited, as well as a feature to save their favorite locations on their account to access any time. To make the website more helpful, we could include additional information such as tips and articles on pet health and safety, as well as nearby veterinary services or pet stores. Finally, we would love to extend the project further to allow users to look for various pet-friendly locations instead of just dog-friendly locations, so all pet-owners can enjoy using the website!

To improve our project code, we would store hashed passwords, which creates unique strings and improves the user security as no one would be able to access users’ passwords, even if they gained access to the database. Protecting users' data is crucial, and by doing this we will reduce the impact of any data breaches. We could further improve our code readability by breaking down larger functions into smaller and more manageable ones. As well as this, we could improve the consistency of the error handling and logging by raising and logging errors in the same format throughout the project code.