Tradespeople Near Me: Final Project Report

DT228
BSc in Computer Science

Aron O'Neill

C16466214

Jonathan McCarthy

School of Computer Science

Technological University, Dublin

11/4/20

Abstract

"One in four people are now renting their home in Dublin, the highest figure since quarterly records began" (1). This figure emphasises the volatility of the Irish housing market, with more and more people moving into new areas for accommodation.

With so many people now renting, it is unlikely that they know which tradespeople operate within their area. *Tradespeople* Near Me will be a user-friendly application making it easier than ever to connect consumers with their desired tradesperson. The aim of this application is to eliminate the stress associated with finding a tradesperson to carry out an urgent job. This web application will then act as a contacts directory for future use while providing for offline functionality.

Tradespeople Near Me will also be in the best interest of local tradespeople as it will act as a free advertisement for their services. In an age where very little is free, tradespeople should leap at the chance to have their own page displayed in this application as it will only help to increase business for tradespeople.

The continuous demands from consumers to employ tradespeople will see this application grow in popularity once deployed.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

_____

Student Name

Date: 11/4/20

Acknowledgements

I would like to take this opportunity to give a huge thank you to my friends, family, lecturers and most importantly, my supervisor, Jonathan McCarthy. Without the continuous support from this group of people, building this project would not have been possible. If it wasn’t for the goals set by Jonathan and I during our weekly meetings, none of this would have been possible. It has been a real pleasure working with such a devote supervisor who provided constructive feedback week on week. It was this engaging feedback which enabled me to conduct the required research and develop the prototype simultaneously.

Table of Contents

Table of Figures

1. Introduction

1.1. Project Background

Tradespeople Near Me is a web application which enables people to locate various professionals in their area for a particular problem.

As the property crisis continues to grow in Ireland, so does the need for property owners to employ tradespeople in their area who they are not familiar with. ”There are 895,600 people living in rented accommodation in Ireland and, as a proportion of the overall population at 18.9%, is now the highest since records began“ (2). This continuous growth in the number of people living in rented accommodation has resulted in a greater demand for a platform where individuals can search for tradespeople. A platform to facilitate this need would also benefit landlords as they simply cannot know a tradesperson for each property location. Despite the advancements in technology in modern times, most people still turn to a friend or family member to recommend a tradesperson. Therefore, the standard of work carried out by the suggested tradesperson predominately depends on your who you know. Tradespeople Near Me aims to help solve this issue as it will provide consumers with the necessary information to make an informed decision on who to contact. Consumers will be capable of viewing a portfolio of each tradesperson‘s previous work Thus, enabling consumers to make an educated decision before contacting their desired tradesperson.

The inspiration for this project came from my own experience renting in Dublin. Our apartment developed a serious leak during the summer of this year. After contacting our landlord, he informed us that he and a plumber would arrive first thing in the morning. Upon arrival, I was amazed to see that the landlord had brought a plumber he had known from his own county, Cavan. After completing the job, the plumber was then entitled to the cost of the job itself, as well as the cost of the 3-hour round commute. It was obvious from this experience and many others, that people do not tend to stray from their tried and trusted tradespeople. Some people would prefer to pay the extra travelling fee for that added peace of mind, got from dealing with a professional whom they have dealt with before. With this in mind, one of the aims of this application is to provide a safe and reliable

option for consumers who would rather employ a local tradesperson and benefit financially in the meantime.

Another issue of existing solutions for contacting and employing a tradesperson is the response time between initially contacting them, and arrival time. For those emergency call-outs from a tradesperson such as a plumber, they will have to fill out a form detailing their job to be carried out. For those emergency call-outs This response delay could prove costly, depending on the job that needs to be carried out. Subsequently, another goal of this application is to improve the response rime between connecting a consumer to a tradesperson.

1.2. Project Description

A simple, user-friendly web application will act as a medium between tradespeople and consumers. This application will therefore enable users to login as either a tradesperson or a consumer. There will also be an admin whom will ultimately control the data being displayed to the consumer. Tradespeople Near Me will act as a platform that provides tradespeople with an opportunity to advertise their services in the form of a web page, with this web page containing helpful information for visiting consumers. It will also illustrate to the consumer, the diameter in which each individual tradesperson is willing to travel to carry out a job. Not only this but it will give them an opportunity to display pictures of their previous work as well as their contact information. This will be implemented in the form of a portfolio so that registered tradespeople can tailor their page to their specific requirements. Upon viewing these tradespeople, consumers will have the opportunity to save their details into a favourites page so that they can be easily accessed again. This project will take the form of a progressive web app (PWA) and will contain various web pages so that consumers can query every possible tradesperson in their area. Storing a tradesperson as a favourite will result in it being stored in cache on the device being used. This will be accomplished using a service worker and will allow for offline use.

Tradespeople Near Me will provide consumers with the option to filter their search results based on the attributes they would like prioritized. Thus, taking them closer to the required

tradesperson. This project will be implemented as a progressive web app so that any user can access it cross platform. This will subsequently enable the application to reach the widest possible user base, with users being able to acquire it on any device, mobile or otherwise. Tradespeople Near Me will incorporate the use of a service worker so that consumers can access their desired contact information offline. This will ultimately allow the application to cater for the largest feasible target market.

This service will be launched in the coming year but prior to this, the relevant testing must be carried out. The application will then be evaluated by applying comprehensive usability testing which involves users continuously digging into the UI of the app to identify any usability issues. However, before this can occur, the appropriate research and development must be carried out.

1.3. Project Aims and Objectives

The overall aim of this project is to build a secure application which connects consumers and tradespeople. This will be achieved by developing a user interface as the application's front-end such that it connects to the necessary components of the application. Once the project is completed, the following objectives will be present in the application.

1. The overall objective of this application is to act as a medium between consumers and tradespeople so that it displays the necessary tradespeople contact information.
2. A back-end will be developed in order to store the necessary tradespeople attributes such as their maximum travel distance and contact information.
3. A scalable front-end will be implemented with consumer satisfaction in mind. The front-end will supply a login option for either, a consumer or a tradesperson.
4. Middleware will consist of AJAX requests so that the required template data can be passed to and from the application's back-end. Not only this but a service worker will also be implemented so that this information can persist once offline.
5. A cloud provider will be incorporated alongside container software in order to make Tradespeople Near Me deployable from one computing environment to another.

1.4. Project Scope

There are four main areas of development within this application, these include the application's front-end, middle-tier and its back-end. Once these areas have been sufficiently covered in the application's development, its focus will then turn to deployment.

1.4.1 Front-end

The front-end of this application will take the form of a progressive web application (PWA) which caters to all consumers. Building an application as a PWA gives it the look and feel of a native app as it provides offline functionality. The front-end functionalities provided to the user will include login, register, searching for a tradesperson, favouriting a tradesperson and enabling tradespeople to add and edit their own information.

Login & Register

Upon visiting this web application, consumers will be able to search for tradespeople without needing to register or login. Users can register an account as either a consumer or tradesperson, depending on their preference. However, registering an account will be needed if a consumer decides to favourite a tradesperson, or if a tradesperson decides to create a page.

Searching for a Tradesperson

A visiting consumer will have the ability to search for their required tradesperson. After they have selected their preferred tradesperson's occupation, consumers will then be able to filter their search based on their county and whether they own a website or not. As an extra piece of functionality, consumers will also be able to search for those tradespeople who have availed of the platform's premium service. The objective of this process is to be to be as user friendly as possible.

Favouriting a Tradesperson

A consumer will be provided with the capability to favourite their preferred tradespeople at any time. However, before favouriting a tradesperson, the consumer must login so that their selected tradespeople can be stored. After a tradesperson has been favourited, their information, as well as that of the current user will stored in the appropriate model. Following this, it will be displayed in the application's favourites page. The role of this favourites page is so that consumers can be connected to their desired tradespeople without the need to search again. This information will also be available for offline use so that regardless of the consumers situation, they will be provided with the means to contact a tradesperson.

Adding Tradespeople Information

A registering tradesperson will have the opportunity to add any relevant information so that it can be displayed to their page. This is achieved using a combination of two forms so that the tradesperson can add mainstream details such as their working hours while also enabling them to fill out a portfolio so that their previous work can be displayed.

Editing User Information

Users whom have logged into the application will be provided with the functionality to edit their account details using links found in the navigation bar. For consumers, this will be in the form of an edit details link while tradespeople will be provided with two links so that they can update their details and portfolio. When either user decides to edit these details, they will be shown their existing details so that they only have to change their desired fields.

1.4.2 Middle-Tier

The middle tier of this application will arguably be its most important and complex area of development as its core function is to connect the program's front-end to its back-end. This will be performed using AJAX to send various RESTful requests, with this being expanded on in its corresponding section.

Retrieving stored Tradespeople Information

'GET' AJAX requests will be implemented on numerous occasions so that the relevant tradespeople information can be queried from the application's back-end and returned to its corresponding template. This data will then be accessed in JavaScript so that only the relevant information will be displayed to the consumer.

Storing Favourited Tradespeople Information

Tradespeople Near Me will also execute 'POST' AJAX requests so that consumer's favourited tradespeople can be retrieved from the application's template and passed to its back-end. Once here, this information can then be stored in its relevant model. Unfavouriting a tradesperson will be executed in a similar manner with only fewer details needing to be passed. Executing these AJAX requests correctly will impact the overall efficiency of the application and so carry significant importance.

Service Worker

A service worker will be integrated into this project so that it operates as a progressive web app by still functioning once offline. This is achieved by caching all necessary files to the user's device when they first visit the application. Implementing this technology is one of the application's core functionalities as it enables it to operate just like a person's contacts directory on their phone.

1.4.3 Back-end

The back-end of this application will centre around the maintaining of database entries. It will involve handling the login, register, searching for tradespeople, favouriting a tradesperson and adding tradespeople information. It will also handle users editing their registered details.

Login & Register

Upon registering, both consumers and tradespeople will have to meet certain requirements so that their information can be verified before being stored in the appropriate model. Both users must enter a unique username, a password (with six characters or more containing a capital letter and a number), their first name, last name and a valid email address. They are also required to confirm their password and email so that these can be error checked in the application's back-end. A registering tradesperson will have the added check of their phone number being 7 or more numbers. Users will only be able to register once these requirements have been successfully error checked. Users logging in will also have their submitted details compared to that of the previously registered users. All these credential verifications will be made in the application's back-end and will be covered in a greater capacity in their subsequent section.

Searching for a Tradesperson

When a consumer searches for a tradesperson using the filters provided, the relevant query will be made to the application's database using the consumer's selected filters. This will see the search iterate over the application's stored tradesperson so that it returns the correct results. As there will be a substantial number of possible outcomes from this search, the application's back-end functionality will need to be carefully considered so that it is implemented in the most efficient manner.

Favouriting a Tradesperson

When a consumer attempts to favourite a tradesperson, there will be a verification process in place that ensures only those whom have registered can perform such action. This guarantees that only valid users will be able to persist changes within the application. Once a consumer has logged in and favourited their preferred tradesperson, the application will then add their chosen tradesperson to the relevant model within the database. Both the process of favouriting and unfavouriting a tradesperson will be executed on the

application's back-end so that the tradesperson and the current user's details can be stored or removed, depending on the preference of the consumer.

Adding Tradesperson Information

A registering tradesperson will be required to fill out a form on its template. This form will correspond to two separate forms on the application's back-end. The first of which relates to tradesperson details such as location and company information while the second relates to their portfolio. This portfolio allows each tradesperson to advertise their previous work in the form of a text editor and is beneficial in terms of the application's back-end as it enables it to be stored and retrieved as a single attribute. The more information a tradesperson adds to their page, the more chance they will have of being viewed by consumers.

Editing User Information

The current user's stored details are retrieved in the application's back-end so that they can be returned and displayed to the relevant template. This then enables the user to only change their desired fields, with these fields also being subjected to the same error checking found in the application's registration pages. Three separate back-end functionalities will be implemented so that both, consumers and tradespeople will be able to edit their details, while also allowing tradespeople to edit their portfolio.

1.4.4 Deployment

Tradespeople Near Me will be deployed so that it can be run on any environment regardless of the user's operating system. This will involve packaging the application into a single instance so that it can be deployed in conjunction with its chosen database. This project will be deployed along with a cloud provider so that the application can take advantage of its services in future work. An example of which being that it enables the application to be scaled on demand based on the network traffic it receives.

1.5. Thesis Roadmap

This section will provide an overview of what the following chapters are about.

2. Literature Review

The purpose of this project's Literature Review is to investigate existing solutions to the problem that this application attempts to overcome. It also explores how functions of these existing solutions can be incorporated into this application using the researched technologies.

3. Protype Design

This chapter outlines the development approach to be taken during the project. It accomplishes this by explaining the chosen methodology as well as the designing of various UML diagrams.

4. Prototype Development

This section provides a detailed overview of the project's development process, as well as different provided functionalities within these.

5. Testing and Evaluation

The aim of this chapter is to outline the application's testing and evaluation methods carried out throughout the course of the project.

6. Project Plan & Issues

Within this segment, the project's plan is reviewed, and the issues faced during its development are also discussed.

7. Conclusion and Future Work

This chapter outlines the conclusion of the project as well as the future work that will be carried out.

2. Literature Review

2.1. Introduction

The following section expands on the research carried out on other existing solutions to the problem that this application helps solve. It also discusses the possible technologies that will be used to implement the desired application.

2.2. Alternative Existing Solutions to Your Problem

This section will cover the key topics of research that were instrumental to this project. The studies carried out involve numerous existing applications that currently help tackle the problem that my project aims to overcome. In addition to this, web applications which produce a similar end product to that of the chosen project were studied. Individually, each of the researched applications offer partial functionality towards the project's end product of acting as an efficient medium between tradespeople and consumers. Three useful existing solutions that were evaluated include Tradesconnect, Daft and Airbnb.

2.2.1 Tradesconnect.ie

Tradesconnect.ie is a web application which enables the consumer to contact a tradesperson based on the company's location, reviews and photos. After the user fills out a form, detailing their job requirements, Tradesconnect "will automatically send it on to our relevant verified Tradesmen" (3). Although, this method of communication between the consumer and tradesperson is inefficient, it allows the user to get a quote from the tradesperson before deciding to proceed with the job. This product provides reviews, photos and a rough graphical location of the tradesperson's company to user. However, it fails to incorporate tradespeople whom are not associated to any particular company. This application works well for people who do not require an urgent job to be carried out. The form and geolocation functionality implemented in Tradesconnect can be seen in figure 1.

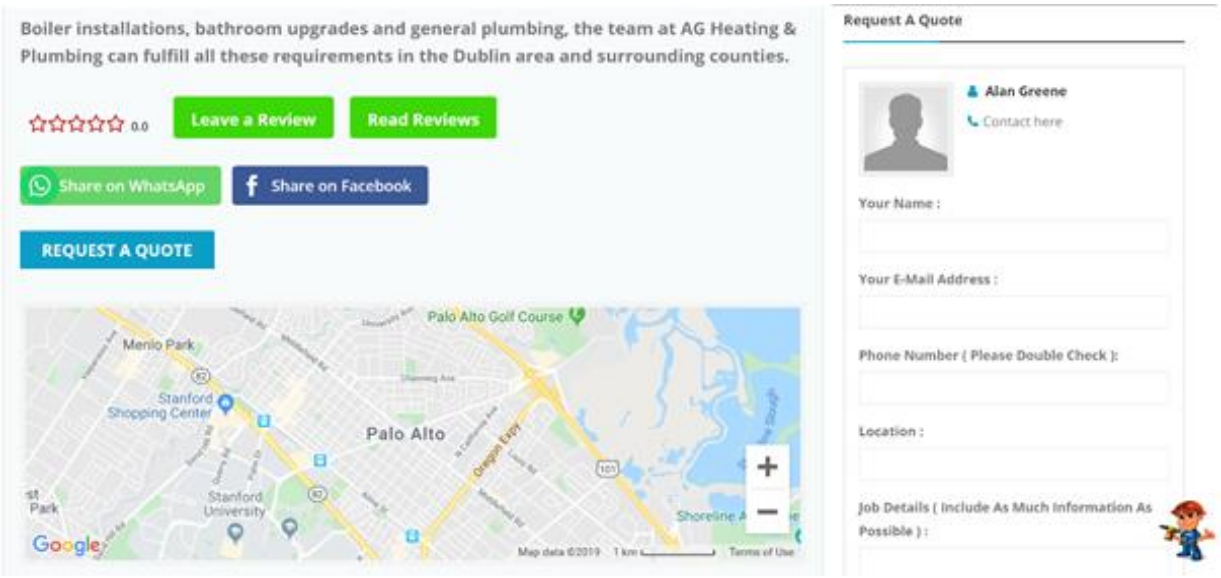


Figure 1: Screenshot of TradesConnect (4)

2.2.2 Daft.ie

Daft is web application, founded in 1997 and is aimed at connecting professionals and property owners. Daft has risen substantially since 1997, boasting an "audience of over 2.5 million users each month" (5). It is a complex app, combining both, geo-boundaries of properties and a recommendation sorting algorithm to display a user interface. Tradespeople Near Me aims to incorporate a similar process to display the tradespeople near a location, based on the consumers inputted filters. A sample of Daft's returned results and filtering system can be seen in the figure below as well as figure 78 in the appendix.

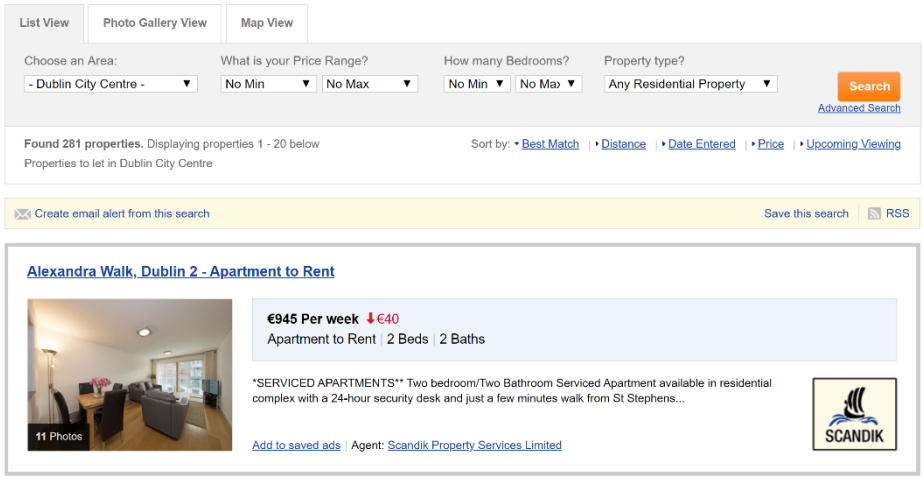


Figure 2: Screenshot of Daft Search Results (6)

2.2.3 Airbnb

Airbnb is an online marketplace, connecting users and homeowners across the world. Airbnb acts as a medium for arranging accommodation and tourism experiences. “It provides access to 7 million unique places to stay in more than 100,000 cities and 191 countries and regions” (7). Similar to daft, Airbnb allows the user to simultaneously view the details and the location of the property through a graphical interface. As well as this, they implement a recommendation search algorithm to return properties best suited to each individual user. Upon selecting a property to view, the user is provided with an area as to where the property is based. It is this location-based user interface shown in figure 3 that Tradespeople Near Me aims to achieve.

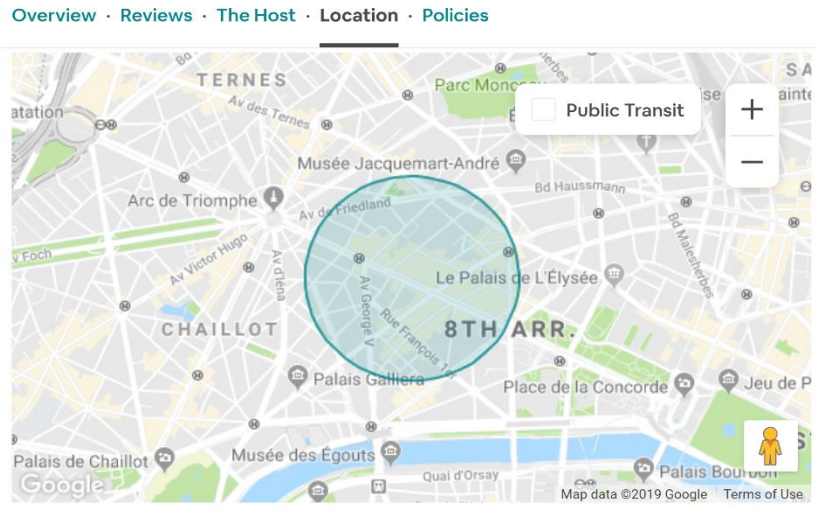


Figure 3: Screenshot of Airbnb Property Location (7)

2.2.4 Conclusive Evaluation

From the analysis carried out on existing tradespeople applications, it is clear to see that there are few services offered to consumers whom require a tradesperson. The majority of these applications prioritise companies and often fail to provide a platform for the self-employed. Therefore, those who are self-employed must rely on their previous clients to publicize their services to others. Not only this, but there are very few company details displayed which could influence a consumer’s decision to contact them. Based on the reasons discussed, it can be concluded that there is a market space for a more inclusive tradesperson application.

2.3. Technology Review

There are a multitude of technologies to implement a progressive web application. Below are a sample of the technologies researched and selected for this project.

2.3.1 Programming Languages

There are numerous programming languages ideal for web application development. For this project, PHP, Node.js and Python were researched, each of which make valid arguments to implement Tradespeople Near Me. The subsequent passages will discuss these three languages in depth.

PHP is a server-side programming language, used to develop web applications. PHP provides the foundations to enable a user to build a high-performance reliable website while maintaining relatively low development costs. PHP is very fast loading language which is compatible on various platforms. PHP is one of the most popular programming languages among website development, but ultimately was not chosen as this application's programming language.

Node.js is a runtime environment, written in JavaScript. It was developed in 2009 and has grown in popularity year on year. Node.js offers a higher performance than most other programming languages as it executes its code using Google's V8 JavaScript engine. Node.js allows the user to access "a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent" (8). Another key advantage of using Node.js is that it can be implemented for both, the front-end and back-end of the project, saving valuable time needed to learn another language as a result. However, a major flaw of using Node.js is that it's API is constantly changing which in turn creates huge problems for developers as the changes are not backward-compatible. Despite the initial time management benefits, Node.js will lose developers time when the API changes. As time management plays such a pivotal role in this project, Node.js was dismissed as a potential technology to implement.

One of the main factors considered when choosing a programming language is the user's prior experience in each language. Having previous experience in Python played a central role in the selecting of it as this project's programming language. Python has been one of the most popular programming languages since its first version was published in 1991. It's consistent popularity over the last decades is attributed in many ways to its extensive features made available to the user. In addition to supplying a large standard library to the user, "the Python Package Index (PyPI) contains numerous third-party modules that make Python capable of interacting with most of the other languages and platforms" (9). The vast advantages affiliated with Python make it the ideal programming language for this project.

2.3.2 Client-Side Scripting

Vanilla JS "is the most lightweight framework available" (9), and involves the use of JavaScript without any additional libraries made available by JavaScript. These libraries include jQuery which is aimed at making JavaScript more efficient. Vanilla JS is substantially faster than any other scripting framework, allowing the user's browser to load Vanilla JS into memory before it even requests the website. Companies who incorporate Vanilla JS into their technologies comprise of Facebook, Google and Netflix.

A scripting framework however is a platform to aid in the development of software applications. It provides the structure on which software developers can build their application on. This pre-defined structure enables developers to streamline the development cycle of a project. "You can expect to build a project in much less time than would be achieved writing code without a framework" (10). There are endless advantages to implementing a scripting framework but the role it plays in integrating the front-end of an application to its back-end, cannot be overlooked. Any large web application relies on an efficient connection between data typically stored in a database, and the user's interface. It is this efficiency and reliability which makes a scripting language essential for this project. After deciding upon the use of a scripting framework for this project, the next logical step was to choose an architectural pattern in which to base the framework of.

2.3.3 Architectural Patterns for Web Applications

MVP, MVVM and MVC are a sample of the most used architectural patterns in application development. They are all patterns which provide a “reusable solution to a commonly occurring problem in software architecture within a given context” (11). The following section will cover the benefits of each pattern and which best satisfied the demands of this project.

The Model-View-Controller (MVC) pattern “divides an application into three major aspects: Model, View and Controller” (12), with each component, being built to handle specific development functionality. The Model represents the user interface, consisting of a collection of classes which display the view. Its controller behaves like an interface which allows the Model to interact with the View component. The controller is an essential part of the MVC pattern as it handles the incoming of requests and the manipulation of data in order to display the View. Implementing these components as described enables the MVC architectural pattern to follow both, the single-responsibility and the separation of concerns design principles. While the single-responsibility design principle involves modularizing code into their separate applications, the separation of concerns design principle involves “separating an application into distinct sections, so each section addresses a separate concern” (13). Implementing the MVC pattern therefore offers a faster development process as well as faster debugging times. Combining this with the fact that the Model limits the need for formatting data, makes it very hard to look past MVC as this project’s architectural pattern.

Model-View-ViewModel (MVVM) is a software architectural pattern used to implement bi-directional data within views. This enables changes of properties in the View-Model to be automatic synced. This is executed through the use of binding data between View and View-Model. “It is responsible for exposing methods, commands, and other properties that help to maintain the state of the view, manipulate the model as the result of actions on the view, and trigger events in the view itself“ (12). MVVM is focused towards separating the user

interface part of the application from the related code as well as allowing different layers to be reused. It also provides easier testing through separating the user interface and application logic. These attractive benefits make the MVVM an appealing choice as this project's architectural pattern.

The Model-View-Presenter (MVP) pattern lets you "decouple business logic (Model) from view logic (Activity/Fragment) by introducing an intermediator called as Presenter" (14). It is derived from the foundation of the MVC pattern and serves to divide the patterns responsibilities over three components. The view oversees the displaying of the user interface, the presenter is in charge of communicating between the view and the model, while the model is responsible for the handling, changing and manipulating of the business logic. With MVP, the view and presenter communicate with each other via an interface, making the implementation of unit testing easier.

After much deliberation, MVC was decided upon as the chosen architectural pattern for this project as it is an ideal pattern to implement a large scalable application. Furthermore, its easy separation of business logic promotes clean, maintainable code. These two benefits to using the MVC pattern make it the perfect architecture on which to implement Tradespeople Near Me.

2.3.4 Framework

For this application, frameworks such as Express, Laravel and Django were studied, each being viable options on their own accord.

Laravel is an open sourced PHP framework designed to make web application development more efficient. This is achieved by containing many built-in libraries. A secure authorization system is integral to any modern web application and as Laravel "provides a simple way to organize authorization logic and control access to resources" (15), it presents an appealing case as the framework to be implemented in this project. On top of this, Laravel also

provides pre-configured error handling for any new project. Therefore, Laravel makes a compelling argument to be the framework of choice for this project.

Express is a Node.js web application framework aimed at minimalizing the code needed to execute tasks. It enables the user to shorten the development time of a project in several ways including that Node.js can be implemented for both, the front-end and back-end of a web application. Express' benefits can be utilized in applications requiring a great amount of user input handling, and for this reason, companies such as Uber chose Node.js for their runtime environment. As time management is such a crucial aspect of this project, it is hard to look past the many benefits associated with the implementation of the Express framework.

Django was ultimately chosen as the framework to deploy this application. This choice was based on the multitude of advantages associated with it. Django is written in Python and "provides code for common operations like database manipulation, HTML templating, URL routing, session management, and security" (16), thus improving development time. Django is one of the most compatible frameworks available, allowing users to develop using most popular operating systems such as Windows, MacOS and Linux. In addition to this, Django contains built-in security features to help protect against unwanted security attacks. Django adheres to the model-view-controller (MVC) design, often referred to as MVT as its main components are the layers, model, view and template. Having a detailed structure for a framework, allows the developer to easily separate the different components of an application. GeoDjango is a Django application, essential for developing a Geographic Information System (GIS), and it was this additional functionality which made Django the best suited framework for this project.

2.3.5 Client-Side Language

A client-side programming language refers to the language implemented on the client's browser to make up the front-end of a web application. There is a huge selection of client-

side languages to pick from but for this section, AngularJS, React and JavaScript will be considered.

AngularJS is a client-side language which has grown significantly in popularity over the last number of years. “AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop” (17). It also enables the user to create a clean and maintainable single page application using reusable components and unit testable code. Angular is compatible with all popular browsers, making it a safe choice for a client-side language. However, Angular would fail to satisfy the requirements of this project as it not as secure as its competitors and is predominately used for the development of large single page applications.

React is a flexible JavaScript library implemented in order to speed up the development cycle of a user interface. “It lets you compose complex UIs from small and isolated pieces of code called ‘components’ “ (18). React offers the user a chance to design simple views for an application which it will in turn update and render the appropriate components when the data changes. The continuous development of ReactJS results in a highly efficient language but one in which the speed of development forces developers to continuously relearn new methods of implementing things. Combined with poor documentation for the rapidly growing technologies, React was not chosen as a client-side language.

JavaScript is one of the most popular client-side scripts as “nearly every site’s front end is a combination of JavaScript and HTML and CSS” (19). The advantages of implementing JavaScript are endless, with it offering a fast scripting language which can be run immediately within the client’s browser. Furthermore, its simple syntax is easy to learn and implement. These factors have seen it become more popular in the industry in previous years. JavaScript allows for the incorporating of additional libraries such as jQuery and RxJS, so that more efficient code can be developed. jQuery is designed to simplify event handling as well as allow for HTML DOM tree manipulation, while RxJS “is a library for reactive

programming using observables that makes it easier to compose asynchronous or callback-based code" (20). Both, jQuery and RxJS are lightweight libraries which enable the developer to condense many lines of JavaScript code into methods that can be called using a single line of code. Having previous experience in JavaScript, in addition to the benefits mentioned, made it the only logical option as this project's client-side programming language.

2.3.6 Data Persistence

With the idea of implementing a progressive web app in mind, choosing a reliable database is extremely important in the running and maintaining of the system. There is a vast array of database technology to choose from when implementing a web application, a sample of which being Oracle, MySQL, Firebase and PostgreSQL.

Oracle Database is a collection of data, used to store, access and retrieve information through the use of a database server. "In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data" (21). Despite the numerous advantages associated with the use of Oracle, it was not selected as this project's database. This was due to two factors, one being the cost of running Oracle and the other being the complexity that it brings with it. Oracle was the most expensive data persistence options researched and therefore was not best suited to this particular project.

Another database studied before the implementation of this project was MariaDB. "MariaDB was initially developed in direct response to Oracle acquiring MySQL. MariaDB offers you an opportunity to "break free from the costs, constraints and complexity of proprietary databases and reinvest in what matters most, developing innovative applications and services as fast as possible" (16). MariaDB offers frequent security releases as well as being open sourced and easily compatible with MySQL. MariaDB is a solid choice

to use for this project but it ultimately was not chosen because of its difficulties with fast reliable caching.

After extensive research, PostgreSQL was decided upon as this projects database. PostgreSQL supports several programming languages such as Python, C, Java and Perl and as well as this, it "supports different kinds of techniques for geographical data storage such as PostGIS, Key-Value Store and DBLink." (22). PostgreSQL is a free, open-sourced database which enables you to use, manipulate and implement it as you see fit. As well as this, PostgreSQL supports the storage of both, geographical and JSON objects, making it ideal for a location-based web application. In addition to this, PostgreSQL provides a geographical extension called 'GIS' which supplies built-in functions to handle geographic data in different formats. Tradespeople Near Me will be implemented using both, the programming language, Python, and the geographical data storage, PostGIS. Therefore, choosing PostgreSQL became an obvious choice.

2.3.7 Cloud Computing Services

Before deploying this project, a cloud provider must be chosen on which to host the application. The researched cloud computing services include Amazon Web Services (AWS), Google Cloud Platform and Digital Ocean.

"Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform" (23). AWS Elastic Beanstalk is a service offered by AWS, providing a platform for deploying and scaling web applications. The documentation for the implementation of an AWS service is very clear and provides detailed tutorials to get started. There are many advantages associated with the implementation of AWS services which include having no commitment to any contract, having built-in security infrastructure and allowing your web app to be completely flexible. There are few issues associated with the use of AWS, one being that service limits are set by the platform so that it can prevent uncontrolled resource usage. Despite its many benefits, AWS was eventually dismissed as a

potential cloud provider for this application due to the investment in time needed to fully understand and implement the extensive features provided.

Google Cloud Platform includes a range of cloud computing services which provide “a web-based, graphical user interface that you can use to manage your GCP projects and resources” (24). As well as supplying useful documentation, Google provides the user with the same security model for their application as they have currently securing YouTube and several other products. These extensive features don’t come cheap however, with fees for both, support and downloading data, thus making it unsuitable for this project.

Digital Ocean was this application’s eventual choice as a cloud platform. Digital Ocean provides many advantages such as daily data backups and end-to-end security. “Managed Databases run on enterprise-class hardware and SSD storage, giving you lightning-fast performance” (25). The only compromise with Digital Ocean is that the developer must source their own SSL certificate and domain name, but these can be obtained using other methods. Digital Ocean also provides students with $50 of free credit so that they can start deploying their application as soon as possible. These outlined benefits make Digital Ocean an smart choice as this application’s cloud provider.

2.3.8 Version Control

Version control refers to the management of changes to any files associated with a project. Tools such as CVS, SVN and Git were considered for this project, with Git ultimately being chosen.

Thanks to its extensive list of features, Git has become one of the most popular version control tools used by developers today. Git is an open-source software which provides developers with a local copy of the entire project’s development history. Thus, allowing them to easily track and maintain changes to the code. The importance of incorporating source management into a project cannot simply be overlooked as it’s “ultimate aim is to

maximize your resources' efficiency" (26). Source management also provides developers with peace of mind to make changes to their application, knowing that they can always revert to a previous version.

Therefore, the choice of source management is critical to ensuring the success or failure of a project. For these reasons, Git was integrated as part of this project as soon as its development began.

2.4. Geo-fencing Research

The complex features within this application arise from its filtering-based search algorithm and through its use of geo-fencing. In addition to this, the project will be developed as a progressive web app. To ensure that these features are integrated seamlessly into the web app, the following research was carried out.

A filtering tool in any given website has traditionally been a left-hand vertical sidebar. However, over the last number of years, a horizontal sorting toolbar has slowly grown in popularity. This is because a horizontal filtering toolbar can limit two issues frequently caused by a traditional sidebar. One being that users fail to notice the filtering sidebar and the other being that users can misinterrupt the site's sorting tool as the site's filtering toolbar. "Both issues are severe as it prevents users from getting a well-defined product list which match their purchasing criteria - and instead leaves them navigating overly broad product lists" (27). This would then cause a tired user who has been searching an unnecessary broad list of results, to navigate to a competitor site. This being the very thing that a clean and efficient user interface is designed to prevent. "Despite the initial plaudits for designing a horizontal sorting toolbar, it does have its drawbacks. The main drawback being that it can quickly become cluttered if there are too many filtering options. A horizontal toolbar is the optimum choice for a filtering tool when the application has a small number of filters and that the number of these filters will not change as the complexity of

the application grows. As this application requires only four filtering options, it is within its best interest to implement a horizontal toolbar used for both, filtering, and sorting.

Many types of websites share searching functionality, with each being tailored to that specific website. This searching functionality is referred to as a 'full text search' and involves storing data as an array of objects in a JSON file which can be iterated over to find the appropriate matches. As the need for an efficient, accurate full text search grows, so does the available options for implementing it. There are now many ways to carry out a search, with this section concentrating on the use of a back-end search engine, commercial search engine and a database with a built-in search engine.

Benefits of using a back-end search engine such as Elasticsearch include that it can handle queries very fast thanks to its process of data ingestion, in which "raw data is parsed, normalized, and enriched before it is indexed in Elasticsearch" (28). In addition to this, Elasticsearch can be easily scaled, allowing users to expand resources. In spite of these initial advantages, back-end search engines were deemed unsuitable for this project due the expensive cost of setup and maintenance.

Another possible way to perform a full text search is by using a commercial engine such as Alogia or Amazon Cloud Search. Commercial engines can simplify the management process of setup, maintenance and updates, with Amazon Cloud Search supporting "features such as highlighting, autocomplete, and geospatial search (29). Although Amazon offer an extensive list of features, these features can result in additional costs, giving it the potential to become expensive. As well as the financial drawbacks, Cloud Search's server-side relies on connectivity and battery usage to operate. Despite this, commercial engines such as Amazon's Cloud Search present a solid case as this projects method of implementing a full text search.

The chosen method to implement this project's searching functionality was to select a database with a built-in search. Databases which support full text search comprise of MongoDB, CouchDB, MySQL and PostgreSQL. Selecting a database with full text search saves a substantial amount of time and money incorporated in the setting up and maintaining of a separate search engine. Thus, shortening the developing time needed to execute a search.

After a user searches for tradespeople near them, they will be greeted with a graphical representation of available tradespeople in their area. This graphical representation will be performed using geo-fencing, which is a "location-based service software that uses GPS, Wi-Fi, Cellular Data, RFID (Radio Frequency Identification) to trigger a precoded action when an RFID tag or mobile device enters the geofence or the virtual boundary that is set up around a geographical location" (30). The software implemented to measure the user's location depends on the positioning of the user. If the user is outdoors, GPS will be used where available, while indoors will involve the use of cellular and Wi-Fi data. The Global Positioning System (GPS) is a network of satellites orbiting Earth whereby the three nearest satellites to the requested GPS position send messages to each other in order to calculate exactly how far each satellite is away so that they can isolate the needed location.

In previous years, the accuracy of GPS has been continuously improved. This can be seen with the EU project, Galileo, "which is only partially complete is intended to be more advanced than any other system in the sky at the moment with position measurements within 1-metre precision and better positioning services at higher latitudes than other positioning systems" (31). Despite this promising information, Galileo's new system is not scheduled for completion until 2020. In addition to Galileo, GPS chip manufacture, Broadcom, are developing a new chip for smart phones which will see it get a location within an accuracy of 30 centimeters using the existing US GPS network. Despite the development of this instrumental GPS technology, it will not become commercially available until 2022.

GPS is the most accurate method of retrieving a location, however, for indoor environments, buildings and walls can make for an inaccurate reading. As a result, geofencing uses a combination of cellular and Wi-Fi data for indoor use. The accuracy of the retrieved user location therefore depends on the type of environment they are in, with urban environments reaching a greater accuracy than that of rural environments. Despite this, the recorded location using this method is “typically within 5-15 metres” (32).

This application will implement the HTML5 ‘geolocation.getCurrentLocation()’ method so that it can retrieve the current geographic location of the device being used. This information will then be “expressed as a set of geographic coordinates together with information about heading and speed” (33). Executing this method will provide the user with the permissions alert shown below and will subsequently enable the application to be executed depending on the permissions granted.

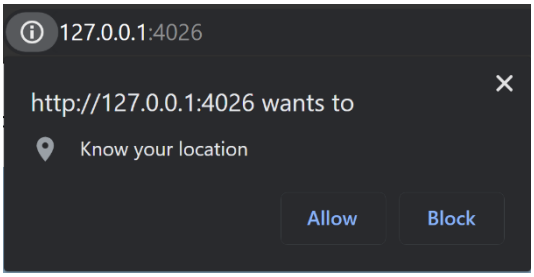


Figure 4: User’s Location Permissions

Integrating HTML5’s Geolocation into this project enables the web browser to locate the user’s location using a combination of GPS, Wi-Fi positional system, mobile network location and IP Geolocation. Should the browser find it difficult to measure the user’s location using the first two previously defined methods, it may opt to implement mobile phone tracking. This method becomes necessary for devices which do not have a GPS chip or access to Wi-Fi, but instead have access to a cell network connection. Examples of which include laptops and iPads. Similar to GPS, network location is measured using cell tower triangulation of three or more towers so that it is “accurate up to 30 metres on average” (34). How the

three cell towers intersect so that the user's position can be measured is shown by the following figure.

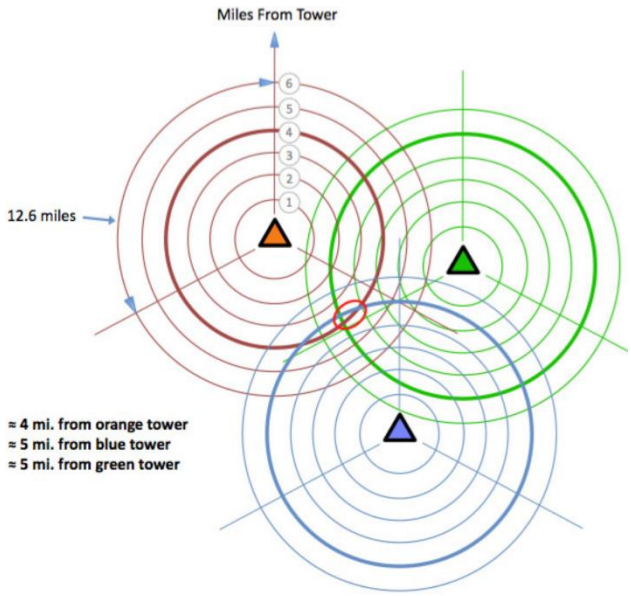


Figure 5: Mobile Network Location

Geolocation's strength is that it caters to all environments. If the three previous methods are not available to the browser, it will implement IP geolocation so that it can receive the IP address of the user's address at the very least. This method does however compromise on the accurate of the location, with its accuracy varying significantly depending on the city the user is located in. Consequently, it is used as a last resort in devices with only a wired connection to the internet. Implementing HTML5's geolocation into this application enables it to be catered to all users regardless of their tracking technology and therefore adheres to the overall goal of this project.

When a tradesperson registers with Tradespeople Near Me, they are agreeing to having their location stored in the application's database. This will be accomplished by obtaining their latitude and longitude coordinates from the method mentioned above and will be then stored following the completion of their registration form. When a consumer then visits the application's home page, each tradesperson details will then be read in from the database so that their specific location can be marked on the displayed map. After this, the distance

in which they are willing to travel to will be read in so that each appropriate circle diameter can be displayed around the previously created marker. Thus, providing the consumer with a visual representation of the area in which each tradesperson is available in.

This application will ask the consumer whether it can use their location, with the following algorithm only being implemented when this permission has been granted. After the consumer has allowed this permission, Tradespeople Near Me will then implement an algorithm to check if the user's location falls into the specified maximum distance each tradesperson is willing to travel to, and based on these results, the application will display the relevant tradespeople to the consumer. This will be accomplished by following a defined number of steps within a selected geo-fencing formula.

The formula that will be implemented in this project will be the Haversine distance formula. "The Haversine Formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface" (35). This formula will involve using the latitude and longitude in radians of two GPS coordinates, their absolute differences and the angle between them. This formula can be expressed in mathematical expression shown in figure 6, where "*d* is the distance between two points with longitude and latitude (ψ,ϕ) and *r* is the radius of the Earth" (36). It is also explained in greater depth in the article, 'The Cosine-Haversine Formula' by C. C. Robusto (37).

$$d = 2r\sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right)+\cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\psi_2-\psi_1}{2}\right)}\right)$$

Figure 6: Haversine Formula (36)

Although the distance between two coordinates could be retrieved using the distance formula found in co-ordinate geometry, it would fail to account for the fact that the earth is a spherical object and not all lines measured are straight. This being said, it is unlikely that a

consumer and tradesperson will be separated by a distance that will see the shape of the earth make a substantial difference in the outcome of the algorithm. Nonetheless, the Haversine formula shown above was implemented into this project so that the most accurate distance between consumer and tradesperson could be obtained. As the Haversine formula accounts for the Earth being a sphere instead of its mathematical term, oblate ellipsoid, it "can result in an error of up to 0.5%" (38). This means that the distance calculated between two coordinates on opposite ends of the earth will have a maximum distortion of 0.5%, with this figure significantly decreasing the closer these coordinates get to one another.

After the consumers location has been obtained, the application's next step is to read in each stored tradesperson's location so that it can calculate the distance between them and the consumer using the Haversine formula shown above. This calculation is then compared with that of each tradesperson's pre-determined maximum travel distance. Once compared, each tradesperson whom is within the identified diameter will be displayed to the consumer. This is illustrated using the figure shown below.

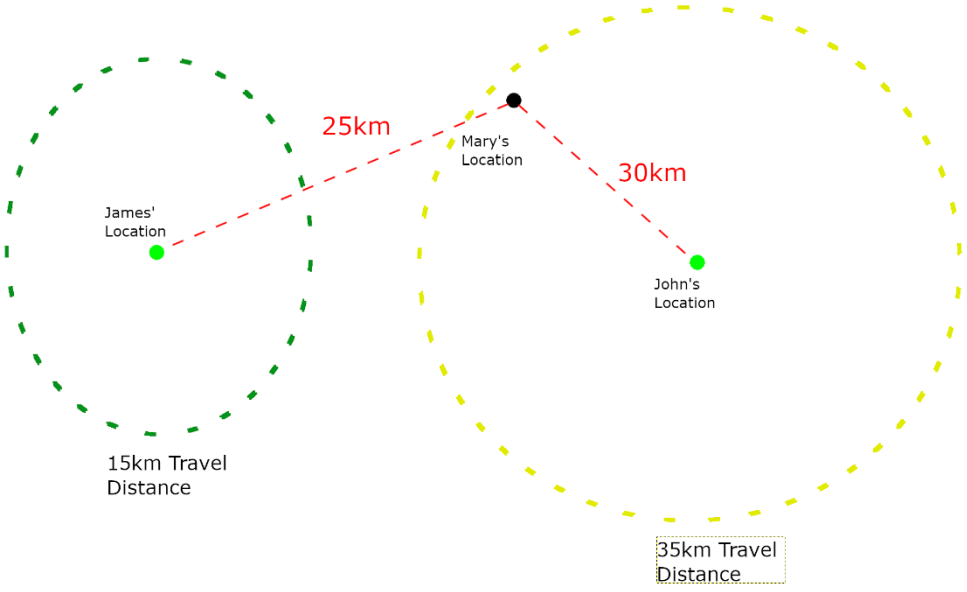


Figure 7: Geo-fencing Algorithm

The geo-fencing algorithm implemented in this application can be explained through the diagram illustrated above. In this example, Mary is the consumer searching for a

tradesperson in her area while James and John are both tradespeople. Here, it can be seen how the distance between Mary and John, is first measured, then compared against the distance that John is willing to travel to. In this case, the consumer's location lies within John's specified maximum distance of 35km but fails to lie within James' desired distance of 15km. Subsequently, John would be the only tradesperson displayed to the user.

In order to present a clear, user friendly interface, this project will incorporate the use of Leaflet and Bootstrap. " Leaflet is the leading open source JavaScript library for mobile-friendly interactive maps" (39), allowing for the creation of a map and location markers. Despite only weighing around 38KB, Leaflet provides an extensive array of mapping features, making it the ideal mapping technology to implement Tradespeople Near Me's intricate geo-fencing. Bootstrap will also be deployed along with it, in order to help adapt the application for mobile use. Bootstrap enables the fast development of a project as well as being "equipped with responsive layout and 12-column grid system that help dynamically adjust the website to a suitable screen resolution" (40). It is extremely important that Tradespeople Near Me is in fact responsive so that it allows for the widest user base possible.

When a user visits Tradespeople Near Me, they will have its relevant files cached to their device so that the user can download it to their home screen via the browser's inspect tab. Thus, giving this web application the appearance of a native app, traditionally downloaded through an app store. To give the overall look and feel of a native app, this application will be developed as a progressive web application (PWA), providing users with offline functionality. To allow for this, a service worker will be integrated into the application, which enables control over network requests. A service worker lets the developer control what data to cache when online and provides the user access to this same content once offline. In this project's case, the users favoured tradespeople's information will be the cache data. Therefore, permitting users to access this information regardless of their network status. The popularity of progressive web applications has steadily grown over the

last several years and despite being in their infancy “they have the potential to create a shift in the way the web works” (41).

Each of the technologies described above, are individually comprehensive components to implement, but the real challenge will see each technology being incorporated together into one user-friendly, responsive progressive web application.

2.5. Existing Final Year Projects

2.5.2 Project 1: Crime Explorer

Student: Deividas Savickas

Description (brief):

Crime Explorer is a web application which aims to allow users to view crimes instantly. The user will be able to filter the crimes based on a region, time and crime type. This returned information is then displayed using a geographical visualization. Members of the public will be able to provide information on ongoing investigations and it will cater to all device’s sizes.

What is complex in this project:

Accessing the available crime statistics and displaying them to the user of the web application was difficult but the hardest part of this project was the filtering of the desired crime data by a user.

What technical architecture was used:

MongoDB, Mapbox, Leaflet, Chart.js, Bootstrap, Node.js

Explain key strengths and weaknesses of this project, as you see it.

The application provided a clear simple user interface which displayed precise results upon request. A weakness of the project could be that there is no administrator access to filter out unhelpful information sent in by the public.

2.5.1 Project 2: FindMyDoggo

Student: Ciarán O'Brien

Description (brief):

FindMyDoggo is a web application which incorporates machine learning. It was built using python and the framework Flask. It serves as an efficient method for the public to locate their missing dogs. This web app incorporates machine learning and image processing techniques to identify and classify features about a dog.

What is complex in this project:

The hardest part of this project to implement was enabling the machine learning to register, classify and gather abstract features about a dog. Incorporating image processing techniques into this web app can also be seen as complex.

What technical architecture was used:

Digital Ocean, Flask, Tensorflow functions, Image Processing, Python, Keras APIs, Neural Networks

Explain key strengths and weaknesses of this project, as you see it:

Hosting the web application using Digital Ocean was a great decision as it is easier to maintain and scale in the future. A weakness of the project could be that machine learning is very difficult to implement and obtain precise results using image processing.

2.6. Conclusions

The research discussed in this section allows the developer of the project to understand exactly what is required to build this application. This ultimately enhances the quality of the prototype designs as depicted in the following section.

3. Prototype Design

3.1 Introduction

The following chapter provides detailed research into the software methodologies implemented within this application. It will also cover the early designing stages of the project along with high fidelity prototypes of the user interfaces.

3.2. Design Methodology

A design methodology outlines the framework implemented for the development lifecycle of a particular system. Implementing a design methodology ensures that the developer or team follow a set defined structure to provide an efficient project delivery. Methodologies are more advantageous for large scale development but also can be used in any project, large or small to maintain quality throughout the development process. To ensure that chosen methodology was best suited to this project, research on existing methodologies was carried out on Agile, Scrum and Kanban.

3.2.1 Agile

"Agile methodology is a practice that helps continuous iteration of development and testing in the SDLC process" (42). Agile enables developers to split the development stages of a project into smaller builds called sprints so that the development is concurrent. Sprints are short time periods whereby the development team work together to complete a defined amount of work. These short sprint cycles allow developers to continuously refine the project while maintaining quality. For Agile, customer engagement is a high priority with emphasis on continuous delivery between the development team and the customer. This continuous customer development helps in the development of a high-quality product which adheres directly to the customers' vision. Within Agile, there are various forms of methodologies that can be applied to a project, two of which are Scrum and Kanban.

3.2.2 Scrum

"Scrum is an agile project management methodology or framework used primarily for software development projects with the goal of delivering new software capability every 2-4 weeks" (43). Scrum is the most popular agile methodology implemented as "70% of software teams use Scrum or a Scrum hybrid" (43). With Scrum, the features of a given project are broken down into sprints lasting for one to four weeks. At the beginning and end of these sprints, meetings are held with the scrum master and development team, during which tasks are first assessed from the previous week, before being assigned for the coming week. During these meetings, any issues obtained during the sprint are brought to the team's attention. The sprints carried out during a project help ensure that customer deadlines are met. The timeboxed iterations associated with Scrum improve the overall quality of the project, with the continuous iterations of the project improving debugging times. Despite the numerous plaudits for the use of Scrum as a software methodology, the reason it was not chosen as this applications methodology was because it works best within a small team environment consisting of at least three people. However, this application is to be developed individually and therefore is not best suited to applying Scrum methodology.

3.2.3 Kanban

"Kanban is a method for managing the creation of products with an emphasis on continual delivery while not overburdening the development team" (44). In a Kanban process, there are cards that move through the process depending on the stage of development. These cards are assigned to a Kanban Board, moving from start to finish as the task it refers to gets implemented. An example of a Kanban Board can be seen in the figure below.

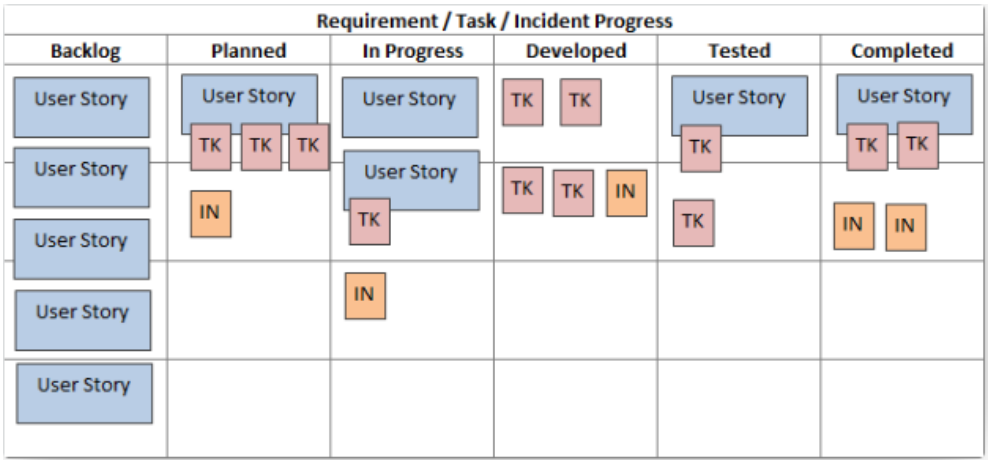


Figure 8: Kanban Board Life Cycle (45)

Here, it is clear to see the development stages of the various tasks within a project. Once assigned, each task makes it way from the 'Planned' stage, to the 'Completed' stage of the Kanban Board. The speed at which each task moves depends on how complex it is, with some tasks requiring more time than others.

Kanban differs to Scrum in that Kanban is event-driven, with timeboxed iterations being optional. Event-driven tasks are ideal for the deployment of this application as sometimes it is extremely hard to estimate the amount of time needed to carry out a task. Thus, making Kanban a perfect methodology to implement as part of this project. Kanban allows users to add new tasks once the capacity is available. In contrast to Scrum, a Kanban Board can be implemented as part of a team or individually. Kanban's ability to continuously add new tasks is ideal for this project as its functionalities will continue to grow.

3.3. Overview of System

The application to be deployed will be a progressive web application (PWA) and will follow the system architecture shown in the figure below and in the appendix at figure 79.

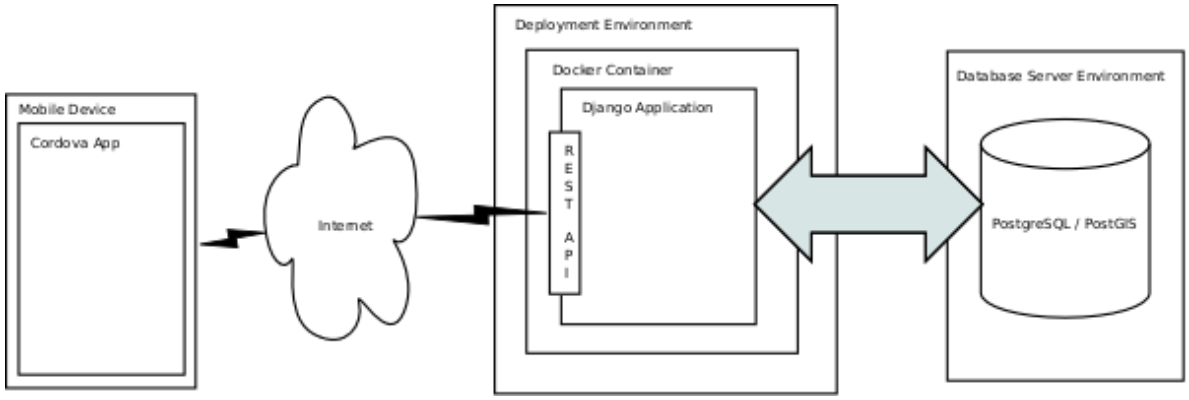


Figure 9: Architecture Diagram for Tradespeople Near Me

The front-end of this web application will take the form of a PWA and will be implemented using Python, HTML, CSS and JavaScript. These technologies will be accompanied with additional JavaScript libraries such as jQuery and Bootstrap. Therefore, shortening the development life cycle of the application. Incorporating Bootstrap into this project also enables Tradespeople Near Me to be scaled according to each user's device. Consumers visiting this application will have an option to favourite their desired tradespeople so that they can be easily accessed for future use. This, combined with offline functionality, will present a very practical application, equipped for daily use. Its front-end will be developed using the framework, Django, and will subsequently follow the Model-View-Template (MVT) design pattern.

The front-end of this application will interact with its back-end from using REST API. REST stands for Representational State Transfer which "means when a RESTful API is called, the server will transfer to the client a representation of the state of the requested resource" (46). REST API follows a request from the user to query data and returns the state of that query. The representation of the state in this application will be in a JSON format. The request the user selects will take the form of a HTML verb, with the most common being GET, POST, PUT and DELETE. These will then be passed to the back-end of the application so that their relevant tasks can be preformed.

This project's back-end will consist of PostgreSQL as the database, Digital Ocean as the cloud provider and Docker as the container software. PostgreSQL will communicate to the front-end using AJAX to send RESTful requests. Thus, allowing the application to interrupt user requests so that its database can be searched for any relevant results. Docker will be deployed so that all dependencies of this project can be packaged together in a docker container. This combined with a Digital Ocean droplet will enable the application to work seamlessly in any environment

3.4. Front-End

3.4.1 Activity Diagram

An activity diagram is an extremely useful tool in UML as it helps describe the operation and flow of one activity to another within a system. The activity diagram outlined below illustrates the flow of events in this application following a search from a consumer.

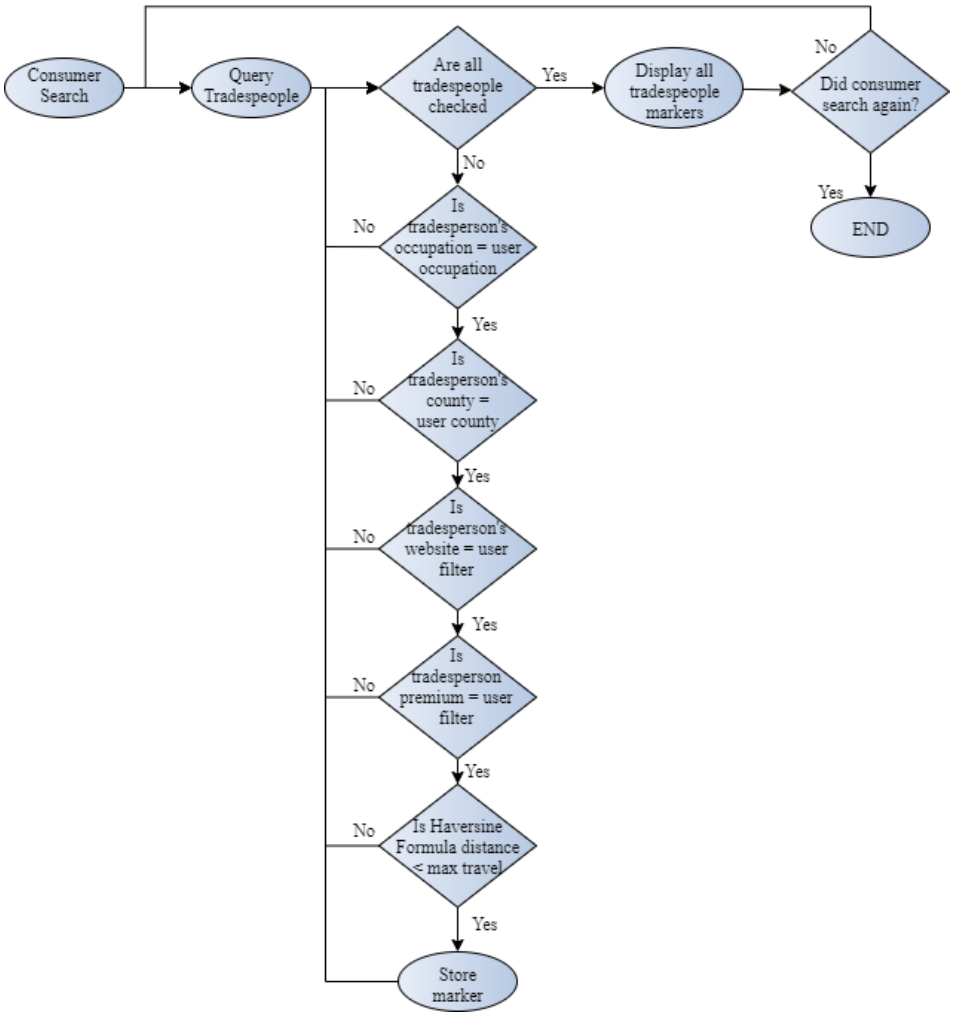


Figure 10: Activity Diagram for Consumer Searching for Tradespeople

The events depicted in figure 10 represent the flow from one activity to another following a search from a consumer. After the consumer has searched for their desired tradesperson, the application will retrieve the relevant data using the database queries until all possible tradespeople have been checked. However, before doing so, each tradesperson is then iterated over to check whether they adhere to the consumer's filtered search. It accomplishes this by checking if the tradesperson's occupation, county, website and premium attributes match that of the consumer's selected choice. After each relevant tradesperson has been returned, they are then passed into the specified Haversine distance algorithm so that the distance from the consumer to each tradesperson can be compared with that of each tradesperson's maximum travel distance. If the consumer falls within the tradesperson's desired distance, it then stores its location as a marker. This procedure is then repeated for each tradesperson until complete. When the application has finally

iterated over every possible tradesperson, it will display all the stored markers to the consumer in the form of a map.

3.4.2 Use Case Diagram

The use case diagram shown below details the different interactions consumers and tradespeople can have with the application's functionalities. It also illustrates how the admin reserves the right to remove each tradesperson's uploaded information.

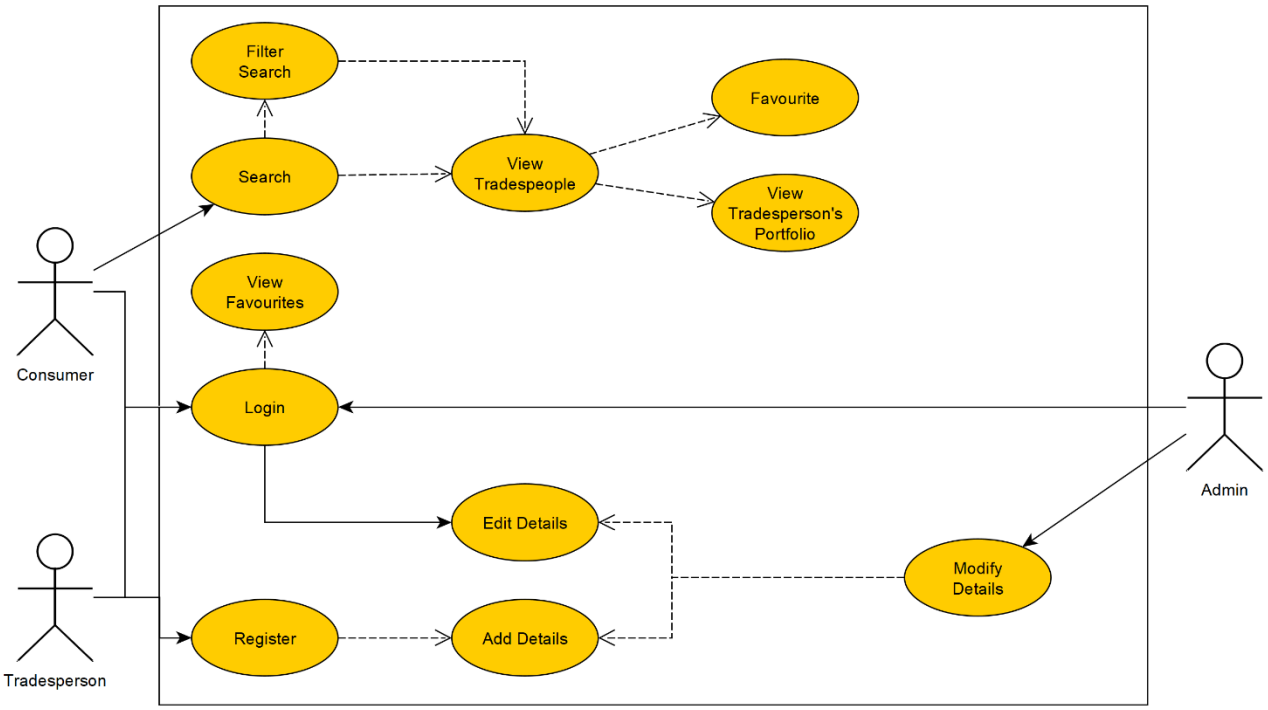


Figure 11: Use Case Diagram - Overview of Application

As seen within the use case diagram above, there are an array of features on offer for both, consumers, and tradespeople. When a tradesperson initially visits this web application, they will be asked to login or proceed to sign up before they are provided the capabilities to add their own professional information. This information ranges from their working hours to the distance they are willing to travel. As well as this, tradespeople will have to fill out a portfolio detailing any previous work they have carried out. Once this information has passed all error checking, it will be displayed to the application for consumers to view. However, the application's admin will ultimately hold the right to remove any information uploaded by tradespeople.

A consumer on the other hand will be able to query tradespeople in their area without the need to login. Providing the consumer with this functionality without the need to login is key to ensuring that the consumer continues using the application, rather than turning to a competitor site. From here the consumer can filter their search to their needs. From the returned search, the consumer is displayed a geographical representation of each tradesperson's location as well as brief description of the service they provide. This description will include helpful information such as their contact information. From here, consumers will also be given the opportunity to view each tradesperson's portfolio. Consumers who login will be provided with the extra functionality of being able to favourite any tradesperson they come across and by doing so, they will be populated into the application's favourites tab. Thus, allowing the consumer to easily find their preferred tradespeople on future occasions. In addition to this, they will also save to the consumer's device so that they can be viewed during offline use.

3.4.3 Class Diagram

The following diagram depicts the classes and interfaces identified for the front-end of this application. This class diagram was compiled during the design stage of this application, leaving some scope for features to be added later in the development process.

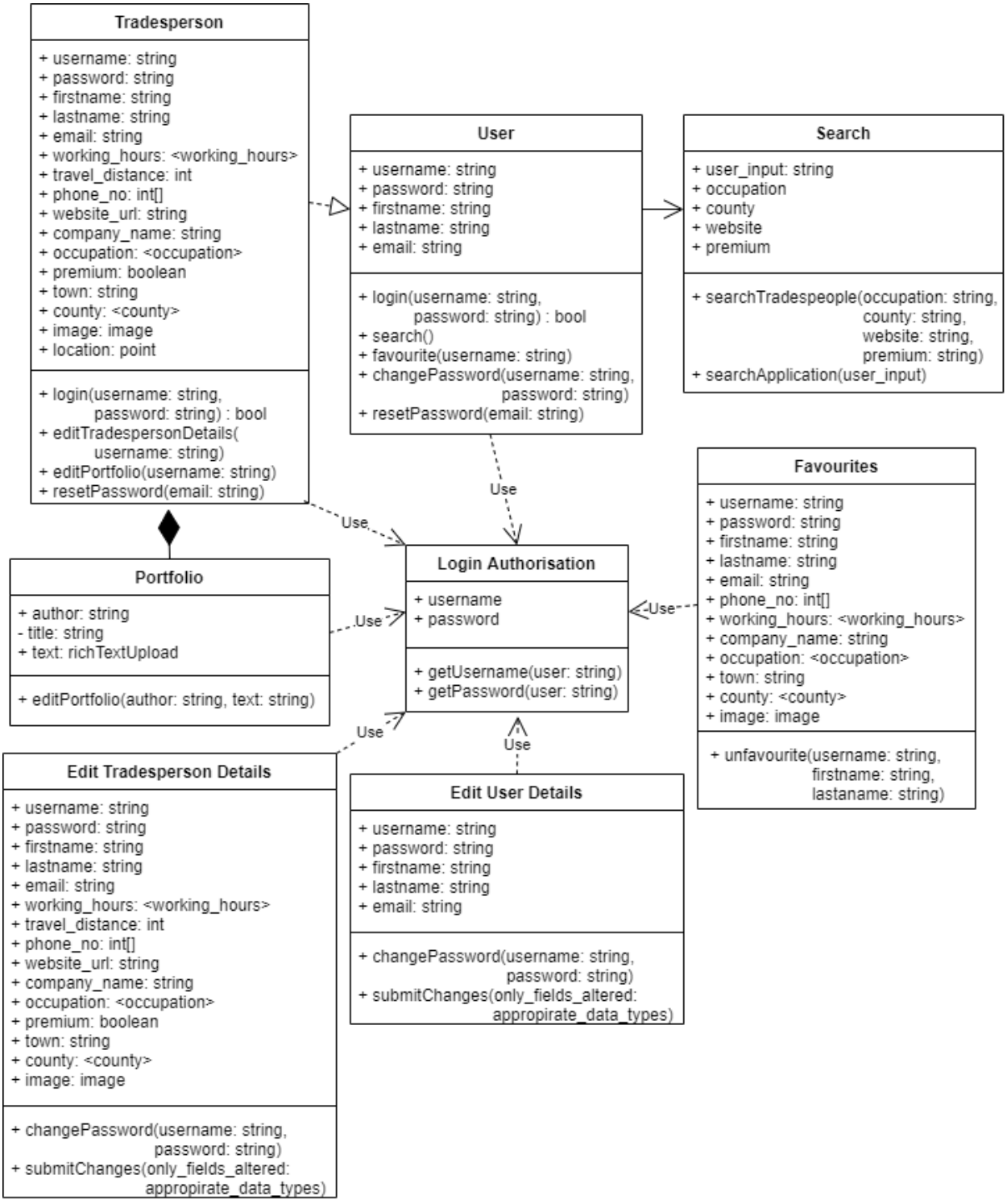


Figure 12: Class Diagram of Tradespeople Near Me Front-end

Figure 12 illustrates the class structure of this application and the relationships they share with one another. There are two types of users of the system, consumers and tradespeople, with consumers being represented by the user class. Tradespeople inherit the same attributes given to consumers with this being shown by the relationship shared between them. In addition to these, tradespeople are given many more attributes so that the intended functionalities of the system may be implemented.

Upon visiting the application, consumers are free to search for tradespeople in their area without the need to create an account. However, should they decide to favourite a tradesperson, they must login to the system. Tradespeople must do likewise before being able to access any of their details. This is evident from the composite relationship connecting the portfolio class to the tradesperson class. Subsequently, the login authorization class will play a significant role in the allocating of the available functionality. Each of the project's classes are provided with multiple functionalities with many of their attributes being passed so that the desired features can be carried out. Despite being a time-consuming process, the designing of a class diagram provides developers with a great overview of the required system as well as preparing them for the substantial work that lies ahead.

3.4.4 Front-end Architectural Diagram

The front-end of this application will be developed using Python, JavaScript, HTML and CSS while also making use of third-party JavaScript libraries. Tradespeople Near Me will utilize the UI library Bootstrap so that the application can be used on any device without the need for developing extensive CSS for all device sizes. This application will follow the front-end architectural pattern, MVC as shown in figure 13 below.

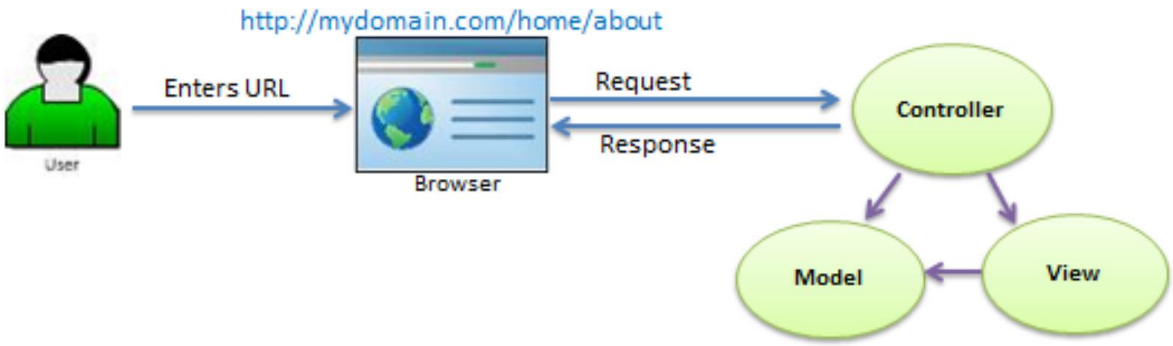


Figure 13: User Interaction with the MVC Architectural Pattern (47)

Here, it can be seen how the MVC pattern's components interact with each other, as well as how the MVC pattern interacts with the client accessing the browser. The MVC pattern separates an application into three components, model, view and controller, with each having their own specific function. The Model component stores retrieved data and as its related business logic before being requested by the View in order to display the user interface. The controller's role on the other hand is to manage the user's requests as it "renders the appropriate view with the model data as a response" (47). As depicted in the figure above, the user requests data from the controller and after interacting with the Model and View components, it returns the appropriate response. Implementing the MVC architectural pattern enables code to be easily maintained so that the system can be debugged in an efficient manner. As well as this, the MVC pattern enables parallel development of the various components of the system, meaning that its different functionalities can be developed simultaneously. Implementing an MVC pattern provides numerous benefits associated with making the system development cycle more efficient. It is therefore an ideal architectural pattern for this application, where time is such a valuable commodity.

3.4.4 Front-end Prototype Design

The high-fidelity prototype below was developed after extensive research and design. Although it requires a significant investment of time, a high-fidelity prototype allows developers to design a model closer to that of the final product. For this project, it helps to separate the various components of the application. Before developing the project, low-fidelity prototypes were first created using paper. This is shown in the appendix from figures 80 – 82.

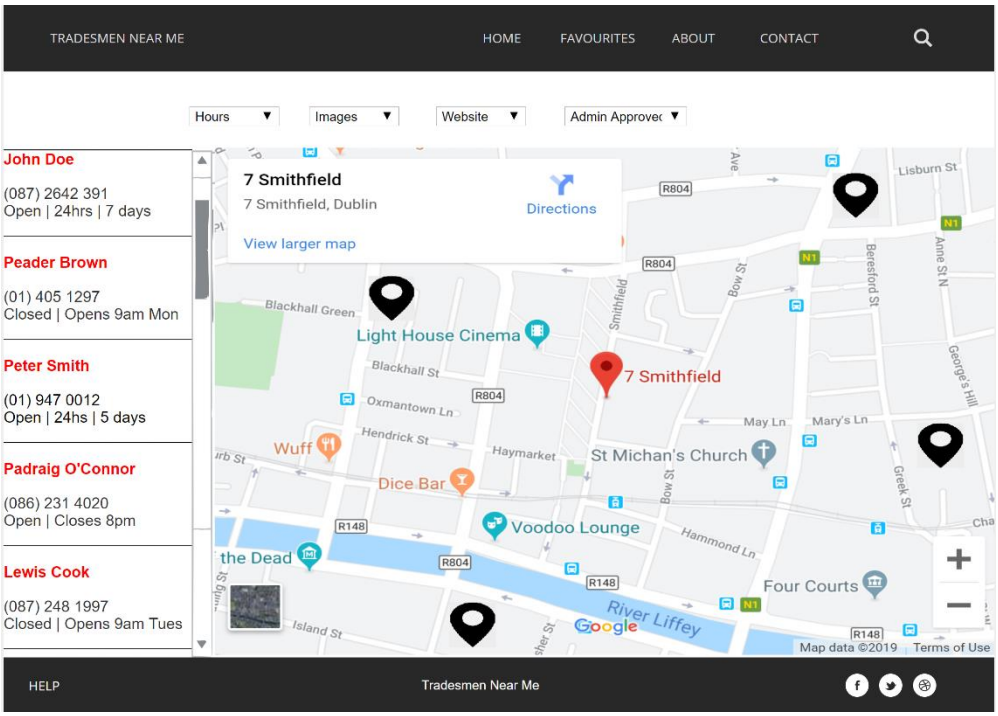


Figure 14: Front-End Prototype Design

Figure 14 represents the desired front-end of this application. Here, the consumer searching for a particular tradesperson can see their location as displayed in red while the locations of tradespeople in their area are highlighted in black. The consumer will also be able to filter their search based on their requirements using the supplied filtering options. Upon searching, the consumer will be provided with a graphical representation of the returned tradespeople alongside a concise scrollable list showing a brief description of their page.

3.5. Middle-Tier

3.5.1 Consumer Middle-Tier

The middle-tier of any web application refers to the handling of user requests such as submitting a form. In the case of a consumer in this application, it alludes to the searching and favouriting functionalities performed by consumers. This searching functionality is shown in figure 15 below.

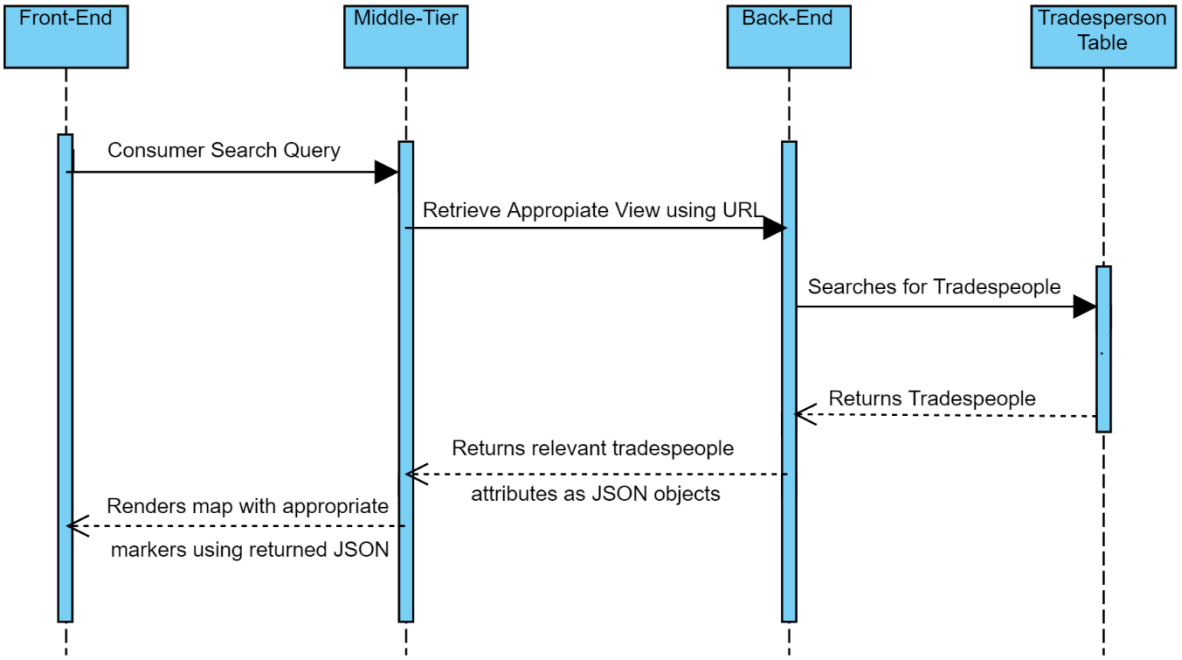


Figure 15: Sequence Diagram for consumers querying tradespeople

Here, it is clear to see the different operations assigned to each layer of the application. Following a consumer search, the appropriate SQL statement queries the PostgreSQL database. It then navigates to the previously created Tradespeople table where it iterates over each row in order to return the necessary tradespeople. After this, the relevant tradespeople are returned to the front-end of the application so that they can be rendered as markers on a map and subsequently displayed to the user.

3.5.2 Tradesperson Middle-Tier

In contrast, for tradespeople accessing this web application, the middle-tier performs a different operation despite consisting of a similar form. It can be seen when a tradesperson adds their own information for their page, as shown below.

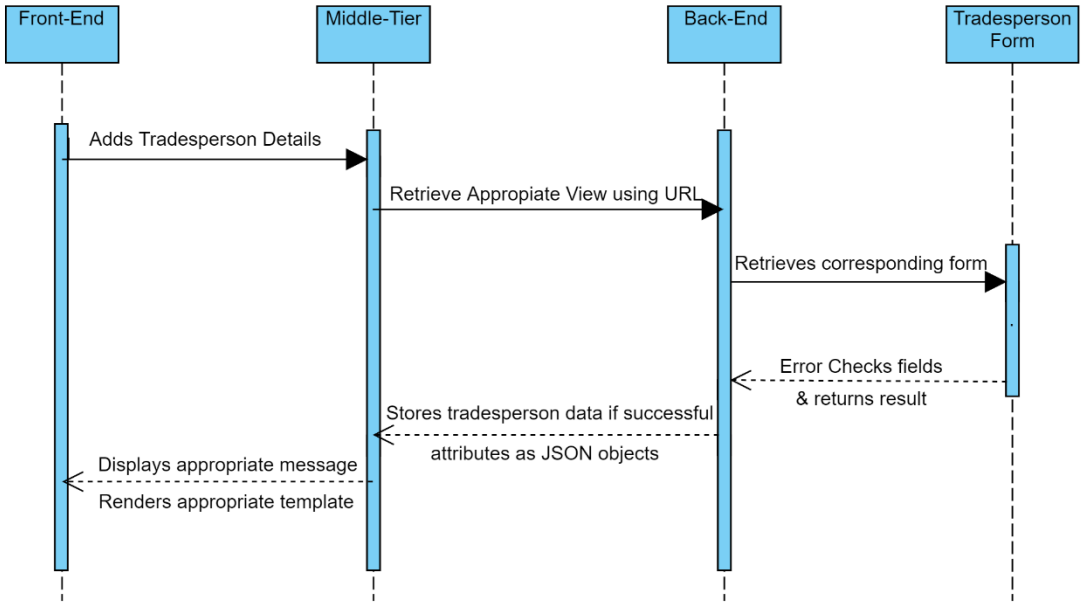


Figure 16: Sequence Diagram for registering tradespeople

Sequence diagrams such as the one shown in figure 16 allow developers to easily document a system's requirements during the designing stages of a project. When a tradesperson registers with this application, their submitted information will be retrieved by the application's form and returned to its back-end view. This data is then error checked and once approved, it is then stored in the pre-defined database. Therefore, enabling consumers to view the newly registered tradesperson.

3.5.3 Favouriting a Tradesperson

The act of favouriting a tradesperson can be performed by a consumer and will involve the most complex middle-tier functionality within the application. This is shown in the figure outlined below.

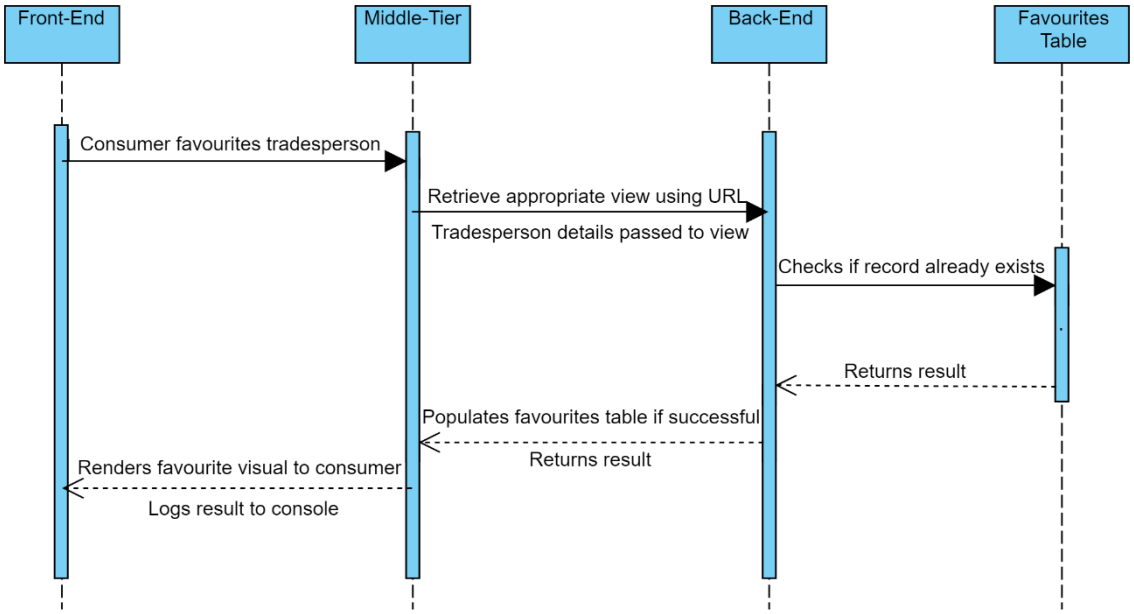


Figure 17: Sequence Diagram for favouriting a tradesperson

The favourites middle-tier functionality differs to that of the previous ones in that it requires the sending of information from the application's front-end to its back-end, depending on what tradesperson the consumer has chosen to favourite. Following this, the tradesperson will then be stored in the favourites table so that it can be later rendered to the consumer. Exactly how this functionality is implemented will be expanded on in the development section of this report.

3.6. Back-End

The figure below features an Entity Relationship Diagram (ERD) which illustrates the database entities needed to implement this application, as well as their relationships between one another.

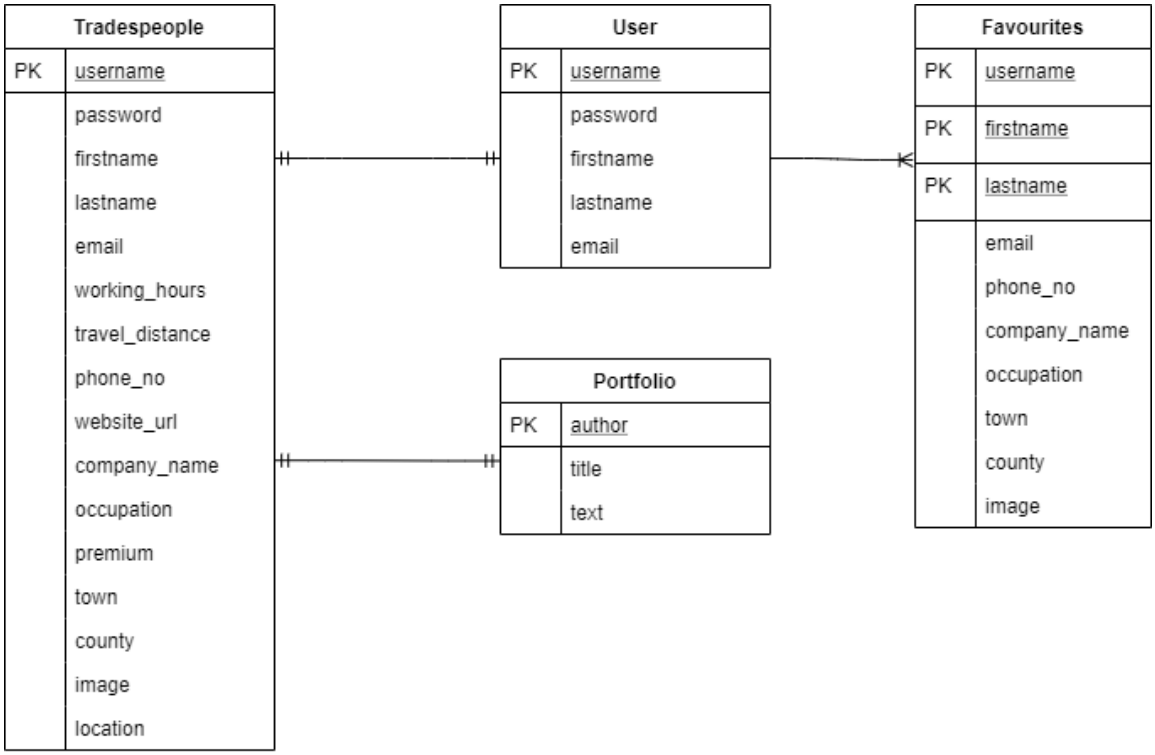


Figure 18: ERD for Tradespeople Near Me

The diagram above details the relationship diagram of this system's database schema which will be implemented using PostgreSQL. Each table's corresponding primary keys are clearly stated as a single attribute, except for the favourites table whereby the username, firstname and lastname attributes are required to find a distinctive record within the database. It can also be noted that the tradespeople table consists of a significant number of attributes and therefore will used to provide most of the application's functionalities. In addition to this, each tradesperson will have a portfolio which will be linked via the foreign key, author. Author will obtain the same value stored as the tradesperson's username so that it can be easily accessed on demand.

The ERD also emphasises the functionality given to the consumer which allows them to favourite multiple tradespeople. Subsequently, the user and favourites tables have a one-to-many relationship. As the favourites table is storing tradespeople and the consumer who is favouriting them, it therefore consists of a combination of attributes found in both, the user and tradespeople tables. Designing this ERD was an essential part in the planning of this application's back-end as it gives an accurate visual representation of its database layout.

3.7. Conclusions

With a chosen methodology, design and initial prototype developed, the development of this application's core functionality can take place. The next development stages of the project will see the project being developed in accordance to the diagrams defined in this section.

4. Prototype Development

4.1. Introduction

This project's prototype development was centred around the configuration of its various components. The main focus of its early development evolved around the connecting of this application's chosen technologies. The implemented technologies as well as their key functionalities are explained in greater detail in the following section.

4.2. Project Structure

One of the major benefits of using Django as the project's framework, is that it allows for the separation of functionalities. This can be seen in figure 19, whereby the accounts, favourites, searching and wysiwyg applications have their own directory within the overall Tradesperson project. Structuring the project like this, enables each application to have its own models, views, urls and forms, with each of these components having their own individual purpose in according to its Model-View-Template architectural pattern.

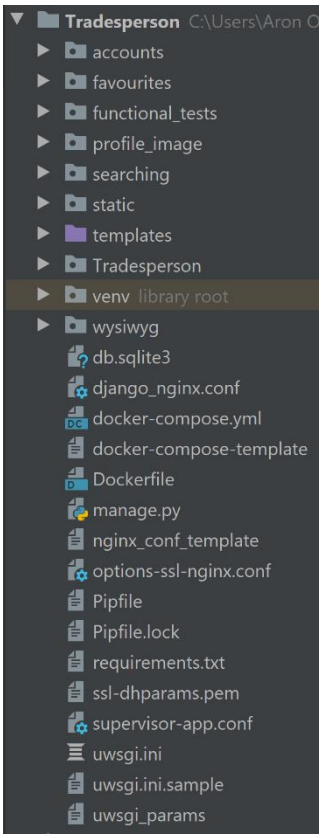


Figure 19: Project Structure

The models are defined in models.py and serve to represent a collection in the application's Postgres database. The views found in views.py contain functions so that the application can read various web requests and issue an appropriate response. The role of urls.py is to define url patterns so that Django can iterate through them to find the requested url. The application's templates associated with more than one application are defined in the project's templates folder as shown in the appendix at figure 91. Each application then has their own set of template files within their directory.

Each of these application's must then be declared as installed apps inside the project's settings.py so that their files may be recognised by the project. Tradesperson's settings.py is responsible for the central configuration of all the project's implemented components. These include its installed applications, middleware, database and deployment technologies. Therefore, the application's settings file is the most important file within the project's directory above.

Not only does Tradespeople Near Me need HTML files to render the application's templates, it also requires additional files such as JavaScript, CSS and images. In Django, these files are referred to as 'Static' files and declaring them in a single directory allows them to be easily stored once deployed. The various static files incorporated into this application are shown in the appendix at figure 90. Storing them in a single location enables each functionality to then be stored in separate sub-directory. Therefore, the static files within this project include mapping, images, admin and CKEditor. The purposes for each of these will become apparent in the subsequent sections.

The numerous files stored in the application's root directory as shown in figure 19 are incorporated for deployment reasons, with these files also being explaining further in its corresponding section. This clear separation of application logic allows for faster development as it allows the different components of the application to be worked on simultaneously. Therefore, as the project grew in complexity, it became apparent that Django was the perfect choice for the application's wide range of functionalities.

4.3. Front-End

4.3.1 Introduction

This section will discuss the operation of the application's templates and their associated functionalities, of which there are many. Therefore, the covered technologies include HTML, CSS and JavaScript, with Leaflet being implemented for the application's mapping functionality.

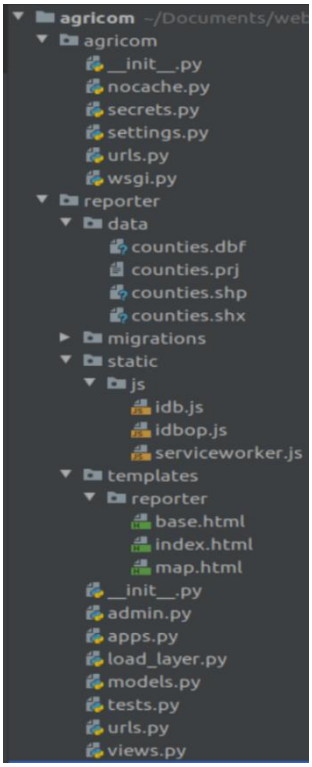


Figure 20: Application's Directory

4.3.2 Early Development

The initial development of the application's front-end concentrated on the use of Django as a framework. Django is a Python-based framework which follows its own variation of the Model-View-Controller architectural pattern, called Model-View-Template. Django therefore enables developers to divide their project into three key aspects. This separation of functionality helps to deliver a fast solution using higher quality code.

PyCharm was chosen as this application's IDE (Integrated Development Environment) in which to implement Django. Tradespeople Near Me combines the front-end technologies, HTML, CSS and JavaScript so that a clear, event-driven user interface can be displayed.

The files associated with the early development of this project can be seen clearly within the application's directory in figure 20 above. The HTML files have their necessary CSS code within them as the CSS was yet to go into excessive detail. The JavaScript files as shown within the static folder were responsible for the handling of Tradespeople Near Me's service worker. Figure 21 illustrates this project's early front-end development.

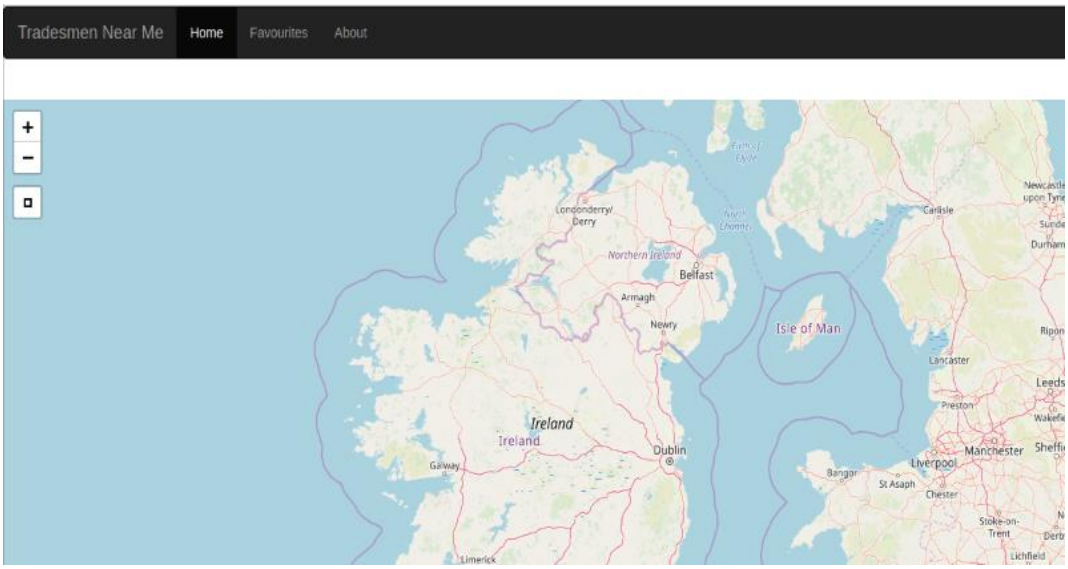


Figure 21: Front-End User Interface

Despite focusing the majority of the project's initial efforts on the internal structure of the application, the development of its front-end successfully incorporated a GeoDjango map into its display. Following this early development, the application's functionalities could be concentrated on so that the best possible user experience could be obtained. This will now be discussed in the subsequent section.

4.3.3 Tradesperson Listings

The aim of Tradespeople Near Me is to efficiently display to a consumer the available tradespeople in their area. This is accomplished using premium, latest and related listings to display the relevant data.

Most of the data that will now be discussed is returned to the application's templates via JavaScript. Exactly how this data is retrieved will be covered in the ensuing back-end section of this document. When recursively creating HTML elements from JavaScript, one must account for various components such as bootstrap classes and IDs so that these elements can be styled accordingly. After these elements have been successfully created and assigned their relevant styling, they must be then appended to an existing HTML Document Object

Model (DOM). Each of the following application's listings follow this format, with a snippet of this being shown in the appendix at figure 83.

Upon visiting the website, the first listings presented to the user is that of the premium listings. The focus of these listings is to provide tradespeople with the opportunity to obtain a prominent position in which to be viewed. This section is therefore reserved to those individuals or companies who are willing to contribute financially to obtain a significantly better position on this application. Not only will this increase views for those who acquire this service, but it will almost certainly increase their daily enquires. Thus, making it a significantly beneficial service for any aspiring tradesperson to obtain.

Directly below the premium listings are the latest listings and it is here where the last four tradespeople to sign up get displayed. Once a tradesperson has successfully signed up, they will be redirected to the application's home page where they will be able to see their details displayed in this section. This section provides tradespeople with an insight into what it is like to secure the premium service detailed above.

After a consumer has viewed a tradesperson's portfolio, the related listings get populated at the bottom of the page. The related listings are populated with tradespeople whom share the same occupation to that of the viewed tradesperson. This subtle addition adds to the overall efficiency of the application as it uses this opportunity to display similar tradespeople to that of the use's previous search.

Each listing displayed to the user is recursively called using the various components Django has to offer, with this being expanded on in greater detail in the subsequent sections. The returned attributes are then displayed to the front-end of the application so that a consumer searching for a tradesperson can see who it is they are contacting, what occupation they hold and what area they are situated in. These attributes enable the consumer to decide instantly whether that tradesperson is of relevance to them.

Tradespeople listings were incorporated into this application to add to its overall efficiency, catering to those individuals who desire not to use Tradespeople Near Me's mapping functionality. For those who require a tradesperson imminently, listings offer a faster alternative than that of the provided mapping functionality. Thus, making it a worthwhile feature to have added to the application.

4.3.4 Tradesperson Portfolio

Tradespeople Near Me acts as a medium between consumers and tradespeople, targeting quality tradespeople as opposed to quantity. This is achieved using a portfolio, which forces tradespeople who wish to register with the platform, to upload examples of previous work in the form of textual description and images.

In addition to the tradesperson details shown to the consumer as outlined above, there will be a link displayed so that they can view further details as well as their portfolio of the tradesperson's pervious work. Despite containing substantial more information, a tradesperson's portfolio is retrieved using a single JavaScript element shown in figure 22.

```
for (var i = 0; i < data.length; i++) {
    if (data[i].author === username) {
        portfolioListings.innerHTML = "";
        var portfolioListingsDiv = document.createElement( tagName: 'DIV');
        portfolioListingsDiv.classList.add("portfolioListingsName");
        portfolioListingsDiv.innerHTML = data[i].text;
        portfolioListings.appendChild(portfolioListingsDiv);
    }
}
```

Figure 22: JavaScript Displaying Tradespeople's Portfolio

Here, it is seen how the attribute, 'text' is assigned before being appended to the appropriate element. This simple font-end implementation is thanks to the application's clever design which sees the incorporation of a 'wysiwyg' portfolio. A tradesperson's portfolio can also be used by them to provide further details such as working hours, phone number and email to consumers. This can be seen in the figure below.

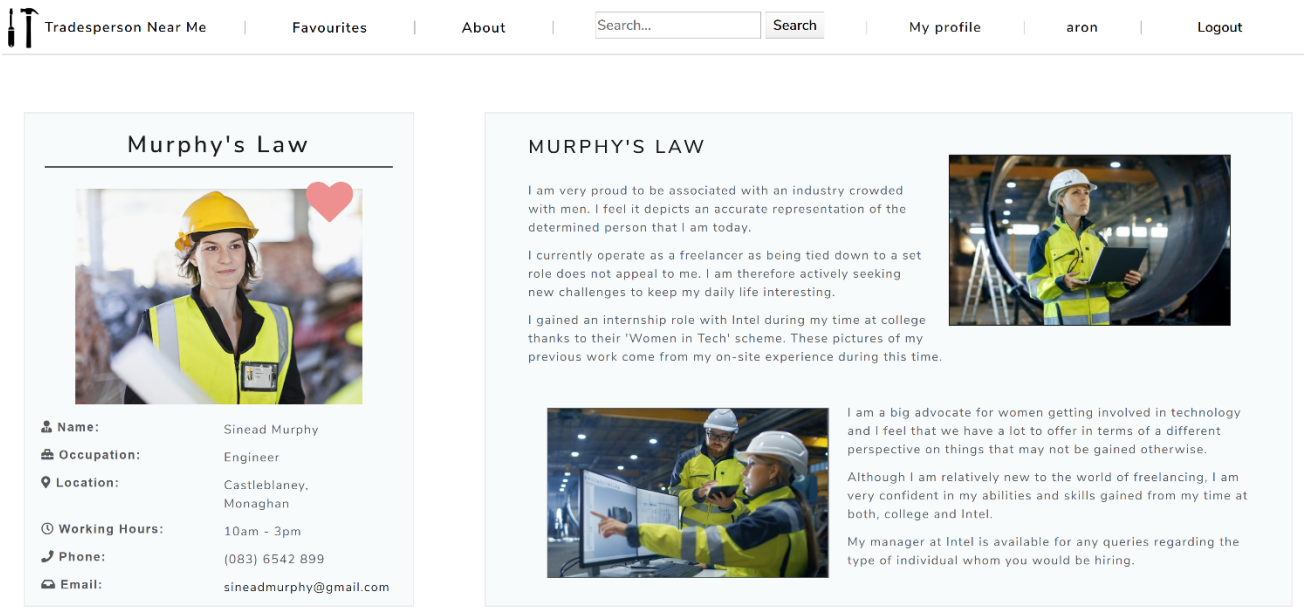


Figure 23: Tradesperson Portfolio

A tradesperson's portfolio enables tradespeople to add a personal touch to advertise their services in an already crowded industry. Not only this, but it provides tradespeople with a chance to clearly outline their skillset to the consumer using imagery and text. How a tradesperson wishes their portfolio to look is completely up to their discretion, with previous work, quotes and references all being potentially appealing to a consumer.

A portfolio is a requirement for a tradesperson who wishes to register with this application as it is an extremely important component to ensuring consumers are hiring a proven, trustworthy tradesperson. A tradesperson's portfolio is not a common piece provided by similar existing applications and for this reason, it was prioritised in the development of this application.

4.3.5 User Authentication

Tradespeople Near Me is targeted towards both, consumers and tradespeople with it providing login and signup functionality for each. While tradespeople must register with the application to avail of its services, consumers do not. Despite this, login functionality is provided for both.

Upon visiting this web application, consumers can search for tradespeople in their area, regardless if they have signed into the application using a previously created account. However, should they decide to create an account, they are then given extra functionality which allows them to save their desired tradespeople into their favourites tab for future and offline use. This is achieved using a service worker and enables this application to function as a progressive web app. “A Progressive Web App (PWA) is a web app that uses modern web capabilities to deliver an app-like experience to users” (48). Such modern web capabilities include the ability of the application to function even when the device is offline. How Tradespeople Near Me implements this technology will be explained in greater depth in the subsequent middle-tier section.

When a user first opts to create an account, they can navigate to the relevant page by first clicking on the login button provided to them in the navigation bar. This then takes them to the applications login menu where they can decide to either login using a previously made account or create a new account. Should a new user decide to create an account, they must then choose whether to sign up as a consumer or as a tradesperson, with both providing contrasting signup forms. While a consumer must provide a unique username, password, first name, last name and email, a tradesperson must fill out an extensive form detailing everything about the service they are seeking to provide. These required fields range from a username and password to a profile picture and maximum travel distance. Furthermore, they are required to fill out their portfolio before being allowed to proceed. These fields are then displayed to the application’s corresponding templates using Django’s crispy forms. Crispy forms is an application “that will let you control the rendering behaviour of your Django forms in a very elegant and DRY way” (49). How this technology is displayed to the relevant template is evident in the figures shown below.

```
<div class="form-group col-md-4 mb-0">
    {{ form.username|as_crispy_field }}
</div>
<div class="form-group col-md-4 mb-0">
    {{ form.password|as_crispy_field }}
</div>
<div class="form-group col-md-4 mb-0">
    {{ form.password2|as_crispy_field }}
</div>
```

Figure 24: Tradesperson Details Form

```
<div class="form-row col-md-12" id="postForm">
    <p>Your Portfolio:</p>
    {{ form2.text | safe }}
    {{ form2.media }}
</div>
```

Figure 25: Tradesperson Portfolio Form

From these figures, it can be clearly seen how two forms are accessed using a single template so that the registering Tradesperson can fill out the necessary fields for both, their details and portfolio. The subsequent processing of these two forms is discussed later in the application's back-end.

Tradespeople who have logged in to this application will then be shown an advertisement in the form of the modal shown in figure 26 below. As a website is not a requirement for the signup form and is not something most tradespeople tend to have, they will likely consider this offer. Should they opt to send an email asking for a quote, they can click on the 'Yes Please' button so that they can send an email to the specified address. Implementing the front-end hyperlink, 'mailto:arononeill98@gmail.com?subject=Website Enquiry' enables this email to be sent to the project owner's address with the subject, 'Website Enquiry'. This advertisement serves to generate further business so that an eventual loop can be created between different platforms and hence create endless opportunities. This will be expanded on in the future work of this application.

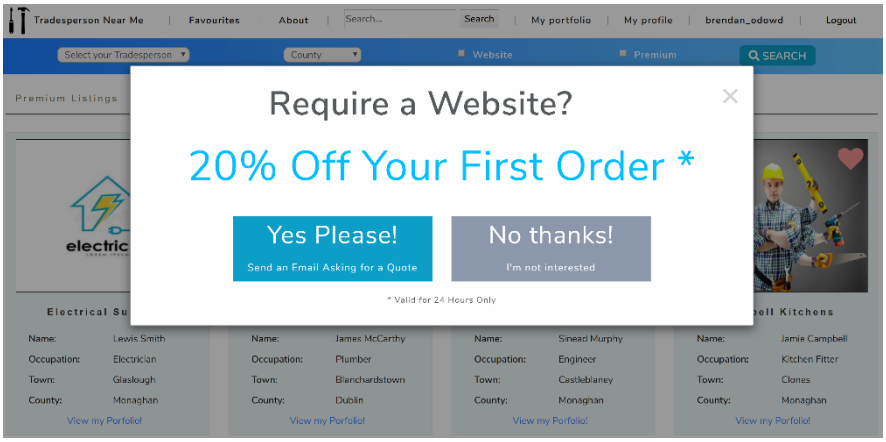


Figure 26: Website Advertisement Modal

After a user has successfully signed up, they are redirected to the application's home page where they can then see their username displayed in the navigation bar. This was designed so that the user has a visual confirmation that they are currently logged in. As well as this, the previously seen login button is altered so that it now displays and functions as a logout button. The distinct separation of the discussed users is essential to maintaining the usability of the application.

4.3.6 Edit User Details

Once a user has successfully created an account and logged into the application, they are provided with the ability to edit their submitted details during registration. This applies to both, consumers and tradespeople but is significantly more beneficial to tradespeople.

Both types of users can modify their stored details by accessing the appropriate page shown in the application's navigation bar. For consumers, it is displayed as a button which redirects them to the 'edit_profile.html' page. Once here, they can view their existing attributes and change these details if desired. Similar to the signup page, these details are displayed using crispy forms, and once these have been successfully altered, the consumer is redirected to the application's home page. When this has occurred, consumers can be assured that their newly submitted details are stored in the database.

Tradespeople on the other hand, will be given the option to modify their details using two separate links, one of which being for their details, and the other being for their portfolio. Both options can be found in the application's navigation bar and will enable tradespeople to alter important details such as contact information. This will appeal to tradespeople who have to alter their working hours based on other factors such as family commitments. Tradespeople will also be able to update their portfolio to accommodate for their most recent work. As a tradesperson gains further experience in their industry by carrying out more complex work, they can avail of this service so that they can advertise their highest quality work.

Should a user also forget their password, they have the option to send themselves an email so that their previously forgotten password can be retrieved. In the user-authenticating focused society that we live in today, it is safe to say that users will utilize this functionality. However, this service currently works locally as it requires an SMTP (Simple Mail Transfer Protocol) server to run. This is therefore covered in the project's future work which is documented in a later section. Additionally, if a consumer desires to change their password, they can follow the link provided in their edit details page so that they can reach the appropriate template. Once here consumers are asked to fill in their old password followed by their new password, of which needs to be confirmed and error checked in a similar manner to the registration page.

Giving consumers the capability to edit their profile is extremely important as they may want to change their login details to something more memorable to them. Providing tradespeople with the ability to update their details was seen as a key component to this application as they have to continuously evolve their expertise based on what previous clients have requested. Thus, enabling them to better cater towards the next potential client.

4.3.7 Favourites

The aim of this application is to provide consumers with the most efficient service possible. One component that plays a central role in achieving this goal is that of its favouriting functionality. Exactly how this functionality was implemented will now be explained in the following section.

Consumers whom have logged into this application are provided with the extra functionality which allows them to store tradespeople as favourites. This is apparent through the favourite button provided on each tradesperson's image as shown in figure 27. After a consumer

Figure 27: Favourite Button

has favourited their desired tradesperson, they will then see this information populated in the application's favourites page.

Consumers can navigate to their favourited tradespeople at any point using the link provided in the applications header. Once here, they will be displayed a listing of their previously favourited tradespeople so that their relevant information can be retrieved with ease. As well as this, consumers will have the ability to unfavourite tradespeople when desired. This is shown in the figure below.

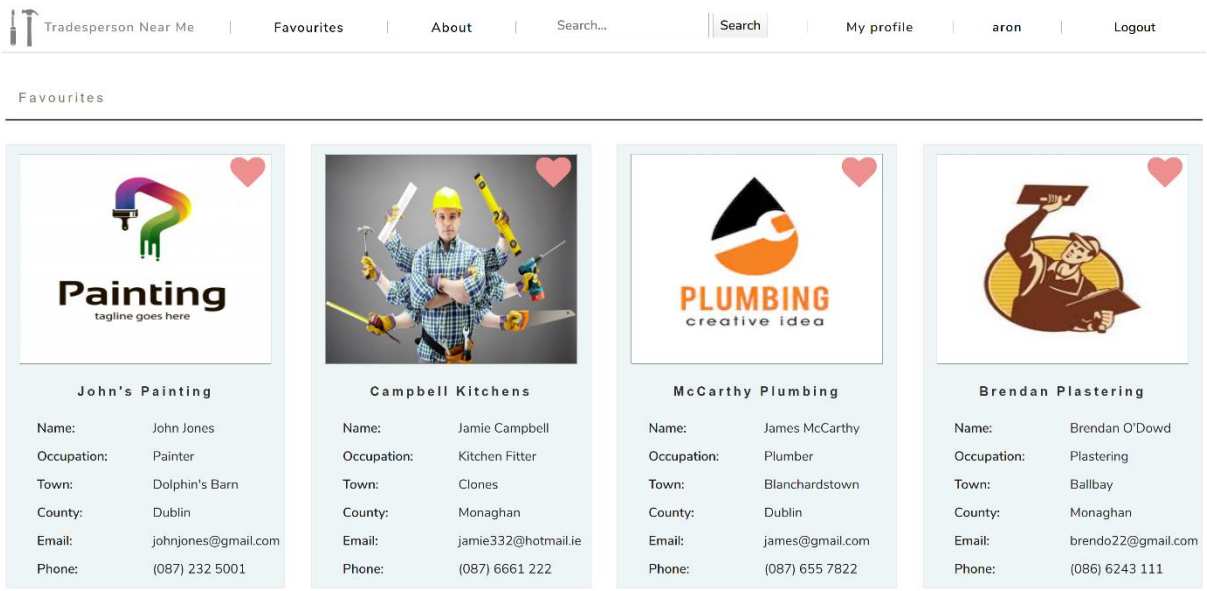


Figure 28: Favourites Page

Implementing this favouriting functionality enables consumers to store their desired tradespeople so that they can be easily accessed for future and offline use. This service therefore played an important part in the application's development as it helps to improve the overall efficiency of the application.

4.3.8 Searching

Consumers visiting Tradespeople Near Me can also look for their desired tradesperson via the application's searching functionality as displayed in its navigation bar.

Following a search, the application then iterates over each tradesperson record in the database to find information related to their query. This queried information includes each tradesperson's attributes, occupation, first name, last name, town, county and company name. After a successful query, the consumer is redirected to the program's search page where they are shown a comprehensive list of their results in the form of a listing as described previously. This is executed using a loop as shown in the following figure.

```
{% for each_queryList in queryList %}
    <div class="col-md-3" id="SearchListingsDiv">
        <div class="container" id="PremiumDetailsInner">
            {% if each_queryList.image %}
                <div class="container">
                    <img class="PremiumListings1" src="{{each_queryList.image.url}}" alt="No Image">
                    <div class="favDiv">
                        <i class="i_element" onclick="addFavourite($(this), '{{ each_queryList.firstname }}',
                                '{{ each_queryList.lastname }}', '{{ each_queryList.email }}',
                                '{{ each_queryList.phone_no }}', '{{ each_queryList.company_name }}',
                                '{{ each_queryList.occupation }}', '{{ each_queryList.town }}',
                                '{{ each_queryList.county }}', '{{ each_queryList.image.url }}');"></i>
                        <span class="span">Button</span>
                    </div>
                </div>
            {% endif %}
            <div class="PremiumDetails">
                <h3 class="service-heading" id="PremiumListingsName">{{each_queryList.company_name}}</h3>
                <p class="col-md-5 col-5" id="floatLeft"><b>Name: </b></p>
                <p class="col-md-7 col-7" id="floatRight">{{each_queryList.firstname}} {{each_queryList.lastname}}</p>
                <p class="col-md-5 col-5" id="floatLeft"><b>Occupation: </b></p>
                <p class="col-md-7 col-7" id="floatRight">{{each_queryList.occupation}}</p>
                <p class="col-md-5 col-5" id="floatLeft"><b>Town:</b></p>
                <p class="col-md-7 col-7" id="floatRight">{{each_queryList.town}}</p>
                <p class="col-md-5 col-5" id="floatLeft"><b>County:</b></p>
                <p class="col-md-7 col-7" id="floatRight">{{each_queryList.county}}</p>
            </div>
        </div>
```

Figure 29: Displayed Searching Attributes

Following a search from a consumer, each relevant record is iterated over so that their attributes can be accessed using dot notation. It is thanks to this format that there is no limit to the data being displayed to the template. Thus, comprehensively satisfying the consumer's request.

The aim of Tradespeople Near Me is to provide a suitable platform for all possible consumers, with each implemented functionality taking us closer to this goal. Providing the consumer with a search engine enables them to look for a known tradesperson without the need to sift through various information. This subsequently connects the consumer to their desired tradesperson in a faster manner, saving invaluable time as a result.

4.3.9 Mapping

Tradespeople Near Me's mapping functionality was the main focus of application's front-end development and can be seen on screen following the premium and latest listings. Leaflet was incorporated in conjunction with JavaScript so that the appropriate markers could be displayed to the consumer.

Upon visiting Tradespeople Near Me, a consumer will be greeted with an alert asking them for permission to access their location. This simple request then has a significant impact on how the application operates. The handling of this request is shown in the figure below.

```
function our_layers(map) {
    navigator.geolocation.getCurrentPosition(function(position) {
        console.log("Location permissions granted");
        createMapWithLocation(map, position.coords.latitude, position.coords.longitude);
    },
    function(error) {
    if (error.code === error.PERMISSION_DENIED)
        console.log("Location permissions denied");
        createMap(map);
        showAllPremiumTradespeople(map);
        showAllLatestTradespeople(map);
    });
}
```

Figure 30: User Location Permissions

Should they desire not to supply this application with access to their location, they will be initially shown every tradesperson who has registered with the service. This information will then be displayed to them on a map and will only be filtered after a user has selected their desired tradesperson options from the provided dropdown list. The available filters to choose from include type of tradesperson, county, and whether they own a company website. As an additional feature, consumers can also choose to view tradespeople who have opted for the application's premium service. After a consumer has filtered their search based on their needs, they are redirected to the application's map which will now only display tradespeople related to their search. These flexible searching options enable Tradespeople Near Me to cater to anyone who interacts with the service, regardless of their location.

A disadvantage to the consumer not providing their location is that the application will not be able to take their distance to each tradesperson into account. Consumers will therefore have to click on each tradesperson individually to check whether they are willing to travel to their area. Clicking on any tradesperson on the map enables the consumer to view their maximum travel distance in the form of a radius around the chosen tradesperson's marker. As well as this, it will also bind a popup to the marker so that the tradesperson's name and occupation can be displayed. Despite still being highly effective, this option does diminish the application's overall efficiency. However, this efficiency is significantly increased whenever the visiting consumer allows the application to access their location. Once the consumer has agreed with the provided location permission, the application can implement the Haversine formula as detailed previously in the project's research. This is accomplished by passing the latitude and longitude of the user as well as the stored tradespeople to the function 'getDistanceBetweenPoints' as shown in figure 31.

```
function getDistanceBetweenPoints(lat1, lon1, lat2, lon2) {
    var R = 6371; // Radius of the earth in km
    var dLat = (lat2-lat1) * (Math.PI/180);
    var dLon = (lon2-lon1) * (Math.PI/180);
    var a =
        Math.sin( x: dLat/2) * Math.sin( x: dLat/2) +
        Math.cos( x: (lat1) * (Math.PI/180)) * Math.cos( x: (lat2) * (Math.PI/180)) *
        Math.sin( x: dLon/2) * Math.sin( x: dLon/2)
        ;
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt( x: 1-a));
    var distance = R * c;
    return distance;
}
```

Figure 31: Havsersine Formula

Here, it can be seen how the parameters lat1, lng1, lat2 and lng2 are passed to the function with the computed distance being returned following the relevant calculations. This is implemented following Django's architectural pattern, Model-View-Template in which the template in question calls the appropriate view via the application's url. This is executed using an AJAX call, 'new L.GeoJSON.AJAX("{% url 'tradesperson' %}" which in turn calls the view 'tradesperson_data' so that each tradesperson is returned as geojson. After this function has been called, the returned distance between the consumer and tradesperson

gets compared to that of the tradesperson's maximum desired travel distance. This is shown in the figure below.

```
var points = new L.GeoJSON.AJAX("{% url 'tradesperson' %}",{
    onEachFeature: function(feature, layer) {
        var tradespersonLat = layer._latlng.lat;
        var tradespersonLng = layer._latlng.lng;

        var distance = getDistanceBetweenPoints(userLat, userLng, tradespersonLat, tradespersonLng);
        var travel_distance = feature.properties.travel_distance;
        if (distance < travel_distance) {
            var name = feature.properties.firstname + " " + feature.properties.lastname;
            if (feature.properties.occupation === 'Plumber') {
                var lat = layer._latlng.lat;
                var long = layer._latlng.lng;
                var marker3 = L.marker([lat, long], {
                    icon: redIcon
                }).bindPopup(name + '<br>' + feature.properties.occupation)
```

Figure 32: Haversine Computation Compared with Tradesperson's Maxium Travel Distance

Those tradespeople who's defined travel distance falls within the calculated distance are bound to the map so that they can be viewed by the consumer. However, those who do not fall within this distance are dismissed as potential tradespeople. Subsequently removing any tradespeople from the map who are outside the scope of the consumer. This then ensures that only the relevant tradespeople are being displayed to the consumer, with these tradespeople being bound to the map using a Leaflet marker.

When a consumer chooses to view a tradesperson's portfolio, they will see all other tradespeople cleared from the map which is directly below their portfolio. This leaves their chosen tradesperson as the only viewable marker. The consumer is made aware of this by the now fixed popup showing the tradesperson's name and occupation. Should they decide to view another portfolio, their map will navigate to the newly selected tradesperson's location whilst leaving their previously viewed tradesperson's marker on screen. This feature allows the consumer to gauge which tradesperson would be better suited to them, relative to that of their location.

From the features detailed above, it is easy to see how the development of the application's mapping functionality took precedence over the application's many other front-end features. The implemented Haversine formula enables Tradespeople Near Me to become a faster and more efficient alternative to any existing application. However, it was extremely important that each of the services outlined above were dovetailed seemingly so that the application could operate smoothly and therefore provide the best possible user experience to the consumer.

4.3.10 Conclusion

The majority of Tradespeople Near Me's front-end functionalities are dynamically created using JavaScript and is therefore extremely important that each of these features are integrated seamlessly into the application's templates. As a result, a significant amount of development time was invested into the functionalities outlined above, with complex features such as mapping and favouriting taking precedence over others.

4.4 Middle-Tier

4.4.1 Introduction

This application's middle-tier consists of a service worker and various AJAX requests. A service worker is implemented into this project so that it can converted into a progressive web app (PWA) while AJAX requests are executed so that its front-end can effectively communicate with its back-end. The internal workings of these two complex features will now be covered in the subsequent section.

4.4.2 Early Development

The Middle-Tier of this application initialled consisted of Fetch API calls to the relevant methods as defined in its views. These Fetch API calls are incorporated into the configuration of this application's service worker.

A service worker “is essentially a JavaScript file that runs separately from the main browser thread, intercepting network requests, caching or retrieving resources from the cache, and delivering push messages” (50). A service worker allows developers to control how network requests are handled with a given page. Before, incorporating a service worker into this application, an SSL cert had to be obtained as service workers only run over HTTPS. The SSL cert for this application can be seen in the at the url shown in the subsequent figure.

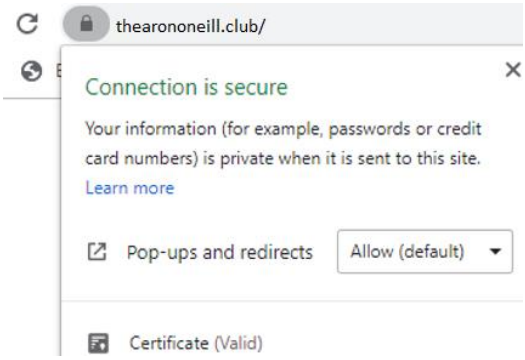


Figure 33: SSL certificate

There are many other important benefits associated with the acquiring of an SSL cert. SSL is used to encrypt sensitive information sent across the internet as well as authenticating the server its being sent to. An SSL cert is displayed to the left of the URL as shown in the figure above. “Using the Fetch API inside a service worker, we can intercept network requests and then modify the response with content other than the requested resource” (50). This is illustrated in figure 34.

```
//collect latest post from server and store in idb
fetch('http://127.0.0.1:5006/getdata').then(function(response){
    return response.json();
}).then(function(jsondata){
    console.log(jsondata);
    dbPromise.then(function(db){
        var tx = db.transaction('feeds', 'readwrite');
        var feedsStore = tx.objectStore('feeds');
        for(var key in jsondata){
            if (jsondata.hasOwnProperty(key)) {
                feedsStore.put(jsondata[key]);
            }
        }
    });
});
```

Figure 34: Implementation of a service worker

In the figure above, it can be seen how the initial service worker fetched the appropriate API which in turn queried the ‘getdata’ method, as defined in this application’s views.py file. The received API response then returned the requested counties as JSON objects. The returned

objects were subsequently used to include a service worker with the application as outlined in figure 84 in the appendix.

As the development of the application continued, so did the implementation of its service worker. Subsequently, this resulted in a move away from the service worker as depicted in figure 34 above, and towards a more elaborate service worker that not only stored JSON data, but also full HTML and JavaScript pages. As the data being returned to the application's template also grew in complexity, so did the calls needed to access this information. This ultimately saw the incorporation of AJAX requests into the program's middle-tier as detailed below.

4.4.3 'GET' AJAX Requests

The middle tier of this application takes the form of AJAX requests and urls. The urls of any Django application enables developers to connect a template (front-end) to its corresponding view (back-end). To facilitate this connection, an AJAX request must be made.

Tradespeople Near Me implements two different kinds of AJAX requests, depending on the type of data being returned. For the first, most common AJAX call made in this application, the data is returned as JSON data. In order to return JSON data, an XMLHttpRequest() is made using the appropriate url. This can be seen with each piece of functionality in the application, with each request being tailored to the requirement of that function. This can be seen in figure 35, where the url being called is dictated by the options the user selects to filter their search.

```
function showAllPlumbers(occupation, WebsiteCheckedValue, PremiumCheckedValue, county,
    var req = new XMLHttpRequest();
    var url;
    if (occupation === 'Builder') {
        if (WebsiteCheckedValue) {
            if (PremiumCheckedValue) {
                url = '/builder_data_website_premium';
            }
            else {
                url = '/builder_data_website';
            }
        }
        else {
            if (PremiumCheckedValue) {
                url = '/builder_data_premium';
            }
            else {
                url = '/builder_data';
            }
        }
    }
```

Figure 35: AJAX Request for Filtered Tradespeople

Here, it can be seen how the url being called is based on the options selected by a consumer searching for their ideal tradesperson. Each of the urls called then correspond to a url defined in the application's 'urls.py'. This can be seen at figure 87 in the appendix of this documentation. Defining separate urls based on their desired implementation increases the runtime efficiency of this application as it enables the data to be filtered before being returned. Another example of this operation can also be found in the appendix at figure 85, whereby the data being returned only consists of premium tradespeople. Each url called corresponds to a relevant view, with this being extremely beneficial as it inhibits the processing of redundant JSON data.

Tradespeople Near Me makes a second type of AJAX request to accommodate its mapping functionality. It differs to that of the previously described AJAX request in that it returns the GeoJSON data needed for the appropriate tradespeople to be placed as markers on the application's map. Similar to the first AJAX request, it calls a url which returns all tradespeople from the database. This information then gets returned in a GeoJSON format so that both, the tradesperson's attributes and their location can be accessed. This is shown in the figure below.

```
var points = new L.GeoJSON.AJAX("{% url 'tradesperson' %}",{
    onEachFeature: function(feature, layer) {
        var tradespersonLat = layer._latlng.lat;
        var tradespersonLng = layer._latlng.lng;

        var distance = getDistanceBetweenPoints(userLat, userLng,
        var travel_distance = feature.properties.travel_distance;
        if (distance < travel_distance) {
```

Figure 36: AJAX Request Returning GeoJSON data

Following the AJAX request to the application's database, the 'onEachFeature' method enables each tradesperson to be iterated over so that the subsequent code can be executed for each tradesperson. The retrieved GeoJSON takes the form of a feature and a layer, whereby the feature is used to implement the Haversine formula, and the layer is used to plot the location of the tradesperson on the map. This AJAX request is made on three separate occasions within 'index.html' so that it can handle which tradespeople to plot on the map depending if the consumers allows their location to be used. It is also implemented once more so that when a consumer views a tradesperson's portfolio, only that tradesperson is visible on the map.

AJAX requests are fundamental to any complex Django application as they serve to connect a template to its relevant view so that any stored data can be displayed to the user. Without the correct implementation of these calls, a visiting consumer will be shown otherwise irrelevant data. This makes AJAX requests an integral part of this application as Tradespeople Near Me is an information driven application.

4.4.4 'POST' AJAX Requests

'POST' AJAX requests in this application are implemented so that a consumer can favourite and unfavourite their desired tradespeople. With each tradesperson being dynamically created and displayed via JavaScript, this technology must also be implemented in the same way. In addition to these requests, the application will retrieve a newly registered tradesperson's location using a third type of 'POST' request. These three different

implementations of ‘POST’ requests in this application will now be covered in the following section.

The first of these AJAX requests that a consumer has a chance to execute is for favouriting a tradesperson. When each tradesperson displayed to the consumer is iterated over, an event listener is also created so that their own AJAX request can be made once clicked. This is accomplished using jQuery as shown in figure 37.

```
i_element.addEventListener( type: "click",  listener: function () {
    //AJAX Implementation
    $(this).toggleClass("press", 1000);
    const values = [data[i].firstname, data[i].lastname, data[i].email, data[i].phone_no,
        data[i].company_name, data[i].occupation, data[i].town, data[i].county, data[i].image];
    $.ajax({
        type: "POST",
        url: "/edit_favorites/",
        data: {'tradesperson': values},
        traditional: true,
        dataType: 'html',
        success: function () {
            $('#message').html("<h2>Favourite Submitted!</h2>")
        }
    });
});
```

Figure 37: AJAX Request to Favourite Tradespeople

Here, it can be seen how the instance of the clicked element is retrieved using ‘$(this)’ so that an additional class can be appended to it. This class then gives the button a red colour so that the consumer is provided with a visual representation of the implemented functionality. After this, the values of the selected tradesperson are assigned to an array so that they can be passed to the corresponding view. jQuery enables the type, url and data to be defined so that the relevant functionality can be implemented. In figure 37, it is shown how the necessary tradesperson details are passed to the url ‘edit_favourites’ with this being accomplished using the type ‘POST’. These values are passed to the back-end of the application so that its corresponding functionality can be executed.

The next AJAX request that a consumer can interact with is unfavouriting a tradesperson. This is achieved through the application’s favourite page and although it is similar to the previous request, it differs in that it is called through HTML.

After a consumer has decided to remove one of their favourite tradespeople, they can then click on the close button so that it passes the tradesperson's first and last name to the 'removeFavourite()' function shown in figure 38. Here, these two parameters are defined in an array so that they can be passed to its corresponding view through the url, 'remove_favourite'. Following the successful implementation of this view, the page is then reloaded to show their updated list of favourite tradespeople.

```
function removeFavourite(firstname, lastname) {
    //AJAX Implementation
    $(this).toggleClass("press", 1000);
    const values = [firstname, lastname];
    $.ajax({
        type: "POST",
        url: "/remove_favorite/",
        data: {'remove_tradesperson': values},
        traditional: true,
        dataType: 'html',
        success: function (msg) {
            window.location.reload();
            console.log(msg);
        }
    });
}
```

Figure 38: AJAX Request to Unfavourite Tradespeople

The final 'POST' AJAX request carried out in this project sees the tradesperson's location passed as two coordinates to the relevant view. This is accomplished using the json data type and its method 'stringify' as shown in figure 39. They are required to be passed in this format so that its back-end functionality can be executed, with this being outlined in its corresponding back-end section.

```
function updateTradespersonLocation() {
    navigator.geolocation.getCurrentPosition( successCallback: function(position) {
        const values = [position.coords.latitude, position.coords.longitude];
        $.ajax({
            type: "POST",
            url: "/update_location/",
            dataType : 'json',
            data: JSON.stringify( value: {'data': values}),
            traditional: true,
            success: function () {
                $('#message').html("<h2>Update Location!</h2>")
            }
        });
    },
     errorCallback: function(error) {
    if (error.code === error.PERMISSION_DENIED)
        console.log("Location permissions denied");
    });
}
```

Figure 39: AJAX Request to Update Tradesperson's Location

The purpose of these 'POST' AJAX requests is to pass the relevant tradespeople details to the application's back-end views so that the corresponding functionality may be executed.

After these views have been carried out, their result gets logged to the console so that its outcome can be easily tracked.

4.4.5 Service Worker

Implementing a service worker in this application carried huge importance as it enables it to function offline and hence become a progressive web app. This is achieved by defining a service worker in the configuration of the application's settings as well as executing the appropriate JavaScript code so that its files can be cached in the user's device. The exact workings of this functionality will be discussed in the following section.

Before developing a service worker, one must define it in the application's settings. This can be seen from the command shown in figure 40. However, for it to operate universally across any device once deployed, it must also be configured in the application's installed apps.

```
PWA_SERVICE_WORKER_PATH = os.path.join(BASE_DIR, 'static/js/', 'serviceworker.js')
```

Figure 40: Configuration of a Service Worker in Setting.py

Defining a service worker like so enables it to be registered, with this being evident in Google Chrome's inspect mode in the appendix at figure 85. From the command above, it can be seen how the service worker is declared as a JavaScript file in the application's static folder. Within this file, the application's cache name is defined as 'v1' and the files in which the application will cache are defined as 'cacheFiles', with this being shown in figure 41.

```
var cacheFiles = [
    'static/css/main.css',
    'static/js/cached_pages.js',
    'static/js/index.js',
    '../',
];
```

Figure 41: Service Worker Cached Files

Once a service worker has been registered, it must then be installed so that all the defined files above can be cached as assets required to run the application. This is executed using the function shown below.

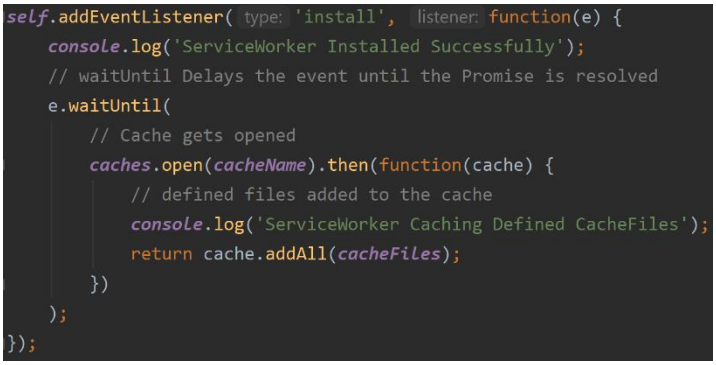


Figure 42: Service Worker's Install Event Handling

Here, it is shown how the 'install' function is implemented as a promise object, meaning that it "represents the eventual completion (or failure) of an asynchronous operation, and its resulting value" (51). It is executed in this way so that so that it ensures the defined files are installed before they are activated. Once installed, these files are compared within any existing cached files so that the most recent version may be cached. This is accomplished by carrying out the '.map()' function so that each individual file can be iterated over. The if statement displayed in the figure below checks if the cached file is saved under a previous cache name and subsequently deletes if this is the case. Thus, allowing for the updated file to be stored.

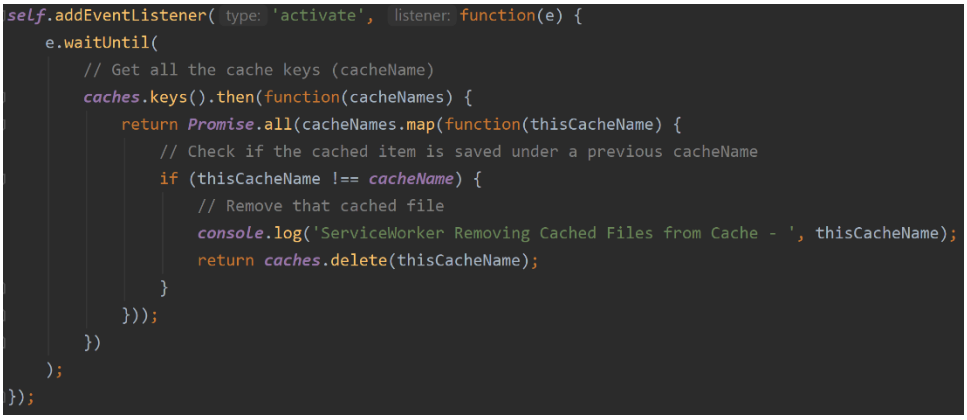


Figure 43: Service Worker's Activate Event Handling

The service worker's 'fetch()' method is then implemented so that it prevents the default and allows for its request to be handled using its 'respondWith()' method. The 'cache.match(e.request)' method then enables the application to check in the cache to see if the requested url already exists and returns the cached version if it is. Therefore, the cache will not have to be fetched again. This is illustrated in the figure below.

```
self.addEventListener( type: 'fetch', listener: function(e) {
    console.log('ServiceWorker Fetch Executed', e.request.url);
    e.respondWith( // e.respondWidth Responds to the fetch event
        caches.match(e.request) // Check in cache for the request being made
            .then(function(response) { // If the request is in the cache
                if ( response ) {
                    console.log("ServiceWorker Found in Cache", e.request.url, response);
                    return response; // Return the cached version
                }
                var requestClone = e.request.clone(); // If request is NOT in the cache, fetch and cache
                fetch(requestClone)
                    .then(function(response) {
                        if ( !response ) {
                            return response;
                        }
                        var responseClone = response.clone();
                        caches.open(cacheName).then(function(cache) { //  Open the cache
                            cache.put(e.request, responseClone); // Put the fetched response in the cache
                            console.log('[ServiceWorker] New Data Cached', e.request.url);
                            return response; // Return the response.
                        });
                    })
                    .catch(function(err) {
                        console.log('ServiceWorker Error Fetching & Caching New Data', err);
                    });
            })
    );
});
```

Figure 44: Service Worker's Fetch Event Handling

However, if the cache is not found, it must be fetched again. This is accomplished using the '.clone()' method so that the event request stream can be duplicated and executed once more. Following this, its response is error checked before it is cloned in the same manner as the request. Using the previously defined cache name, 'v1', the cache is then opened before it is populated with the newly fetched response.

It is necessary to log all steps associated with the caching of the application's files so that they can be tracked by the developer. This, combined with the '.catch()' method enables for faster debugging should any problems arise from during development.

4.4.6 Conclusion

This project's middle-tier went from initially being a small piece of the application's development to becoming arguably the most important. Implementing AJAX requests

enables data to be sent and received from the application's back-end while the service worker allows consumers to access their desired tradespeople offline. As a result, executing both of these features correctly carried significant importance during the application's development.

4.5. Back-End

4.5.1 Introduction

The development of this projects back-end included the use of Postgres as its database and Digital Ocean as its cloud provider. These, along with the incorporation of Docker accounted for a significant portion of the application's development time. Postgres is configured in application's settings so that it can define database attributes such as engine, name and host. The layout of the directory files associated with Tradespeople Near Me's deployment technologies can be seen in figure 45. However, before delving into the details of how this application connects to Docker, one must examine how it connects to Postgres using its various models and views.

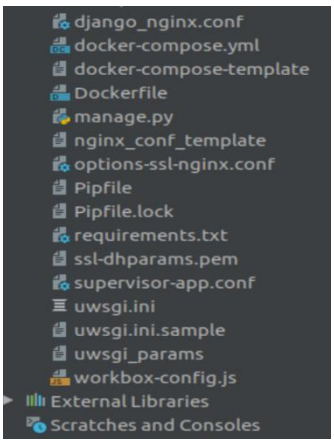


Figure 45: Back-end configuration files

4.5.2 Models

To recap, a Django application implements a Model-View-Template architectural pattern, meaning that its back-end consists of both, the application's models, and its corresponding views. Tradespeople Near Me implements four separate models, three of which are custom models while the other one is a built-in user model. These models are then accessed in the application's views, and therefore must be clearly outlined before discussing each view's separate functionality.

The first implemented model in this application is its built-in user model. It is implemented for visiting consumers as they require little details in comparison to that of a tradesperson.

This model consists of the attributes, username, password, email, first name and last name, as well as providing an active boolean variable so that the current user can be retrieved.

The second model defined in this application is the Tradesperson model. It comprises of the same attributes found in the user model but also comes with additional attributes as it is significantly more complex. These include the tradesperson's working hours, maximum travel distance, phone number, website url, company name, occupation, town, county and profile image. As well as this, their location is stored using a PointField variable and a premium boolean is defined so that those who have availed of the application's premium service can be tracked. Defining their location as a PointField variable enables both, their latitude and longitude to be stored. This, as well as every other attribute can be seen in the model shown in the figure below.

```
class Tradesperson(models.Model):
    username = models.CharField(max_length=25)
    password = models.CharField(max_length=20)
    firstname = models.CharField(max_length=20)
    lastname = models.CharField(max_length=20)
    email = models.CharField(max_length=30)
    working_hours = models.CharField(max_length=30)
    travel_distance = models.CharField(max_length=20)
    phone_no = models.CharField(max_length=20)
    website_url = models.CharField(max_length=20, blank=True, null=True)
    company_name = models.CharField(max_length=20)
    occupation = models.CharField(max_length=25)
    premium = models.BooleanField(default=False)
    town = models.CharField(max_length=25)
    county = models.CharField(max_length=25)
    image = models.ImageField(upload_to='profile_image', blank=True, null=True)
    location = models.PointField(srid=4326)
    objects = GeoManager()
```

Figure 46: Tradesperson Model

Here, it can be seen how the attribute, 'image' is uploaded to the 'profile_image' folder so that it can be correctly accessed in application's tradesperson signup form.

Despite all these tradesperson attributes, another model was created so that each tradesperson's portfolio could be stored. This model was defined as 'Post' and took the

form of a ‘wysiwyg’ editor so that each tradesperson can upload a personalised portfolio for consumers to view. This can be seen in figure 47.

```
class Post(models.Model):
    author = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    title = models.CharField(max_length=30)
    text = RichTextUploadingField(blank=True, null=True)
```

Figure 47: Post (wysiwyg) Model

Despite only consisting of three attributes, the ‘text’ attribute enables tradespeople to upload a portfolio tailored to them using the ‘RichTextUploadingField’ variable type. This portfolio model then relates to the Tradesperson model via its foreign key ‘author’. Therefore, allowing it to be stored using the tradesperson’s corresponding username.

The final model defined in this application was the favourites model. The purpose of this model is to store each consumer’s favourited tradespeople using the attributes shown below.

```
class Favourites(models.Model):
    username = models.CharField(max_length=25)
    firstname = models.CharField(max_length=20)
    lastname = models.CharField(max_length=20)
    email = models.CharField(max_length=30)
    phone_no = models.CharField(max_length=20)
    company_name = models.CharField(max_length=20)
    occupation = models.CharField(max_length=25)
    town = models.CharField(max_length=25)
    county = models.CharField(max_length=25)
    image = models.ImageField(upload_to='profile_image', blank=True, null=True)
```

Figure 48: Favourites Model

Here, it can be seen how the majority of the previously defined tradespeople attributes are stored along with the username of the current user. Storing the current user instead of the tradesperson’s username enables each tradesperson in the Favourites model to be cross referenced with that of the current user so that only their favourites are displayed.

Each of these models are then accessed in the application’s views so that their corresponding functionalities can be carried out. These views will now be discussed in the following section.

4.5.3 Tradesperson Listings

The three types of listings displayed to the user on the home page of this application include premium, latest and related listings. This section will now detail how each of these listings are retrieved as JSON using their corresponding view.

Premium Listings was the first of these to be developed, with the view 'getAllPremiumTradespeople' being defined so that only the premium tradespeople get returned. This is achieved by returning a filtered set of Tradespeople objects as shown in the figure below.

```
def getAllPremiumTradespeople(request):
    listPremiumPlumbers = list()
    premiumTradespeople = Tradesperson.objects.all().filter(premium=True)
    plumber_data(request)

    for people in premiumTradespeople:
        print(people)
        ser = TradespersonSerializer(people)
        listPremiumPlumbers.append(ser.data)
    import json
    return HttpResponse(json.dumps(listPremiumPlumbers))
```

Figure 49: Premium Tradespeople View

Here, it can be seen how the filter 'premium=True' gets executed so that only the desired premium tradespeople are returned. Each returned tradesperson object is then iterated over so that its data can be serialized before being appended to a list. The model 'TradespersonSerializer' is a serialized version of the Tradesperson model defined previously and is a necessary component when returning data to the necessary template. This model can be seen in the appendix at figure 88. After this data has been successfully serialized, it is then returned as JSON to the appropriate front-end template using a HTTP Response.

The application's latest listings take a similar form to that of the premium listings, with tradespeople now being ordered based on the time in which they were added to the model. This is executed using the '.order_by('id)' extension in place of the previously described

filter() method. The workings of the latest listings view can also be seen in the appendix at figure 88.

After a consumer has viewed a portfolio, the application retrieves similar listings for them to view. This is implemented using the related listings function found in its the front-end so that all tradespeople can be retrieved via the 'getAllTradespeople' view. As shown below, this view is executed so that this information is returned as JSON data.

```
def getAllTradespeople(request):
    listPlumbers = list()
    tradespeople = Tradesperson.objects.all()
    for people in tradespeople:
        ser = TradespersonSerializer(people)
        listPlumbers.append(ser.data)
    import json
    return HttpResponse(json.dumps(listPlumbers))
```

Figure 50: View Retrieving all Tradespeople

Unlike the previous listings, related listings request that all tradespeople be returned so that each tradesperson's occupation can be compared to that of the previously viewed tradesperson.

Although these views are not technically necessary for this application to function, they can be seen as essential components to ensuring a swift runtime experience for any visiting consumer.

4.5.4 Tradesperson Portfolio

A consumer has the option to view each tradesperson's portfolio using the link provided in the application's home page. This information is then retrieved from two separate models before being displayed to the application's template.

Using an event listener, the username of the tradesperson clicked is passed to the relevant function. An AJAX request is then called so that that all tradespeople can be returned from the application's back-end. This is achieved using the 'getAllTradespeople' view as defined previously. This view once again enables each tradesperson's details to be serialized before being returned in the form of JSON. This JSON data is then compared to that of the tradesperson clicked as defined in the application's front-end section.

The next step in the retrieval of each tradesperson's portfolio is to query the Post model so that only the clicked tradesperson is displayed to the consumer. Similar to the previously view, the Post model is returned to the appropriate function using the method shown in figure 51.

```
def getPortfolio(request):
    listPortfolio = list()
    posts = Post.objects.all()
    for post in posts:
        ser = PostSerializer(post)
        listPortfolio.append(ser.data)
    import json
    return HttpResponse(json.dumps(listPortfolio))
```

Figure 51: View to Retrieve Tradesperson's Portfolio

Here, it can be seen how each tradesperson's portfolio is acquired using the 'Post.objects.all()' command so that this information can be serialized and returned as JSON. This view is implemented with relative ease thanks to the application's cleverly designed 'wysiwyg' model. Therefore, despite consisting of few lines of code, this view enables each tradesperson's portfolio to be returned and subsequently displayed to the consumer.

Retrieving information from both, the tradesperson and the post models is vital to ensuring the consumer can view all possible data available to them. Defining the portfolio model as a 'wysiwyg' editor played an important role in that it enabled the serialization of each tradesperson's previous work without the need the need for unnecessary complexity. As the 14th-century logician and theologian, William of Ockham, states: "the simplest solution is almost always the best" (52).

4.5.5 User Authentication

The back-end of this application's user authentication can be viewed in two separate sections, the first being the user login and the second being the user registration. As a user

can sign up as either a tradesperson or a consumer, two separate registration forms were required to be developed. Therefore, a view and its corresponding form was created for each of these three options. The development of these components will now be discussed in the subsequent section.

When a consumer chooses to register with this application, they will be displayed a form which in turn gets processed by its appropriate view. This registration form is rendered to the consumer with the fields, username, password, first name, last name and email as seen in the appendix at figure 93. This form is then accessed in the view shown below so that the values entered by a consumer can be validated.

```
def user_register_view(request):
    next = request.GET.get('next')
    global_user_form = GlobalUserRegisterForm(request.POST or None)

    if global_user_form.is_valid():
        User = global_user_form.save(commit=False)
        password = global_user_form.cleaned_data.get('password')
        User.set_password(password)
        User.save()
        login(request, User)
        return redirect('/')

    context = {
        'form': global_user_form
    }

    return render(request, 'UserSignUp.html', context)
```

Figure 52: Consumer Registration View

Here, it can be seen how the entered values are retrieved using the 'cleaned_data.get()' method before being thoroughly error checked. These error checks are performed on the password and email fields so that the confirmed value of each matches the original. Not only this, but the password must be at least 6 characters in length and contain both, a number and capital letter. Making these measures a requirement for consumers is within their best interest as well as the applications as it adds to the security of the service. Should a user fail to meet these requirements, they are shown an appropriate error message. Once this error checking has been successfully performed, the consumer has their details stored

in the database via the ‘User.save()’ command. Following this, they are logged in before being redirected to the application’s home page.

However, should the visiting user opt to register as a tradesperson, they will be met with more stringent error checking measures than that of a consumer. A tradesperson requires interaction with all three models within the application and their corresponding forms, with significant more development time being committed as a result. This view therefore sees a tradesperson’s details added to the Tradesperson, User and Post Models. This is evident in the figure below, whereby all three forms are called prior to any error checking being performed.

```
def tradesperson_register_view(request):
    next = request.GET.get('next')
    form = UserRegisterForm(request.POST, request.FILES)
    global_user_form = GlobalUserRegisterForm(request.POST or None)
    post_form = PostForm(request.POST, request.FILES)
    if form.is_valid():
        if global_user_form.is_valid():
            if post_form.is_valid():
                password = form.cleaned_data.get('password')
                password2 = form.cleaned_data.get('password2')
                email = form.cleaned_data.get('email')
                email2 = form.cleaned_data.get('email2')
                phone = form.cleaned_data.get('phone_no')

                if password == password2:
                    if len(password) < 6:
                        messages.info(request, 'Make sure your password is at least 6 characters!')
                    elif re.search('[0-9]', password) is None:
                        messages.info(request, 'Make sure your password contains a number!')
                    elif re.search('[A-Z]', password) is None:
                        messages.info(request, 'Make sure your password contains a captial letter!')
                    else:
                        if email == email2:
                            if len(phone) > 7:
```

Figure 53: Tradesperson Register View

After retrieving the necessary attributes from the tradesperson registration form, ‘UserRegisterForm’, they are then error checked just like the consumers were. However, in addition to these checks, is the check for the length of the entered phone number, of which there needs to be at least 7 digits. Following the successful error checking of these retrieved attributes, the entered data gets stored in the application’s three models. The first, most complex of these models to have its data stored is the Tradesperson model. However,

before being stored, the tradesperson's latitude and longitude is obtained using a geocoder as shown in the figure below.

```
Tradesperson1 = form.save(commit=False)
Tradesperson1.password = password
g = geocoder.ip('me')
lat = g.lat
long = g.lng
point = Point(x=long, y=lat)
Tradesperson1.location = point

User = global_user_form.save(commit=False)
password = global_user_form.cleaned_data.get('password')
User.set_password(password)

Post = post_form.save(commit=False)
User.save()
Tradesperson1.save()
login(request, User)
Post.author = request.user
Post.save()
return redirect('/')
```

Figure 54: Tradesperson Data handling following error checking

These same details then get populated in the user model before the tradesperson's portfolio can be saved. Following this, the instance of the current user is retrieved so that its username can be assigned to the author of the Post model. When these three records have been successfully stored in their corresponding models, the registering tradesperson is logged in before being redirected to the home page of the application.

Once a user has created their desired account, they can simply login when they visit the application again. This is achieved by querying the user model so that the entered data in the login template can be authenticated. This can be seen in the appendix at figure 94. Adding the tradesperson to the User model as detailed above, enables this data to be verified using the one model. This marginally improves the running performance of the application as it does not need to query multiple models to search for this information. In the same manner as the registration, the user then gets logged in and redirected to the home page of the application.

4.5.6 Edit User Details

Tradespeople Near Me provides users with the ability to edit their profile in a similar way to that of the registration page. However, in this case, the application retrieves the previously submitted data via the appropriate form.

With two separate types of users, comes two separate views in which to implement this functionality on the application's back-end. The first of these views to be developed enables Tradespeople to modify their details using the fields shown in figure 55. Outlining these fields in 'forms.py' enables their records to be returned from the application's database so that they can be displayed to the user. After a tradesperson has submitted their new details, these are then error checked in the same way as they were when registering, with this being shown in the appendix at figure 94.

```
class Meta:
    model = Tradesperson
    fields = [
        'username',
        'password',
        'firstname',
        'lastname',
        'email',
        'working_hours',
        'travel_distance',
        'phone_no',
        'website_url',
        'company_name',
        'occupation',
        'town',
        'county',
        'image',
    ]
```

Figure 55: Edit Tradesperson Fields

A consumer also has the same ability to edit their details, with their details being retrieved from the database as shown in figure 56. This allows their newly submitted details to be error checked in the same manner as they were when registering. Here, consumers are also provided with the functionality to change their password, with their old password being compared to their current stored password and their new password being error checked in a similar manner to the registration page. Following this error checking, their newly submitted password is stored in the application's database.

```
class Meta:
    model = User
    fields = [
        'username',
        'email',
        'first_name',
        'last_name',
        'password'
    ]
```

Figure 56: Edit Consumer Details

Both sets of users are also provided with the ability to reset their password should they forget it. This is accomplished on the application's back-end by defining the 'PasswordResetView' and 'PasswordResetDoneView' in the application's urls. So that this email can be sent locally, the email back-end must be defined as shown in the appendix at

figure 97. For this to be implemented commercially, this configuration must be modified to point to a created SMTP server.

Defining both, the models and fields in the relevant forms enables the appropriate record to be retrieved from the database. This then allows the user to modify their desired attributes without the need for repopulating every associated field.

4.5.7 Favourites

There are three views in this application which combine to implement its favouriting functionality. These include views for adding, removing and displaying tradespeople.

Before displaying the current consumer's favourite tradespeople, first they must be retrieved from the application's Favourites model. Following this, they are then filtered so that only the current user's tradespeople gets returned as JSON to the relevant template. This is apparent in figure 57.

```
def favourites_template(request):
    favouriteList = Favourites.objects.all()
    favouriteList = favouriteList.filter(username=request.user)

    return render(request, "favourites.html", {"favouriteList": favouriteList})
```

Figure 57: View to Retrieve Current User's Favourited Tradespeople

Should a consumer decide to favourite a tradesperson, they will see their selected tradesperson's details sent to its corresponding view. This information is then retrieved using the 'request.POST' command as shown below.

```
def favourites(request):
    if request.is_ajax():
        t = request.POST.getlist('tradesperson')
        try:
            Favourites.objects.get(username=request.user, firstname=t[0], lastname=t[0])
        except Favourites.DoesNotExist:
            t = Favourites(username=request.user, firstname=t[0], lastname=t[1],
                           email=t[2], phone_no=t[3], company_name=t[4],
                           occupation=t[5], town=t[6], county=t[7], image=t[8])
            t.save()
        message = "Yes, AJAX!"
    else:
        message = "Not Ajax"
    return HttpResponse(message)
```

Figure 58: View to Favourite a Tradesperson

Following this, the Favourites model is queried to check if the tradesperson already exists for the current user. If this returns to be false, the tradesperson is then stored in corresponding favourites model. After this, the appropriate message is returned so that it can be logged.

When a consumer chooses to remove a tradesperson from their favourites, they are triggering a similar back-end implementation to occur. However, this time the first and last names of the tradesperson are retrieved so that they can query the Favourites model in conjunction with current user. Following this, their desired tradesperson is removed from the subsequent model using the ‘delete()’ method as seen in the appendix at figure 99. The template is then reloaded on the application’s front-end so that the consumer is displayed a visual representation of the functionality implemented.

Favouriting tradespeople is one of Tradespeople Near Me’s core functionalities and so was a key component of the application’s back-end development process. Implementing this service correctly ensured that only the consumer’s desired tradesperson would be added or removed from its corresponding model. It is for this reason, that it carried extreme importance with no margin for error.

4.5.8 Update Tradesperson's Location

When a tradesperson first registers with the application, they have their location stored via the previously described 'POST' AJAX request. This will involve calling the view in figure 57 so that the passed JSON attributes may be retrieved and added to the application's database.

```
@csrf_exempt
def update_tradesperson_location(request):
    if request.is_ajax():
        json_data = json.loads(request.body)
        tradesperson_location = json_data['data']

        lat = tradesperson_location[0]
        lng = tradesperson_location[1]
        point = Point(x=lng, y=lat)

        current_tradesperson = Tradesperson.objects.get(username=request.user)
        current_tradesperson.location = point
        current_tradesperson.save()
        message = "Yes, AJAX!"
    else:
        message = "Not Ajax"
    return HttpResponse(message)
```

Figure 59: Update Tradesperson's Location View

Here, it can be seen how the current tradesperson is acquired before having its location updated using the 'Point' variable. This variable was very difficult to implement as it required the data to be passed as JSON instead of the previously implemented strings. After the tradesperson's instance has been recalibrated with their current location, it is then saved to the database. This view is only called the first time the tradesperson logs in but can also can be executed when the tradesperson visits their edit details page via the provided button.

4.5.9 Searching

The search bar in the application's header executes a view which in turn queries the relevant tradespeople. This is achieved using the imported Q method as shown in the figure below.

```
def search(request):
    queryList = Tradesperson.objects.all()
    query = request.GET.get('q')
    queryList = queryList.filter(
        Q(occupation__iexact=query) |
        Q(firstname__iexact=query) |
        Q(lastname__iexact=query) |
        Q(town__iexact=query) |
        Q(county__iexact=query) |
        Q(company_name__iexact=query)
    ).distinct()

    return render(request, "search.html", {"queryList": queryList})
```

Figure 60: Search View

Here it can be seen how the attributes, occupation, first name, last name, town, county and company name get queried for the consumer's searched term. The 'iexact' extension then enables each Tradesperson attribute to be iterated over regardless of their uppercase or lowercase syntax. Following this, the filtered 'queryList' is returned as JSON to the application's search template.

4.5.10 Mapping

In terms of the application's back-end, the implemented mapping functionality gets executed using a common view, with the template then handling it's returned data. This data being returned to the application's front-end can be seen in the figure below. Here, it can be seen how each Tradesperson gets serialized as GeoJSON before being returned to the application's front-end.

```
def tradesperson_data(request):
    tradespeople = serialize('geojson', Tradesperson.objects.all())
    return HttpResponse(tradespeople, content_type='json')
```

Figure 61: View returning all tradespeople

The application's back-end mapping functionality is quite concise thanks to the definitive Tradesperson model outlined previously. Therefore, the complexity of the mapping services arises from the application's templates.

4.5.11 Conclusion

Upon reflection on the application's back-end, it is clear to see the numerous components needed to make up the desired program. This is shown through the development of the application's various models and their corresponding functionalities outlined in this section. Implementing these back-end functionalities carried huge significance in the development of this project as each template returned to the user mainly consists of data being retrieved from the application's models. Therefore, the successful execution of this part of the project was vital to ensuring the data being displayed to the user was correct.

4.6 Deployment

4.6.1 Introduction

This project will be deployed using the chosen technologies, Digital Ocean and Docker. The following section will discuss how these features were initially integrated into the application as well as how they were added to whenever the project's functionalities began to get more complex.

4.6.2 Docker

Docker is "the only independent container platform that enables organizations to seamlessly build, share and run any application, anywhere" (53). Docker allows developers to package their code and its dependencies in the form of a docker container. A Docker container is simply an instance of a docker image. Implementing Docker as part of this application enables Tradespeople Near Me to run reliably from one computing environment to another. The containers created as part of this application can be seen on a virtual machine using the 'docker ps' command, as shown in the figure below.

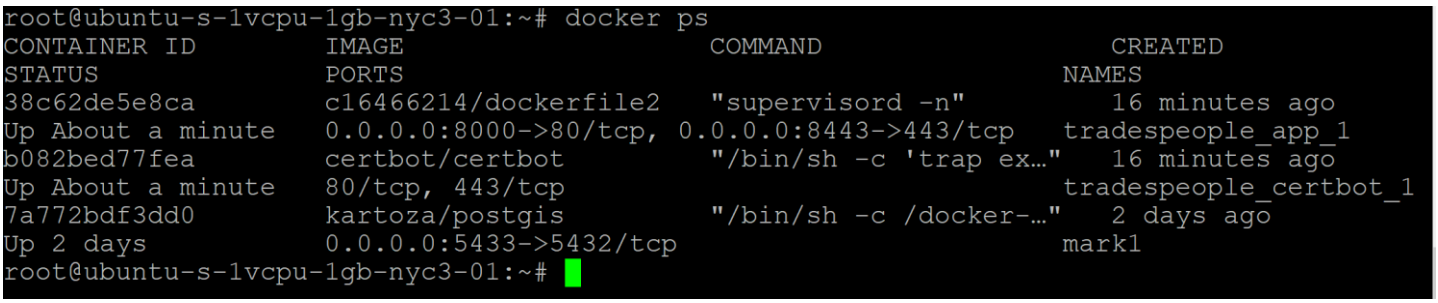


Figure 62: Deployed Docker images

In order to run a live Django application, the test environment must build a docker image from a dockerfile. The sequence at which this dockerfile is compiled is shown in the appendix at figure 98. This Dockerfile then enables the required files shown at figure 19 in the early development, to connect to one another. Thus, allowing the overall application to be packaged and ran as a single docker image. After doing this correctly, Docker push must then be executed so that the newest image can be pushed to the specified Docker Hub account. The next step in deploying a Docker container is to pull the latest Docker image from Docker Hub on a virtual machine. This process is significantly more complex without the use of Windows 10 Pro as Docker Toolbox must then be incorporated as part of the application's technologies. Following this, Docker compose is used to stop and restart the Docker container when needed.

4.6.3 Digital Ocean

Digital Ocean is this application's cloud provider and along with Docker was incorporated into the early development stages of the project. Upon obtaining a domain name, a droplet was created with a specified IP address. This IP address is necessary for obtaining the required SSL certificate. Digital Ocean's servers only use solid state disks, making its droplet a very efficient choice for this application. The created droplet for this application can be seen in below in figure 63.

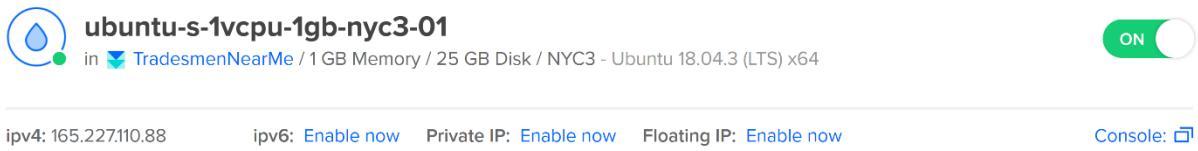


Figure 63: Digital Ocean Droplet

Digital Ocean aims to be designed for developers, enabling them to "build more and spend less time managing your infrastructure with our easy-to-use control panel and API" (54). The droplet used for this application consists of an Ubuntu virtual machine and allows the developer to gain access to the server's console. Tradespeople Near Me is connected to the acquired domain and droplet via the application's settings. Declaring these attributes is compulsory for any live Django application and can be seen in figure 64.

```
'MY_DOMAIN_NAME': 'thearononeill.club',
'DOCKER_IMAGE': 'c16466214/dockerfile2',
'DOCKER_COMPOSE_FILE': 'docker-compose.yml',
'NGINX_CONF': 'django_nginx.conf',
'SECRET_KEY': 'p03f5z5l(6%43!zo4((j@#g2zfg=uo*tsn-okzsl6se7zpor8=',
'DATABASES': {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'gis',
        'HOST': '165.227.110.88',
        'PORT': '5433',
        'USER': 'docker',
        'PASSWORD': 'docker',
    }
},
'ALLOWED_HOSTNAMES': [
    'DESKTOP-0LO4DML',
],
```

Figure 64: Domain & Droplet Configuration

Here, it can be also seen how the allowed hostname is defined so that development is not hindered on a local machine. Not only this but the created 'gis' database is configured so that the docker image can connect successfully with the application's Postgres database. Therefore, duplicating the previously created database so that it can run in parallel with the application's docker environment.

4.6.4 Conclusion

Despite the application's back-end features requiring significant development time, they had to be developed simultaneously with the project's deployment so that the overall goal of a running Django application could be achieved. With Tradespeople Near Me's development now complete, the subsequent section will cover the various testing methods carried out on the application.

5. Testing and Evaluation

5.1. Introduction

Testing is a vital part of any software development project, with various testing techniques used throughout the development cycle of this application. Implementing testing is hugely beneficial for pinpointing defects in software so that the appropriate corrections can be made. This can therefore prevent any potential critical errors from occurring. The testing techniques carried out on this project include white box and black box testing.

5.2. Testing

5.2.1 White Box Testing – Unit Testing

One form of testing carried out was white box testing, and more specially, unit testing. Unit testing has been performed on each developed component of this application. These tests ensure components are not throwing unexpected errors. “Unit testing helps identify a majority of bugs, early in the software development lifecycle” (55). Bugs identified at this early stage are easier to fix and therefore improve the project’s overall development time.

In this project, unit testing was implemented for each separate application functionality as shown in the project’s directory. Within each of these applications, there are subdirectories containing the various tests for each the different urls, models, views and forms. The directory structure for each application’s testing is shown in the figure 65. The first type of unit testing executed during the development of this project was for the application’s various models. This involved creating a model to validate each attribute associated with it. Each model implemented in this project has a corresponding test, with the figure below representing that of the tradesperson model.

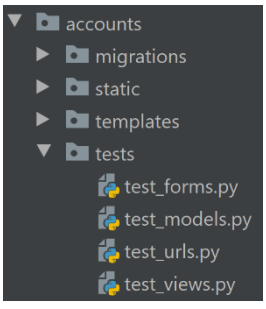


Figure 65: Testing Directory Structure

```
class TestModels(TestCase):
    def test_tradesperson(self):
        g = geocoder.ip('me')
        self.tradesperson1 = Tradesperson.objects.create(
            username='tradesperson1',
            password='tradesperson1',
            firstname='John',
            lastname='Doe',
            email='johndoe@gmail.com',
            working_hours='9am - 6pm',
            travel_distance=40,
            phone_no='082 392 1124',
            website_url='johndoe@gmail.com',
            company_name='John Doe Electrics',
            occupation='Electrician',
            premium='False',
            town='Glaslough',
            county='Monaghan',
            image='j',
            location=Point(x=g.lng, y=g.lat)
        )
```

Figure 66: Testing Accounts' Tradesperson Model

The tradesperson model is the most complex model created in this project and therefore carries a greater significance than that of any other. Here, it can be seen how each defined attribute is assigned an appropriate value for the purpose of the test. The current IP address of the device is also used to test this code so that coordinates can be obtained and assigned to the tradesperson's location attribute. Thus, enabling the test to pass without any error. Should any of these attributes be left out, the test will throw an error.

The next tests to be carried out on the system involve it's urls. This process involves comparing each url name with that of its corresponding view so that the returned data can be verified. Testing each url allows each view to be called in confidence that the data being returned is correct. The project's searching application is responsible for the retrieval of stored data so that it can be displayed to the necessary templates. It is therefore the ideal application to document as it is integral to the project displaying the correct tradespeople data to the consumer. The figure below shows how each url successfully retrieves every type of tradesperson listing.

```python
class TestUrls(SimpleTestCase):

    def test_all_tradespeople_url_is_resolved(self):
        url = reverse('getAllTradespeople')
        self.assertEquals(resolve(url).func, getAllTradespeople)

    def test_latest_tradespeople_url_is_resolved(self):
        url = reverse('getAllLatestTradespeople')
        self.assertEquals(resolve(url).func, getAllLatestTradespeople)

    def test_premium_tradespeople_url_is_resolved(self):
        url = reverse('getAllPremiumTradespeople')
        self.assertEquals(resolve(url).func, getAllPremiumTradespeople)
```

Figure 67: Testing Searching Urls

Unit tests were also written for this project's views so that its numerous templates can be examined for any potential bugs. This involves comparing each application's url names, with the corresponding template being returning by their view. Executing these unit tests enables the developer to trust that calling the url name will return its desired template. Figure 68 below shows how the favourites application tests it's views so that the 'index.html' and 'favourites.html' template files are returned from their corresponding name being called.

```python
class TestViews(TestCase):

    def setUp(self):
        self.client = Client()
        self.base_url = reverse('base_layout')
        self.favourites_url = reverse('favourites_template')

    def test_main_base_view_GET(self):
        response = self.client.get(self.base_url)
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'index.html')

    def test_favourites_view_GET(self):
        response = self.client.get(self.favourites_url)
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'favourites.html')
```

Figure 68: Testing Favourites Views

In the case of its the favourites template, it achieves this by implementing the following steps. First, its url name ‘favourites_template’ is reversed so that it can be assigned to the variable, ‘favourites_url’. A client is then used to simulate a user searching for the url, with its response status then being compared to that of 200, meaning ‘OK’. After this line has confirmed that it is in fact a valid url, it is then compared to ‘favourites.html’. Executing this allows the test to verify that its returned HTML page matches its desired template. This is carried out for each template implemented in this project.

The last unit testing integrated into this project is for its forms which are found in the application, accounts. There are three forms within this application, with the test shown below covering the consumer registration form. The following test involves populating each corresponding form field with mock data so that it can be verified. After assigning each field with a relevant value, the created form is then validated using the ‘form.is_valid’ method. The next form is created so that each of the fields created in the previous form can be deemed as necessary. It successfully proves this by creating an empty form and checking if the number of errors given is equal to the number of fields provided. This extra test is key to ensuring that each one of the fields are required.

```
class TestForms(TestCase):
    def test_user_register_form_valid_data(self):
        form = GlobalUserRegisterForm(data={
            'username': 'Johndoe20',
            'password': 'Johndoe2020',
            'password2': 'Johndoe2020',
            'firstname': 'John',
            'lastname': 'Doe',
            'email': 'johndoe@gmail.com',
            'email2': 'johndoe@gmail.com',
        })
        self.assertTrue(form.is_valid())

    def test_user_register_form_no_data(self):
        form = GlobalUserRegisterForm(data={})
        self.assertFalse(form.is_valid())
        self.assertEquals(len(form.errors), 7)
```

Figure 69: Testing Accounts Forms

After the application has ran each its tests, the outcome of each gets displayed to the console. As there are many separate applications within this project, it is vital that these

tests are executed for each of them. A sample output of these tests can be seen from running the accounts application tests. This is shown the following figure.

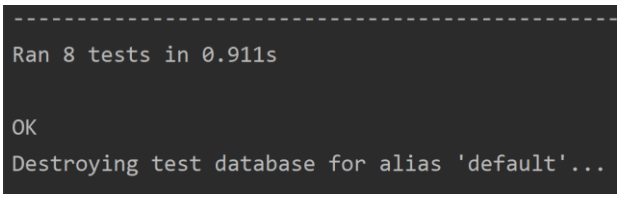


Figure 70: Accounts Unit Testing Results

5.2.1 Black Box Testing – Functional Testing

Black box testing is another form of testing that is implemented in this project. The black box testing carried out during the project's early development centred around API testing. However, as the application's functionalities evolved so did the type of black box testing being implemented. Therefore, functional testing was integrated into the project so that features could be tested against their functional requirements. Unlike unit testing, "this type of testing is not concerned with how processing occurs, but rather, with the results of processing" (56). These tests have been focused on the returning of the correct data from the necessary endpoints. The purpose of these tests is to ensure complete accuracy when returning the desired data requested by the user.

The functional tests executed within this project were composed using the tool called Selenium. Selenium is an ideal tool for this web application as it is an "open-source project for in-browser testing, originally developed by 'ThoughtWorks' and now boasting an active community of developers and users" (57). Selenium tests are executed on this project so that they can validate a range of functionalities. This is achieved using a chrome driver application to simulate the web application being opened, with this being shown in the appendix at figure 100. These tests will now be discussed in the following section.

The first functionality test to be executed in this project verifies if a user can see the displayed tradesperson listings. It carries this out by retrieving the listings element by class

name before comparing its text with the defined ‘Premium Listings’ as shown in the figure below.

```
def test_no_project_alert_is_displayed(self):
    self.browser.get(self.live_server_url)

    # When the user first requests the home page
    alert = self.browser.find_element_by_class_name('listings')
    self.assertEquals(
        alert.find_element_by_tag_name('h2').text,
        'Premium Listings'
    )
```

Figure 71: Premium Listings Selenium Test

The next test carried out was on the application’s favourites link provided to the user in the navigation bar. This test defined below once again creates a live server so that both, the favourites template and its corresponding button event can be retrieved. Following this, both values are then compared using the ‘assertEquals()’ method. Therefore, verifying that they produce the same result.

```
def test_no_project_alert_button_redirects_to_favourites_page(self):
    self.browser.get(self.live_server_url)

    # When the user clicks the favourite link
    favourites_url = self.live_server_url + reverse('favourites_template')
    self.browser.find_element_by_id('hover1').click()
    self.assertEquals(
        self.browser.current_url,
        favourites_url
    )
```

Figure 72: Favourites Tab Selenium Test

The most important functional test implemented in this project was to prove that a consumer can view a newly created tradesperson. In order to achieve this, a tradesperson was created with its relevant attributes, as seen in figure 73. The command ‘time.sleep(3)’ is then called so that browser is given time to load this newly created data. After this, the created class ‘service-heading’ has its text compared with the company name attribute, ‘John Doe Electrics’. As this test returns true, it confirms that the created tradesperson was displayed to the consumer as intended.

```
def test_consumer_sees_tradespeople(self):
    g = geocoder.ip('me')
    Tradesperson.objects.create(
        username='tradesperson1', password='tradesperson1',
        firstname='John', lastname='Doe', email='johndoe@gmail.com',
        working_hours='9am - 6pm', travel_distance=40,
        phone_no='082 392 1124', website_url='johndoe@gmail.com',
        company_name='John Doe Electrics', occupation='Electrician',
        premium='False', town='Glaslough', county='Monaghan',
        image='j', location=Point(x=g.lng, y=g.lat)
    )
    self.browser.get(self.live_server_url)
    time.sleep(3)
    self.assertEquals(
        self.browser.find_element_by_class_name('service-heading').text,
        'John Doe Electrics'
    )
```

Figure 73: Consumer Can See New Tradesperson Selenium Test

The last selenium test executed on this application was to verify that a user can login and hence make use of its favouriting functionality. The figure above illustrates how a sample user is created with the username, 'testuser' and the password '12345'. This user is subsequently saved before being logged in using the built-in 'Client()' method.

```
def test_consumer_can_login(self):
    user = User.objects.create(username='testuser')
    user.set_password('12345')
    user.save()

    c = Client()
    c.login(username='testuser', password='12345')
```

Figure 74: Consumer Can Login Selenium Test

After these tests have been implemented, their outcomes are logged to the application console as shown below. Notice how their runtime is significantly longer than the unit testing described above. This is since each test requires a separate browser window to be opened so that any bug encountered can be associated with its particular functionality.

```
Ran 4 tests in 37.705s

OK
Destroying test database for alias 'default'...
```

Figure 75: Selenium Testing Results

5.3. Evaluation

Before determining this application's type of evaluation, one must select the components that should be evaluated. Once these components were chosen, the following research into the various evaluation types was carried out. These include formative evaluation, process evaluation and outcome evaluation.

5.3.1 Formative Evaluation

Formative evaluation is implemented during the development of a new program or when an existing program is modified. It "ensures that a program or program activity is feasible, appropriate, and acceptable before it is fully implemented" (58). Formative evaluation enables changes to be made to the product before starting the final stages of the development cycle. It was therefore carried out after the first prototype of the application was produced. This was achieved by providing a sample user base with a short assessment to see if they could perform simple tasks on the application. Implementing this type of formative evaluation early on in this project enabled existing functionalities to be modified before being added to. Thus, helping to further improve the quality of the product.

5.3.2 Process Evaluation

Process Evaluation was carried out throughout the life cycle of this project as it "focuses on the implementation process and attempts to determine how successfully the project followed the strategy laid out in the logic mode" (59). It was conducted by assessing the numerous activities and components of the application so that these elements may be tracked. It is related to formative testing in that it poses questions to check whether each feature operates as expected. A sample of the questions created during the development of this project include:

1. Are the desired tradespeople returned from the view?

   ○ Yes ○ No

2. Are the correct tradespeople displayed to the template?

   ○ Yes ○ No

3. Does a created consumer get added to the User model?

   ○ Yes ○ No

4. Does a created tradesperson get added to the Tradesperson model?

   ○ Yes ○ No

5.3.3 Outcome Evaluation

The final evaluation carried out on this project was outcome evaluation. “Outcome Evaluation measures program effects in the target population by assessing the progress in the outcomes that the program is to address” (58). Its process of evaluation begins as soon as one person or group in the target population has interacted with the program. Outcome Evaluation clearly outlines how the program effects the target population’s behaviour while also communicating how effective the program is in meeting, both its deadlines and objectives.

The outcome evaluation carried out on this project followed Nielsen’s Heuristics. It is an ideal method as it successfully finds “usability problems in a user interface design so that they can be attended to as part of an iterative design process” (60). As there are two separate types of users for this application, the evaluation must be implemented from the perspective of both, consumers and tradespeople. Due to the current pandemic, the majority of feedback was provided via email or video chat, with the exception of a few close relatives whom provided face to face feedback. In addition to this, the sampled user base was quite small in comparison to the project’s initial plans. Each participate was able to

interact with the system thanks to it being deployed using Docker, as described previously. Consumers were the first user base to be provided with the evaluation form shown below.

Consumer Usability Evaluation Form

1. What device was used to access Tradespeople Near Me?

○ Mobile Phone ○ Laptop/Computer ○ Tablet/iPad

2. Have you ever required a tradesperson?

○ Yes ○ No

3. If yes, how did you find the tradesperson?

○ Through a friend ○ Through a relative ○ Through a ○ Other

4. Did you find the application useful?

○ Yes ○ No

5. How useful did you think the system is for finding a tradesperson?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

6. How easy was it to use the application's mapping functionality?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

7. How useful do you think it is to save your favourited tradespeople?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

8. Would you use Tradespeople Near Me given the opportunity?

○ Yes ○ No

Unfortunately, given the circumstances, it proved difficult to obtain feedback from reputable tradespeople. Nonetheless, a relative whom employs tradespeople as part of his business was asked for his opinion on the application. As well as this, he was able to ask two of his colleges for their opinion on the system. The evaluation form provided to them was as follows.

Tradesperson Usability Evaluation Form

1. What device was used to access Tradespeople Near Me?

○ Mobile Phone ○ Laptop/Computer ○ Tablet/iPad

2. Were you able to successfully sign up as a tradesperson?

○ Yes ○ No

3. How useful do you think creating a tradesperson's portfolio is?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

4. Would you provide the application with access to your location?

○ Yes ○ No

5. Were you able to change your tradesperson profile details?

○ Yes ○ No

6. Did you find the application useful?

○ Yes ○ No

7. How useful would you find the application for a tradesperson?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

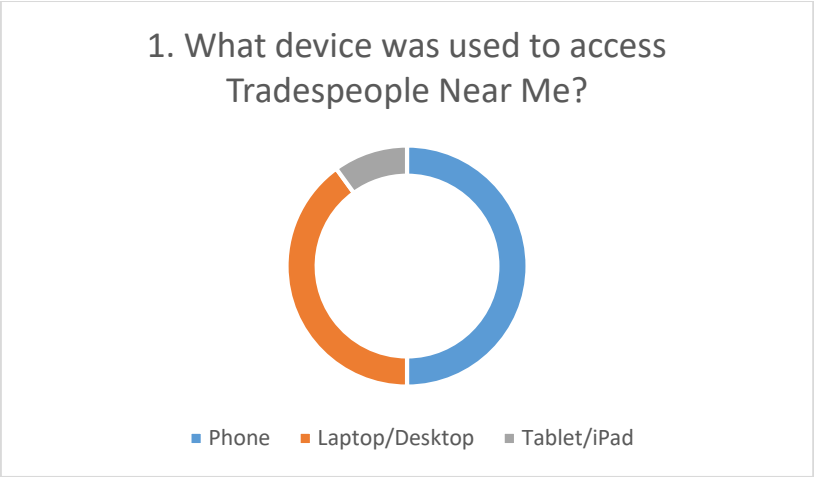
8. Would you use Tradespeople Near Me given the opportunity?

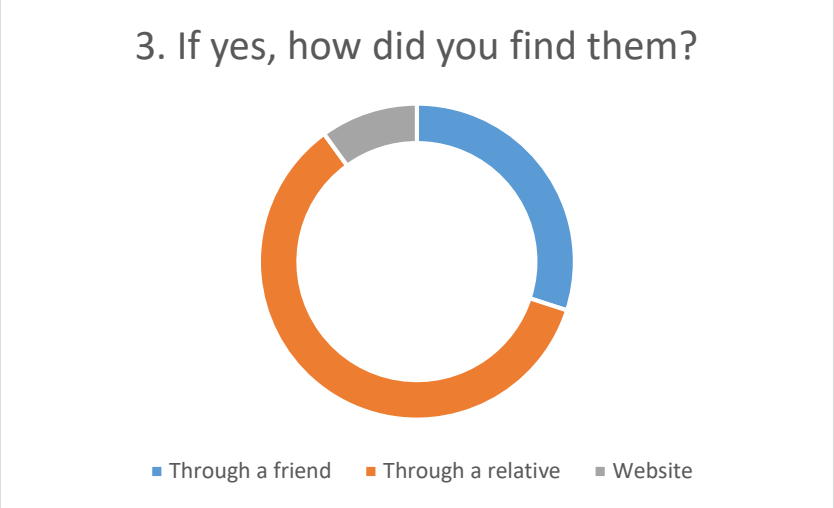
○ Yes ○ No

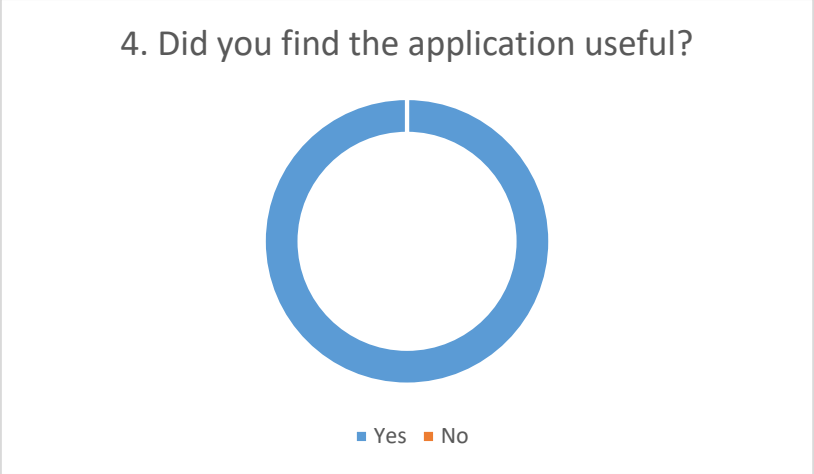
5.3.4 User Evaluation Feedback

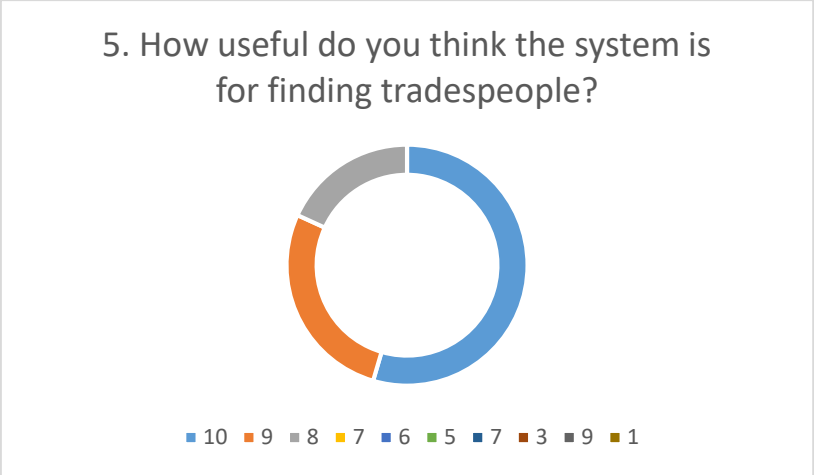
The user evaluation feedback is again categorised into two groups with both types of users viewing the application from two different outlooks. Due to the unprecedented situation currently being experienced, both sets of user groups are smaller than desired. The consumer feedback shown below therefore consists of results from ten potential consumers, while the feedback is from three tradespeople within the construction industry.

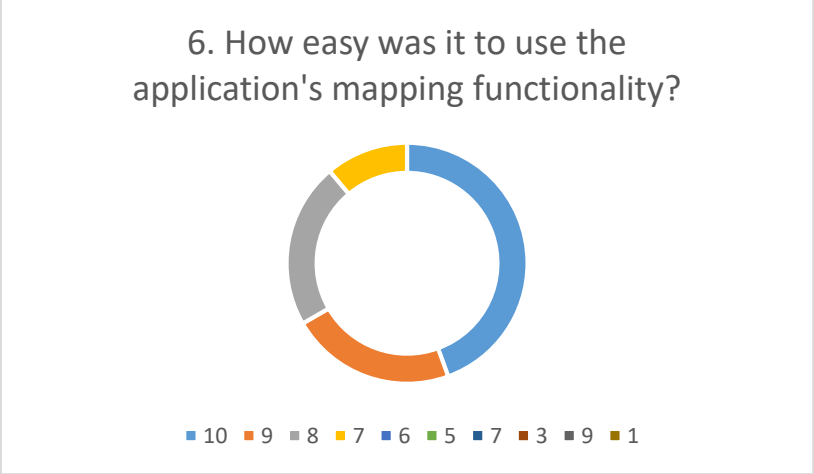
Consumer Usability Evaluation Feedback (10 responses)

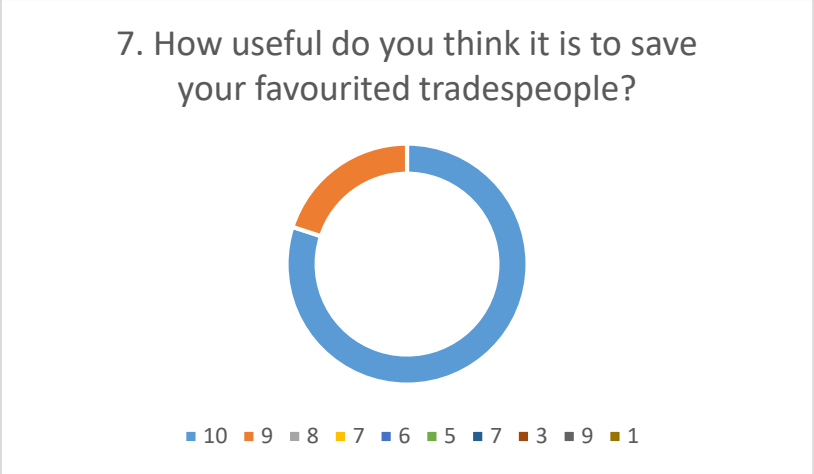


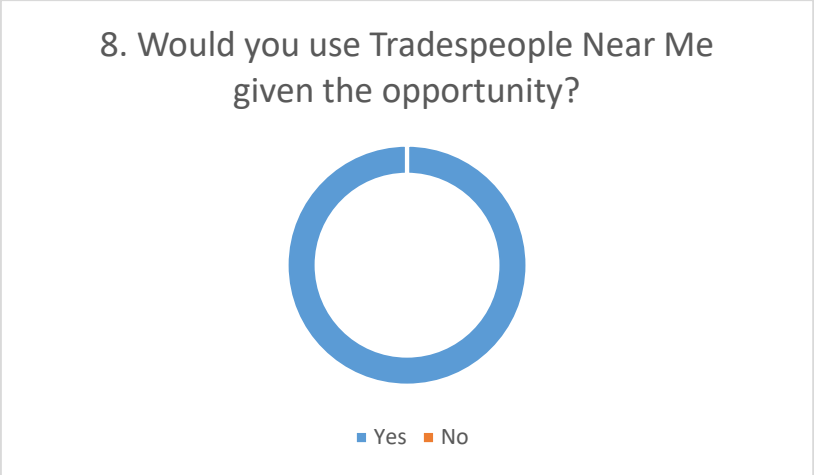



4. Did you find the application useful?
Yes
No

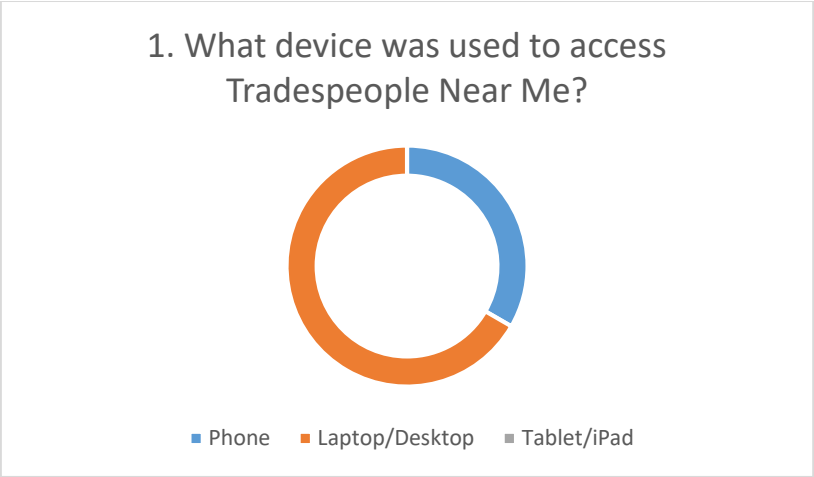

5. How useful do you think the system is for finding tradespeople?
10
9
8
7
6
5
7
3
9
1

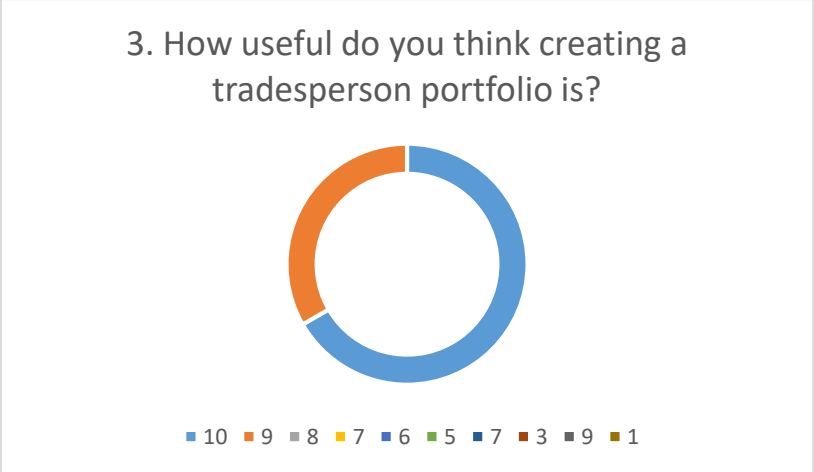

6. How easy was it to use the application's mapping functionality?
10
9
8
7
6
5
7
3
9
1

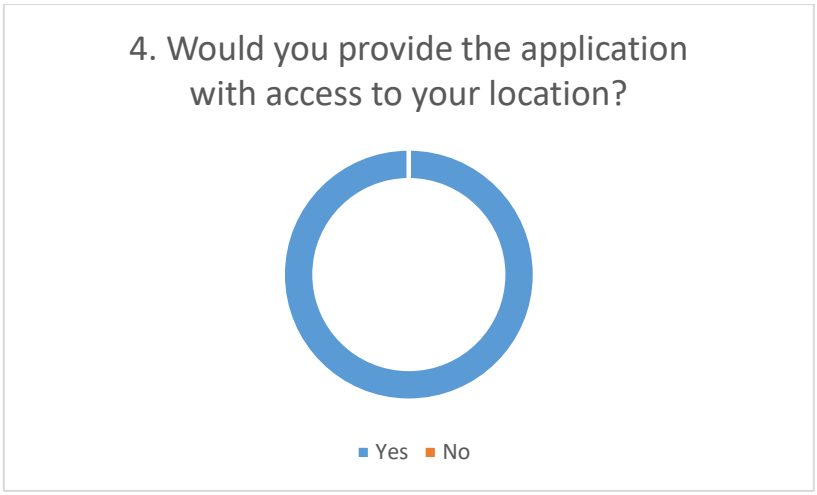

7. How useful do you think it is to save your favourited tradespeople?
10
9
8
7
6
5
7
3
9
1

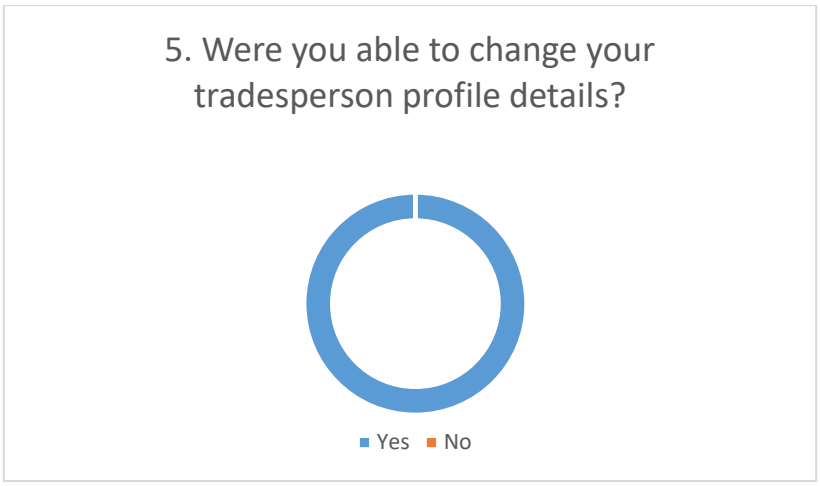

8. Would you use Tradespeople Near Me given the opportunity?
Yes
No

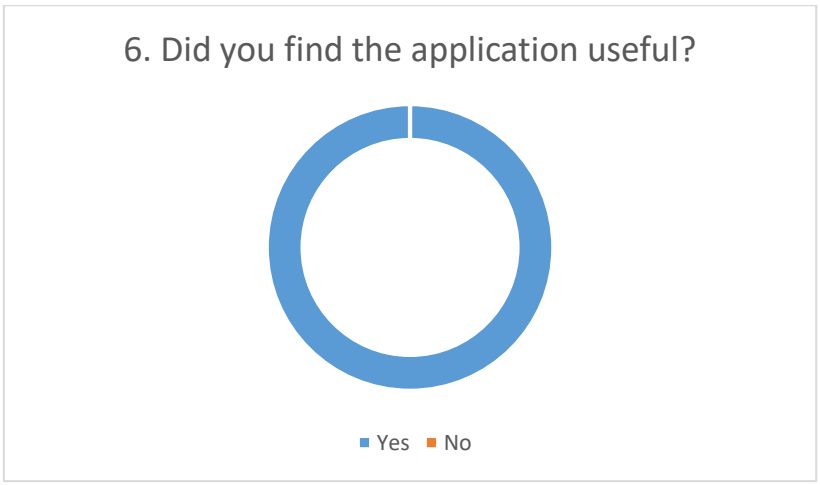
Tradesperson Usability Evaluation Feedback (3 responses)

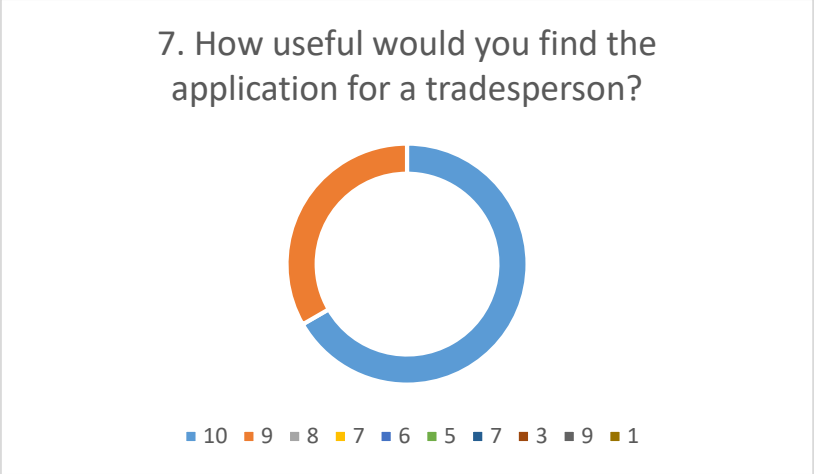


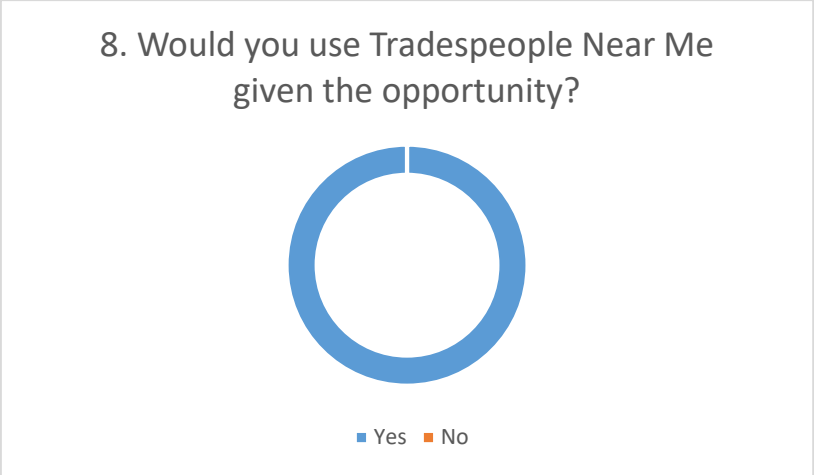



4. Would you provide the application with access to your location?
Yes
No


5. Were you able to change your tradesperson profile details?
Yes
No


6. Did you find the application useful?
Yes
No





From the user feedback shown above, it can be concluded that the application is extremely useful to both, consumers and tradespeople. The feedback provided from consumers also coincides with the previous research carried out in the early stages of this project. This can be noted in question three whereby very few consumers were able to rely on a website to provide them with adequate tradespeople in their area. The concluded question posed to both users also shows how Tradespeople Near Me will have a significant demand when it is made commercially available.

5.4. Conclusions

The testing and evaluating carried out during this project were extremely important to ensuring that any potential bugs could be found as early on as possible in its development. The testing carried out on the application became more intense as the functionalities grew in complexity. Upon entering the final stages of development, a detailed plan for testing was outlined so that it could be proven that the application executes as intended. It was important that this phase was integrated into the project before it was deployed. Although, evaluating the application was initially targeted towards a wide user base, a sample user base had to be obtained due the current epidemic being experienced, with minimal contact being advised. Despite this, the importance of the results from each user whom interacted with the system cannot be underestimated. Therefore, it can be concluded that the deployed application is desirable for both, consumers and tradespeople.

6. Project Plan & Issues

6.1. Introduction

Substantial planning went into the designing and developing of this project, with numerous issues encountered throughout the course of its duration. Most of the complex issues experienced occurred during the initial development stages of the application. These issues included the use of Windows as the project's local environment as well as the various applications needed to be compiled for deployment with Docker. Both, the project plan and the issues faced are outlined in greater detail below.

6.2. Project Plan

The GANTT chart below depicts an accurate picture as to how much time and commitment was required over the previous nine months so that the application was complete on schedule in March.

The GANTT chart's objectives defined in the left-hand column emphasise the importance of research in this project. Not only this but it highlights the significance of testing and evaluating the application. Despite making a solid start to this project before Christmas, there was still a substantial amount of work yet to do to meet this application's required standards.

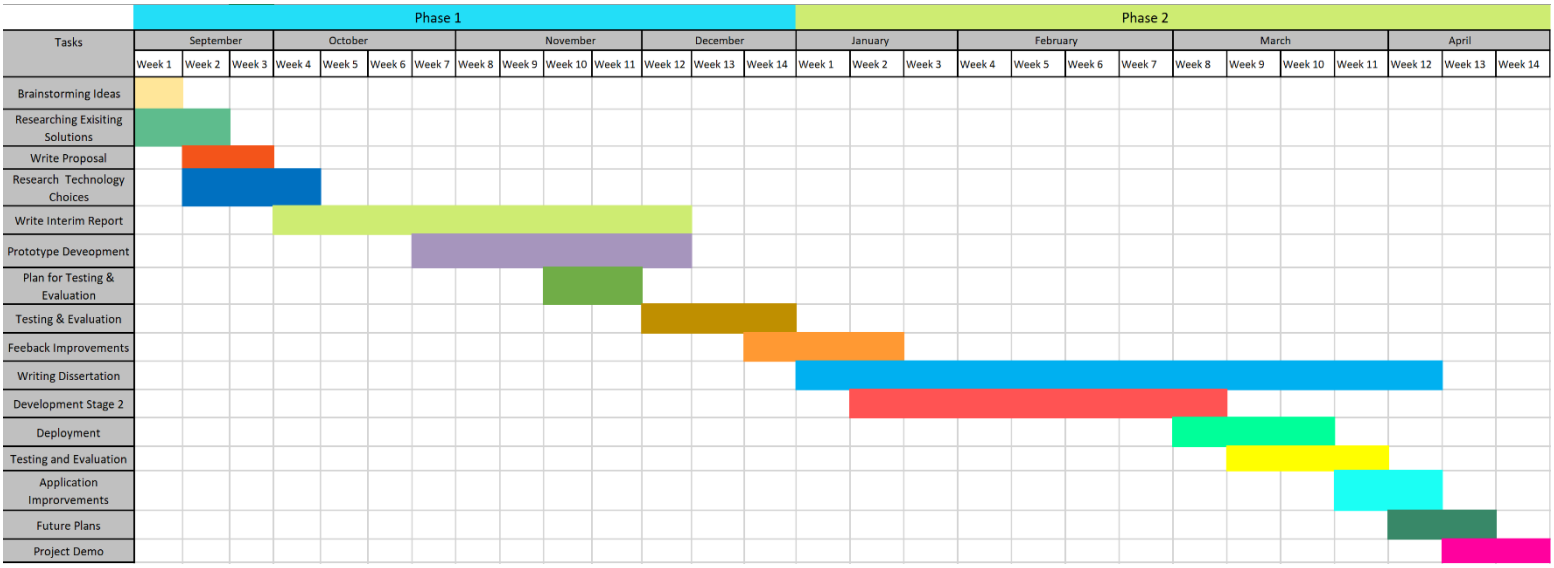


Figure 76: GANTT Chart

Each of these defined tasks got added to the project's Kanban board at various stages of the its development. There was an almost endless sprint feel to this project with functionalities being continuously appended to the board, each at their own stage of development. This is evident in the Kanban board shown below for the week starting on March 2nd.

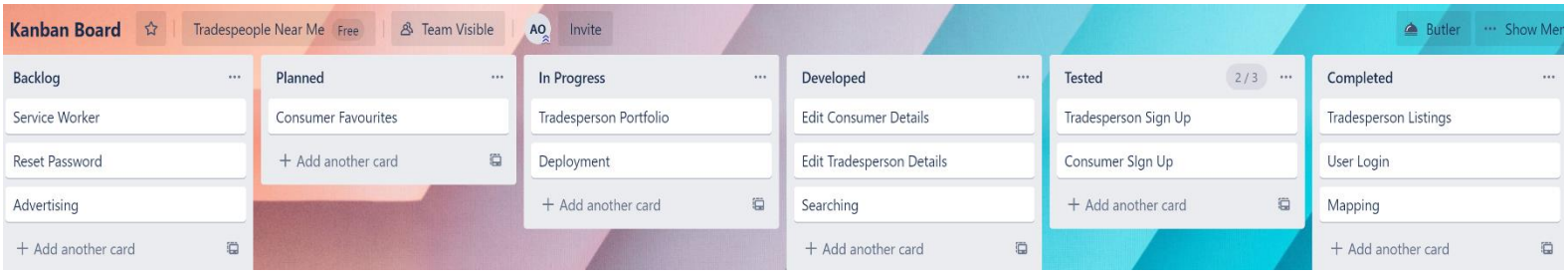


Figure 77: Tradespeople Near Me Kanban Board @ Week Starting March 2nd

6.3. Project Plan Review

The project plan was developed during the initial stages of the project and by in large was adhered to so that the application could be completed within the scheduled time frame.

This was achieved using a Kanban board as the chosen methodology so that multiple functionalities could be developed simultaneously. The time taken to complete the objectives shown for each week would often differ to what was planned, with some features overlapping others due to the errors encountered along the way. Thus, making a Kanban board an ideal choice to help implement the desired system. Every feature initially planned to be developed was successfully carried out by the end of the development cycle. The key to accomplishing this was setting weekly goals to meet, and then further dividing these goals into daily sprints. Subsequently, a combination of both, a Kanban board and daily sprints were integral to ensuring the project was completed by its set date.

The only problem experienced with following the project's plan was that additional features got added to the Kanban board which had not been initially accounted for. There were good intentions of completing these additional features but in the end, some of them had to be documented as the project's future work. Operating a Kanban board enabled numerous functionalities to be implemented simultaneously near the end of its development. This is obvious from figure 77 whereby numerous features where yet to move to the completed stage of the board. Despite committing precious time to developing the GANTT chart and maintaining the Kanban board, they proved to be invaluable commodities throughout the project's development cycle.

6.4. Issues Encountered

There was a multitude of issues encountered throughout the development of this project. These included the implementation of the application's technologies Postgres, Django, Leaflet and Docker. Although there were many more issues associated with each implemented functionality, these four proved to be the most difficult to overcome.

6.4.1 Postgres

During the early development stages of the application, an obscure problem was encountered when attempting to download PostgreSQL as the application's back-end

technology. It saw a single folder fail to download with Postgres' installation. This proved to be a rare problem caused by Windows and experienced by very few people previously. After losing substantial development time, the project was then migrated to Ubuntu using a VirtualBox and once here, Postgres downloaded seamlessly. However, after coming across another time-consuming issue, the project was once more moved back to Windows after the installation issue with Postgres was solved. It was eventually solved by assigning various permissions to a newly created user account and the downloaded file itself. Three weeks of developing time was lost due to this issue but thanks to the project's well managed version control, it was possible to switch operating systems. Even at the beginning of this project, the importance of version control became apparent. This was the most stressful, time consuming error faced during the project's development.

6.4.2 Django

Although there is enough documentation on Django for those implementing a simple web application, there is an insufficient amount relating to the implementation of anything remotely complex. Most of the online information found provided steps of how to carry out functionalities for its built-in user model but failed to expand on these for custom models. Therefore, implementing the features linked to the tradesperson and favourites models became rather difficult. In addition to this, there was ample documentation for performing 'GET' AJAX requests but was lacking for the more advanced 'POST' requests. This resulted in more critical thinking being required to solve problems than initially thought. The time taken to develop Tradespeople Near Me was significantly increased as a result.

6.4.3 Leaflet

There was notable amount of Leaflet knowledge required to develop the application's mapping functionalities. However, the extent of the understanding needed to implement the desired features was not considered during the planning of the project. The hardest of these features to implement proved to be the consumer filtered search which removed unwanted tradespeople from the map. This was eventually accomplished by analysing what exactly was needed to solve the problem. It saw each marker being categorised into a map

layer based on the occupation the corresponding tradesperson held. From here, each layer was able to be removed using an appropriate function.

6.4.4 Docker

Docker also proved to be a source of some issues during the deployment phase of this project. However, unlike the previous issues, this one was somewhat expected as docker involves packaging software in the form of a container so that it can run on any environment. The requirements.txt file found in the project's directory contains the various applications needed to deploy the same solution which can be run locally. Once compiled, there were numerous issues associated with these applications. After a substantial amount of trial and error, the version of application was found to be the cause of these errors. Therefore, the application was required to be uninstalled and replaced with a different version. This was the case for Django and Django-leaflet.

Another stumbling block during deployment was that Docker is designed to work with Windows 10 Pro rather than the universally obtained home edition. Acquiring Windows Pro costs €259 with Microsoft and so was not a viable option. As a workaround, Docker Toolbox was downloaded and configured along with Putty so that a Linux server was able to be ran. Once again, this issue was not envisioned during the planning stages of the project. Although it resulted in the deployment of the application becoming a more complex process than it needed to be, it was successfully overcome.

6.5. Conclusion

Tradespeople Near Me ultimately benefitted from the issues experienced as it enabled a significant amount of knowledge to be obtained on the application's various technologies, which quite simply would not have been gained from reading documentation found online. Despite some development setbacks, the issues encountered helped solve bugs found in the system in a more efficient manner than it would have done should these issues not have occurred.

7. Conclusion & Future Work

7.1. Future Work

With the testing and evaluation carried out successfully on the existing application, one must consider the additional features that could be implemented as part of its second iteration. Although there are a substantial number of features already added to this application, there are many more planned for future work. Therefore, this section will explore the various features that will be added in an additional release of the application.

7.1.1 Enhancing Existing Features

There are three main features that will be enhanced during the future work of this application. These include the adding of an additional filter, converting the tradesperson signup form to be a paid service and deploying a Simple Mail Transfer Protocol (STMP) to handle the resetting of a user's password.

Although, there are four filtering options available to consumers, an extra filter will be integrated into the system so that they can base their search on a tradesperson's working hours. This will better cater to those individuals who require a tradesperson at an obscure time of day for an urgent job. Those tradespeople who then decide to operate a 24-hour service are free to indicate that calls outside a set time range will result in an additional fee. Therefore, both parties will benefit from the implementing of this feature.

During the further development of this application, a subscription service will be incorporating into the tradesperson's signup form. This will be made possible from the already obtained SSL certificate which is needed to protect user transaction information. Converting this application into a paid service for tradespeople will generate a source of income as well as somewhat validating the authenticity of the tradespeople registering.

Although it is possible to acquire a free STMP server to handle emails being sent, more research is needed into the security of these services. Therefore, the best solution will most likely be a paid subscription as it is more likely provide a securer platform. Unfortunately, enhancing these features was not possible during the allocated time frame and so will be accommodated for in the application's future work

7.1.2 Smart Contract

The most exciting feature that will be added to Tradespeople Near Me is that of a smart contract between a consumer and a tradesperson. This will involve the incorporation of a decentralized contract application (dApp) created on the blockchain, with blockchain being a distributed database stored on multiple devices at the same time. The smart contracts themselves will be created using a similar language to JavaScript called Solidity. "Solidity is the widely used and most adopted language that can use to develop Smart Contracts in Ethereum" (61). Remix will be used as the contract application's IDE as it provides account addresses for contracts to be tested with.

After the appropriate dApp has been successfully created, it will be injected into Tradespeople Near Me using an API to read in the desired contract. This will be made possible by the contract address previously obtained from Remix. Following this, the web application's appropriate functions will be called so that the contract between consumer and tradesperson can be displayed to both parties. Implementing this component will then leave Tradespeople Near Me unaccountable for any potential damages caused. Although this will be a very unlikely scenario, it is best practice to safeguard against any potential pitfalls of the application.

7.1.3 Advertising

The plan for the next development stage of this application will see Tradespeople Near Me developed alongside another final year project called 'RealEChain'. This will see both

developers work alongside each other so that both applications can be launched as soon as possible.

RealEChain is a blockchain application that allows individuals to buy, sell and rent properties using Ethereum as the computing platform. Its target market is therefore closely related to Tradespeople Near Me and will subsequently enable both platforms to advertise each other's applications in a modern form of mutualism. Working alongside RealEChain will be extremely advantageous for this application as it will mean acquiring an individual who is well versed in the constantly evolving world of blockchain. Thus, significantly shortening the developing time required to implement the smart contract system described above. Tradespeople Near Me will also provide RealEChain's users with an added incentive to visit their platform as most new homeowners will also desire a tradesperson so that they can tailor their new property to their needs.

Both applications are in a similar stage of production and once complete will produce very lucrative products.

7.1.4 Review System

Another feature to be added to this application includes a review system whereby consumers can rate tradespeople whom they have previously employed. The main task associated with implementing this feature is the authenticity of the reviews posted.

To approve a consumer review for posting, one must validate that they have had work carried out by the claimed tradesperson. This functionality will therefore be implemented following the smart contract described previously so that this link may be proven. It will then take a similar form to the tradespeople's portfolio already designed so that consumers are given the opportunity to add a detailed review of the work carried out. Defining this review-based system as a model with a text editor attribute will enable consumers to

support their review with images of the work carried out. As well as this, consumers who post a review will see their username and a link to message them placed next to it.

However, the ability to message a consumer who has posted a review, will only be given to fellow consumers who have logged into the platform. Consequently, there will be an incentive given to consumers who chose to review a tradesperson. This could involve issuing a barcode to them which will in turn be given to their next tradesperson hired using this system. Thus, giving the consumer a discount off the tradesperson's work and the tradesperson a discount off their monthly subscription.

The current working application supplies consumers with ample information for each tradesperson so that they can be assured of the quality of work that will be delivered. However, implementing a review-based system will provide consumers with an even greater confidence that are hiring the right professional.

7.1.5 Messaging System

Tradespeople Near Me is designed to eliminate any worry associated with hiring an unknown tradesperson. For any individual whom is particularly conscious of this, the next stage of the application's development will involve the integration of a messaging system.

As mention above, this will see a messaging service provided to logged in consumers so that they can message fellow consumers whom have had work done by their desired tradesperson. This service is simply not available on any existing platform and will therefore be a significant attraction to worrisome consumers requiring a trustworthy tradesperson. More time will also be required to fine tune the deployed application as the development time had to be concentrated on the internal workings of the application's various features. Despite this, the project was successfully deployed using Docker, with this being the most complex aspect of deployment.

7.2. Conclusion

The following section will analyse the project's key components and knowledge gained as a result of their implementation.

7.2.1 Front-End

The front-end of this application was initially tricky to develop as it involved retrieving stored information before its main components could be dynamically created using JavaScript. Unlike most conventional web applications, most of this application's content displayed to the user is updated every time the page loads so that the most recently registered tradespeople are being retrieved from the database. This proved to be a somewhat unexpected source of complexity as it involved developing a full stack application before the user was able to view the relevant information. If it were feasible to repeat this project again, it would be wise to research the possibility of implementing Angular as its front-end technology to see if it would improve an already impressive response time. This being said, the front-end technologies incorporating into this project worked extremely well in producing the desired final product. This can be seen by comparing the high-fidelity prototype developed in October with that of the current working application.

7.2.2 Middle-Tier

There was a steep learning curve associated with the development of the application's middle-tier components. This was initially not thought to be the case when planning for its development and subsequently resulted in more time being committed than was first allocated. Despite this, it was accommodated for by overlapping features in the weeks following its implementation. The middle-tier of this project therefore benefitted from the Kanban methodology carried out throughout its duration. If the project were to be undertaken again, more time would have to be allocated for both, its research and development.

7.2.3 Back-End

This project's back-end worked very well after the bizarre issue associated with the download of Postgres had been overcome. Tradespeople Near Me was designed with a tight schedule in mind

and could ill afford to lose time to any complex issues experienced. Although, this could have been accounted for during its planning, it would have meant diminishing some of the application's existing features. However, this issue would not have been experienced with Linux as its operating system and should a similar project be undertaken, it would be developed using Ubuntu as its local environment.

7.2.4 Deployment

Deploying this application also proved to be quite difficult, with Windows once again being the root cause of any problems encountered. Docker uses a Linux based environment and so the project's settings had to be configured to run the application on both, Windows as the local machine, and Linux as the production environment. Not only this but having Windows 10 Home instead of the pro edition made the deployment unnecessarily more complex. However, despite these setbacks, Docker and Digital Ocean worked exceptionally well given the circumstances. Identical to its back-end, the only change that would be made to the project should it be repeated would be to switch to a Linux based local environment. Therefore, considerably shortening the development time needed to complete the project.

7.2.5 Final Statement

Tradespeople Near Me was chosen for this project so that it could combat an existing problem found in society today. Instead of working on an application solely for completing a final year project, Tradespeople Near Me was developed as a possible business venture. Therefore, reaping the rewards both, in college and as personal project. Should this be decided upon, Tradespeople Near Me will act as a very solid foundation in which to base a further release. The current application was developed within the space of seven months and is arguably already better than any similar existing platform. Therefore, to have the opportunity to further improve and advertise it over the coming months would see it grow into a suitable business. This would further justify the vast amount of time spent working on the project.

7.3 Resources

7.3.1 Demonstration Video

A brief demonstration of the various implemented features can be found at:

https://www.youtube.com/watch?v=dObqYSrQ8dM

7.3.2 GitHub Link

The files associated with the development of this project can be found at:

https://github.com/arononeill/geodjango_app/tree/master/tradesmen_near_me

7.3.3 Deployed Project

Although the deployed project does not match what was created locally, it was able to use Digital Ocean and Docker to deploy the application. This can is the hardest aspect of deployment as documented in the issues experienced.

The deployed project can be found at:

https://thearononeill.club:8443/

8. Bibliography

1. Reddan F. Rents in Dublin to rise by 5-6% a year and surpass €2,000 by 2019 [Internet]. The Irish Times. [cited 2019 Dec 7]. Available from: https://www.irishtimes.com/life-and-style/homes-and-property/rents-in-dublin-to-rise-by-5-6-a-year-and-surpass-2-000-by-2019-1.3314199

2. Savills. Number of people renting in Ireland at record high. 2017 [Internet]. [cited 2019 Mar 11]; Available from: https://www.rte.ie/news/business/2017/1204/924777-number-of-people-renting-at-record-high-savills/

3. About Us - Find Online Tradesmen [Internet]. [cited 2019 Nov 3]. Available from: https://tradesconnect.ie/page/about_us

4. Security Specialists | Dublin 24 | Dublin | Advance Access - Access Control, Automatic Barriers, Car Park Solutions, Revolving Door Providers | Trades Connect [Internet]. [cited 2019 Nov 3]. Available from: https://tradesconnect.ie/security-specialists/advance-access-access-control-automatic-barriers-car-park-solutions-revolving-door-providers_i720

5. Daft.ie : About us at Daft.ie - Ireland's No.1 Property Website [Internet]. [cited 2019 Nov 3]. Available from: https://www.daft.ie/about/

6. Property to rent in Dublin City Centre, Dublin City | Daft.ie [Internet]. [cited 2019 Nov 3]. Available from: https://www.daft.ie/dublin-city/residential-property-for-rent/dublin-city-centre/?s%5Bignored_agents%5D%5B0%5D=1551&searchSource=rental&offset=20

7. About Us [Internet]. Airbnb Newsroom. [cited 2019 Nov 3]. Available from: https://news.airbnb.com/about-us/

8. Node.js - Introduction - Tutorialspoint [Internet]. [cited 2019 Nov 4]. Available from: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

9. Benefits of Python over Other Programming Languages - Invensis Technologies [Internet]. [cited 2019 Nov 4]. Available from: https://www.invensis.net/blog/it/benefits-of-python-over-other-programming-languages/

10. 5 software frameworks advantages for web app development [Internet]. Syndicode - Ruby on Rails development agency. 2017 [cited 2019 Nov 9]. Available from: https://syndicode.com/2017/08/21/5-open-frameworks-advantages-for-web-app-development/

11. Mallawaarachchi V. 10 Common Software Architectural Patterns in a nutshell [Internet]. Medium. 2018 [cited 2019 Nov 9]. Available from: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

12. Sinhal A. MVC, MVP and MVVM Design Pattern [Internet]. Medium. 2017 [cited 2019 Nov 9]. Available from: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad

13. How to implement Design Pattern – Separation of concerns [Internet]. Default. [cited 2020 Mar 30]. Available from: https://www.castsoftware.com/blog/how-to-implement-design-pattern-separation-of-concerns

14. Architectural Guidelines to follow for MVP pattern in Android [Internet]. [cited 2019 Nov 11]. Available from: https://android.jlelse.eu/architectural-guidelines-to-follow-for-mvp-pattern-in-android-2374848a0157

15. The Top 10 Advantages Of Using Laravel PHP Framework [Internet]. Belitsoft. 16:50:00+00:00 [cited 2019 Nov 4]. Available from: https://belitsoft.com/blog/10-benefits-using-laravel-php-framework

16. Advantages and Disadvantages of Django | Blogs @ Mindfire Solutions [Internet]. [cited 2019 Nov 4]. Available from: http://www.mindfiresolutions.com/blog/2018/04/advantages-and-disadvantages-of-django/

17. AngularJS — Superheroic JavaScript MVW Framework [Internet]. [cited 2019 Nov 14]. Available from: https://angularjs.org/

18. Tutorial: Intro to React – React [Internet]. [cited 2019 Nov 14]. Available from: https://reactjs.org/tutorial/tutorial.html

19. What is Client-Side Scripting? Choosing the Scripting Languages for your Web Application [Internet]. Hiring Headquarters. 2015 [cited 2019 Nov 14]. Available from: https://www.upwork.com/hiring/development/how-scripting-languages-work/

20. Angular - The RxJS library [Internet]. [cited 2019 Dec 9]. Available from: https://angular.io/guide/rx-library

21. Database Concepts [Internet]. [cited 2019 Nov 11]. Available from: https://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm

22. Advantages of PostgreSQL ⋆ Bitnine Global Inc. [Internet]. [cited 2019 Nov 3]. Available from: https://bitnine.net/blog-postgresql/advantages-of-postgresql/

23. What is AWS [Internet]. Amazon Web Services, Inc. [cited 2019 Dec 7]. Available from: https://aws.amazon.com/what-is-aws/

24. Google Cloud Platform Overview | Overview | Google Cloud [Internet]. [cited 2019 Nov 11]. Available from: https://cloud.google.com/docs/overview/

25. Managed Databases (DBaaS) on DigitalOcean [Internet]. DigitalOcean. [cited 2019 Nov 11]. Available from: https://www.digitalocean.com/products/managed-databases/

26. 5 Reasons Why Resource Management Is Important | Ganttic - Ganttic [Internet]. Resource Planning Software | Ganttic. [cited 2020 Mar 18]. Available from: https://www.ganttic.com/blog/why-is-resource-management-important

27. Filtering UI: A Horizontal Toolbar Can Outperform the Traditional Sidebar - Articles - Baymard Institute [Internet]. [cited 2019 Nov 21]. Available from: https://baymard.com/blog/horizontal-filtering-sorting-design?source=post_page-----beea1798d64b----------

28. What is Elasticsearch | Elastic [Internet]. [cited 2019 Nov 23]. Available from: https://www.elastic.co/what-is/elasticsearch

29. AWS | Amazon CloudSearch - Search Service in the Cloud [Internet]. Amazon Web Services, Inc. [cited 2019 Nov 23]. Available from: https://aws.amazon.com/cloudsearch/

30. KRISHNA S. Code for a Cause: Geofence Technology for Priority Vehicles [Internet]. Medium. 2019 [cited 2019 Nov 23]. Available from: https://medium.com/progate/code-for-a-cause-geofence-technology-for-priority-vehicles-be0e018df0f0

31. Just How Accurate is GPS? - Microlise [Internet]. [cited 2019 Nov 27]. Available from: https://www.microlise.com/blog/just-how-accurate-is-gps/

32. Everything you ever wanted to know about HTML5 Geolocation Accuracy [Internet]. [cited 2020 Mar 30]. Available from: https://www.storelocatorwidgets.com/blogpost/20453/Everything_you_ever_wanted_to_know_about_HTML5_Geolocation_Accuracy

33. Geolocation getCurrentPosition() API - Tutorialspoint [Internet]. [cited 2020 Mar 30]. Available from: https://www.tutorialspoint.com/html5/geolocation_getcurrentposition.htm

34. Mobile location data is accurate up to 30 meters: report | Mobile Marketer [Internet]. [cited 2020 Mar 30]. Available from: https://www.mobilemarketer.com/ex/mobilemarketer/cms/news/research/22928.html

35. Haversine formula to find distance between two points on a sphere [Internet]. GeeksforGeeks. 2018 [cited 2020 Mar 30]. Available from: https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/

36. says A. Haversine formula [Internet]. MrReid.org. 2011 [cited 2020 Mar 30]. Available from: http://wordpress.mrreid.org/2011/12/20/haversine-formula/

37. Robusto CC. The Cosine-Haversine Formula. Am Math Mon. 1957;64(1):38–40.

38. Mapbox. Fast geodesic approximations with Cheap Ruler [Internet]. Medium. 2017 [cited 2020 Apr 3]. Available from: https://blog.mapbox.com/fast-geodesic-approximations-with-cheap-ruler-106f229ad016

39. Leaflet — an open-source JavaScript library for interactive maps [Internet]. [cited 2019 Nov 24]. Available from: https://leafletjs.com/

40. Advantages of Bootstrap framework | Vmoksha [Internet]. [cited 2019 Nov 24]. Available from: https://vmokshagroup.com/blog/bootstrap-advantages/

41. Nath DS. 4 important points to know about Progressive Web Apps (PWA) [Internet]. Medium. 2018 [cited 2019 Nov 24]. Available from: https://medium.com/@deepusnath/4-points-to-keep-in-mind-before-introducing-progressive-web-apps-pwa-to-your-team-8dc66bcf6011

42. Agile Vs Scrum: Know the Difference [Internet]. [cited 2019 Dec 5]. Available from: https://www.guru99.com/agile-vs-scrum.html

43. What Is Scrum Methodology? [Internet]. [cited 2019 Dec 5]. Available from: https://resources.collab.net/agile-101/what-is-scrum

44. What Is Kanban? An Introduction to Kanban Methodology [Internet]. [cited 2019 Dec 5]. Available from: https://resources.collab.net/agile-101/what-is-kanban

45. What is Agile Kanban Methodology? Learn the Methods & Tools [Internet]. [cited 2019 Dec 5]. Available from: https://www.inflectra.com/methodologies/kanban.aspx

46. Avraham SB. What is REST — A Simple Explanation for Beginners, Part 1: Introduction [Internet]. Medium. 2017 [cited 2019 Nov 25]. Available from: https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-1-introduction-b4a072f8740f

47. MVC Architecture [Internet]. [cited 2019 Nov 17]. Available from: https://www.tutorialsteacher.com/mvc/mvc-architecture

48. Ionic. Progressive Web Apps - Ionic Documentation [Internet]. Ionic Docs. [cited 2020 Mar 19]. Available from: https://ionicframework.com/docs/intro/what-are-progressive-web-apps

49. Forms have never been this crispy — django-crispy-forms 1.9.0 documentation [Internet]. [cited 2020 Mar 23]. Available from: https://django-crispy-forms.readthedocs.io/en/latest/

50. Introduction to Service Worker | Web [Internet]. Google Developers. [cited 2019 Dec 5]. Available from: https://developers.google.com/web/ilt/pwa/introduction-to-service-worker

51. Promise [Internet]. MDN Web Docs. [cited 2020 Mar 30]. Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

52. Soegaard M. Occam's Razor: The simplest solution is always the best [Internet]. The Interaction Design Foundation. [cited 2020 Mar 28]. Available from: https://www.interaction-design.org/literature/article/occam-s-razor-the-simplest-solution-is-always-the-best

53. Why Docker? [Internet]. Docker. [cited 2019 Dec 6]. Available from: https://www.docker.com/why-docker

54. DigitalOcean – The developer cloud [Internet]. DigitalOcean. [cited 2019 Dec 6]. Available from: https://www.digitalocean.com/

55. What is WHITE Box Testing? Techniques, Example, Types & Tools [Internet]. [cited 2019 Nov 26]. Available from: https://www.guru99.com/white-box-testing.html

56. Functional Testing [Internet]. Software Testing Fundamentals. 2012 [cited 2020 Apr 1]. Available from: http://softwaretestingfundamentals.com/functional-testing/

57. Holmes A, Kellogg M. Automating functional tests using Selenium. In: AGILE 2006 (AGILE'06). 2006. p. 6 pp. – 275.

58. Types of Evaluation. :2.

59. Process Evaluation [Internet]. [cited 2019 Dec 5]. Available from: http://www.uniteforsight.org/evaluation-course/module5#_ftn1

60. Experience WL in R-BU. Heuristic Evaluation: How-To: Article by Jakob Nielsen [Internet]. Nielsen Norman Group. [cited 2020 Apr 2]. Available from: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/

61. Walpita P. Ethereum Smart Contract Development with a Web App [Internet]. Medium. 2020 [cited 2020 Apr 4]. Available from: https://medium.com/coinmonks/ethereum-smart-contract-development-with-a-web-app-part-1-develop-the-smart-contract-ee2a7c735936

9. Appendix

9.1. Prototype Design

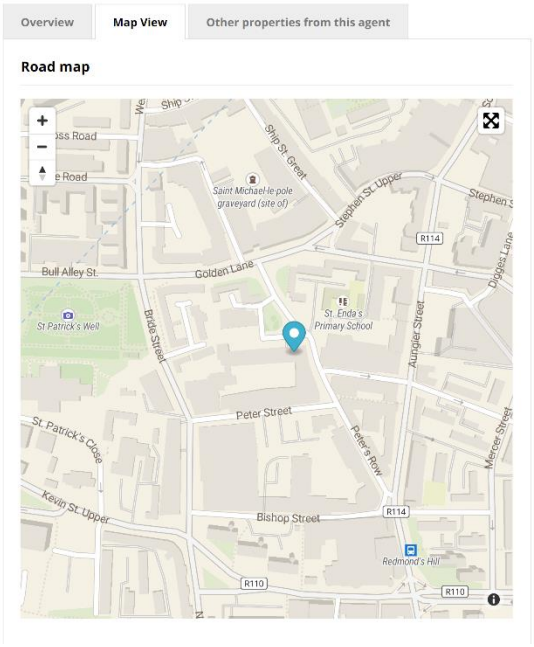


Figure 78: Screenshot of Daft Property Location (46)

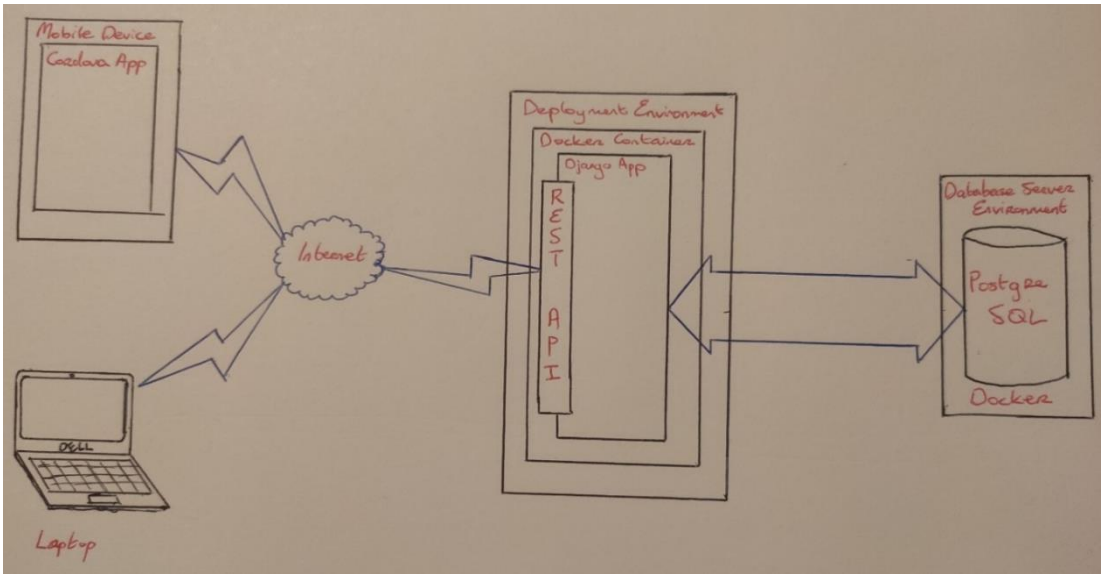


Figure 79: Architectural Diagram Prototype

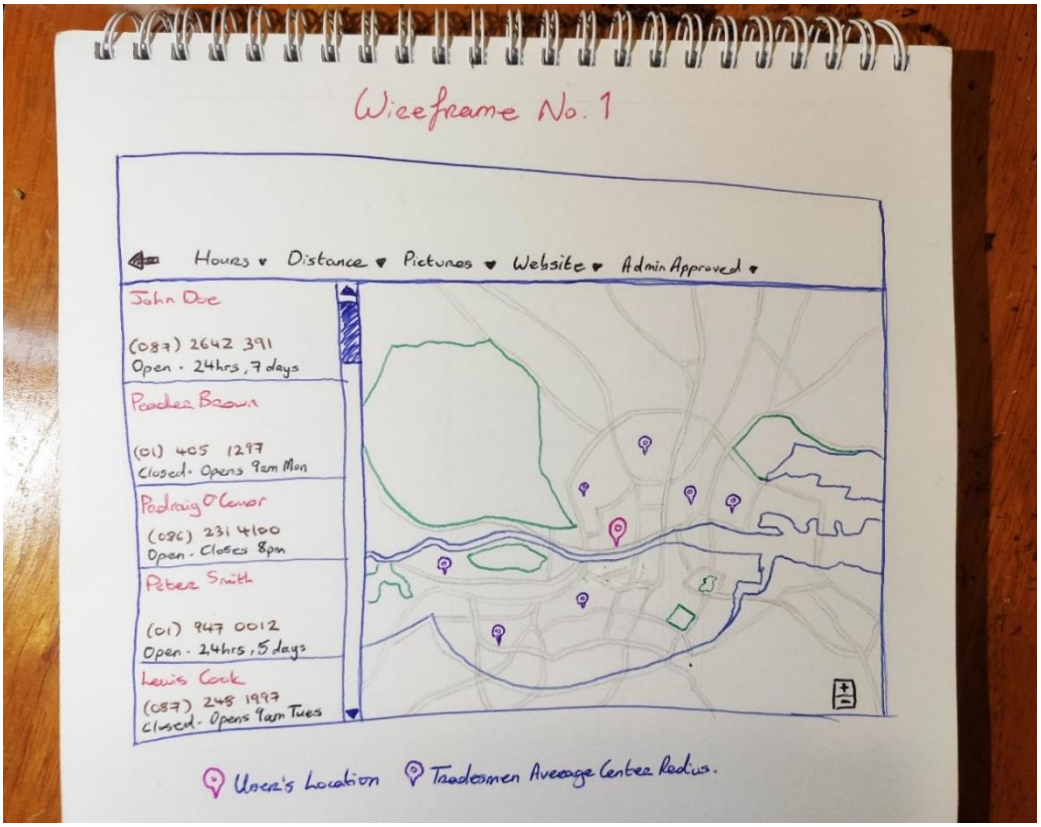


Figure 80: Low-fidelity Prototype 1

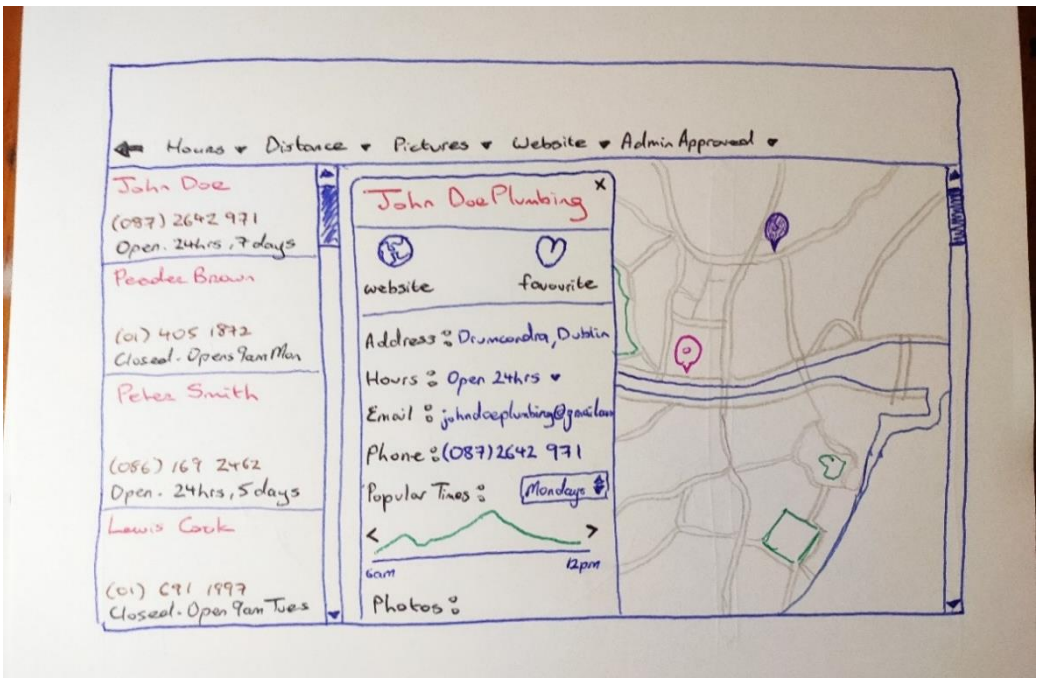


Figure 81: Low-fidelity Prototype 2

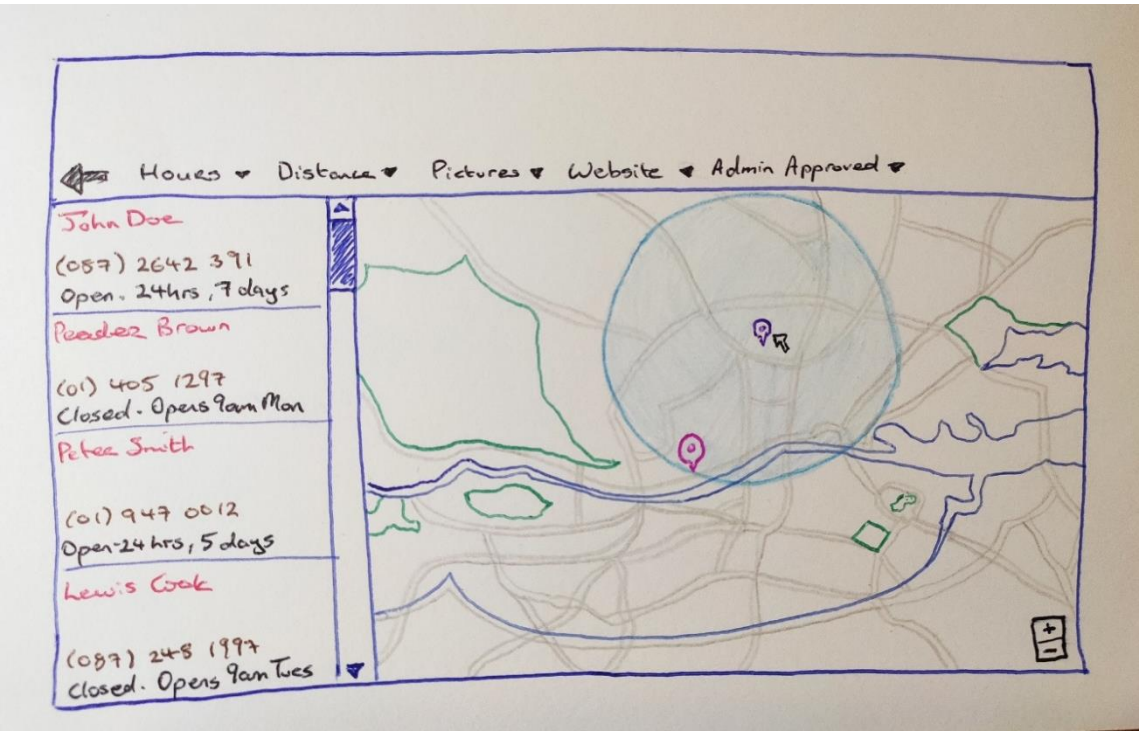


Figure 82: Low-fidelity Prototype 3

9.2. Prototype Development

```
if (this.readyState === 4 && this.status === 200) {
    var data = eval(req.responseText);
    for (let i= 0; i < 4; i++) {
        var preium_listings = document.getElementById( elementId: 'PremiumListings');
        var preium_listings_image_div = document.createElement( tagName: 'DIV');
        preium_listings_image_div.setAttribute( qualifiedName: "class", value: "col-md-3 col-sm-6");
        preium_listings_image_div.setAttribute( qualifiedName: "id", value: "PremiumListingsDiv");

        var preium_listings_details = document.createElement( tagName: 'DIV');
        preium_listings_details.classList.add("PremiumDetails");
        var preium_listings_h3 = document.createElement( tagName: 'H3');
        preium_listings_h3.classList.add("service-heading");
        preium_listings_h3.innerHTML = data[i].company_name;
        preium_listings_h3.setAttribute( qualifiedName: "id", value: "PremiumListingsName");

        preium_listings_details.appendChild(preium_listings_h3);
        preium_listings_image_div.appendChild(preium_listings_details);
        preium_listings.appendChild(preium_listings_image_div);
```

Figure 83: HTML Elements Created & Assigned to the DOM

```
var post="";
dbPromise.then(function(db){
    var tx = db.transaction('feeds', 'readonly');
    var feedsStore = tx.objectStore('feeds');
    return feedsStore.openCursor();
}).then(function logItems(cursor) {
      if (!cursor) {
        document.getElementById('offlinedata').innerHTML=post;
        return;
      }
      for (var field in cursor.value) {
            if(field=='fields'){
                feedsData=cursor.value[field];
                for(var key in feedsData){
                    if(key =='osm_id'){
                        var name = '<h3>'+feedsData[key]+'</h3>';
                    }
                    if(key =='name_tag'){
                        var author = feedsData[key];
                    }
                    if(key == 'name_ga'){
                        var body = '<p>'+feedsData[key]+'</p>';
                    }
                }
                post=post+'<br>'+name+'<br>'+author+'<br>'+body+'<br>';
            }
        }
      return cursor.continue().then(logItems);
    });
```

Figure 84: Implementation of a Service Worker

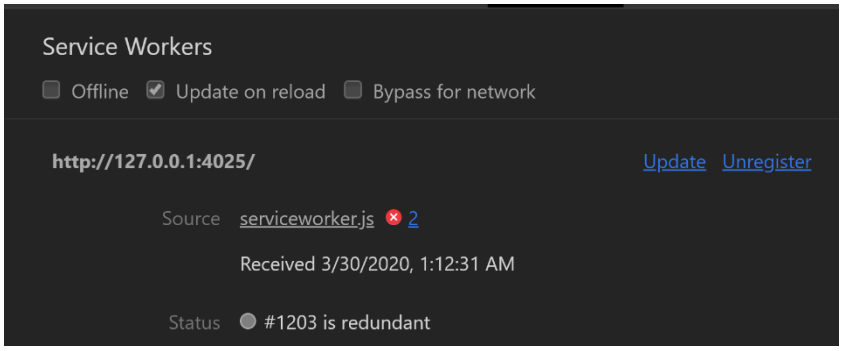


Figure 85: Google Chrome's Inspect Tab Showing Running Service Worker

```
function showAllPremiumTradespeople(map, markerGroup, markerGroup1, markerGroup2) {
    var req = new XMLHttpRequest();
    var url = '/getAllPremiumTradespeople';
        req.onreadystatechange = function() {
            if (this.readyState === 4 && this.status === 200) {
                var data = eval(req.responseText);
```

Figure 86: Searching 'urls.py' connecting the application's template to its view

```
urlpatterns = [
    path(r'', views.main_base_view, name='base_layout'),
    url(r'^getAllTradespeople/$', views.getAllTradespeople, name='getAllPlumbers'),
    url(r'^getAllLatestTradespeople/$', views.getAllLatestTradespeople, name='getAllLatestTradespeople'),
    url(r'^getAllPremiumTradespeople/$', views.getAllPremiumTradespeople, name='getAllPremiumTradespeople'),
    url(r'^search/$', views.search, name='search'),

    url(r'^plumber_data/$', views.plumber_data, name='plumbers'),
    url(r'^plumber_data_website_premium/$', views.plumber_data_website_premium, name='plumber_data_website_premium'),
    url(r'^plumber_data_premium/$', views.plumber_data_premium, name='plumber_data_premium'),
    url(r'^plumber_data_website/$', views.plumber_data_website, name='plumber_data_website'),

    url(r'^electrician_data/$', views.electrician_data, name='electricians'),
    url(r'^electrician_data_website_premium/$', views.electrician_data_website_premium, name='plumber_data_website_premium'),
    url(r'^electrician_data_premium/$', views.electrician_data_premium, name='plumber_data_premium'),
    url(r'^electrician_data_website/$', views.electrician_data_website, name='plumber_data_website'),

    url(r'^painter_data/$', views.painter_data, name='painters'),
    url(r'^painter_data_website_premium/$', views.painter_data_website_premium, name='painter_data_website_premium'),
    url(r'^painter_data_premium/$', views.painter_data_premium, name='painter_data_premium'),
    url(r'^painter_data_website/$', views.painter_data_website, name='painter_data_website'),
]
```

Figure 87: Searching URLs to call certain Tradespeople

```
class TradespersonSerializer(serializers.Serializer):
    username = serializers.CharField(max_length=20)
    password = serializers.CharField(max_length=20)
    firstname = serializers.CharField(max_length=20)
    lastname = serializers.CharField(max_length=20)
    email = serializers.CharField(max_length=20)
    working_hours = serializers.CharField(max_length=20)
    travel_distance = serializers.CharField(max_length=20)
    phone_no = serializers.CharField(max_length=20)
    website_url = serializers.CharField(max_length=20)
    company_name = serializers.CharField(max_length=20)
    occupation = serializers.CharField(max_length=20)
    premium = serializers.BooleanField(default=False)
    town = serializers.CharField(max_length=25)
    county = serializers.CharField(max_length=25)
    image = serializers.ImageField()
```

Figure 88: Tradesperson Serializer Model

```
def getAllLatestTradespeople(request):
    listLatest = list()
    tradespeople = Tradesperson.objects.all().order_by('id')
    for people in tradespeople:
        ser = TradespersonSerializer(people)
        listLatest.append(ser.data)
    import json
    return HttpResponse(json.dumps(listLatest))
```

Figure 89: Latest Listings

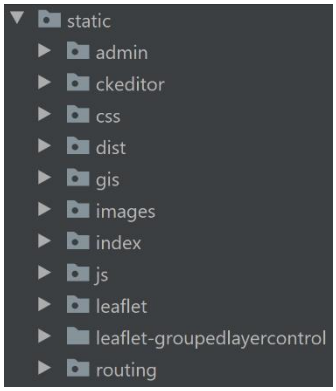


Figure 90: Static Files

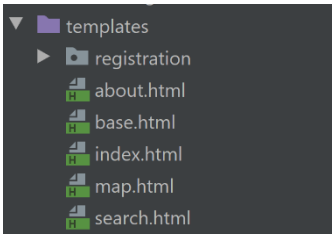


Figure 91: Template Files

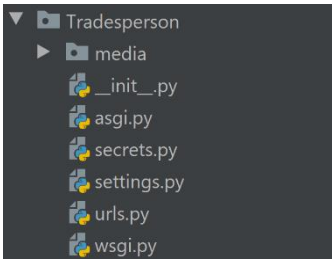


Figure 92: Tradesperson Directory

```python
class GlobalUserRegisterForm(forms.ModelForm):
    username = forms.CharField(label='Enter username')
    password = forms.CharField(label="Enter password")

    class Meta:
        model = User
        fields = [
            'username',
            'password'
        ]

    def clean(self, *args, **kwargs):
        username = self.cleaned_data.get('username')
        password = self.cleaned_data.get('password')
        email_qs = User.objects.filter(username=username)
        if email_qs.exists():
            raise forms.ValidationError(
                "This username is already being used")
        return super(GlobalUserRegisterForm, self).clean(*args, **kwargs)
```

Figure 93: Consumer Registration Form

```python
class UserRegisterForm(forms.ModelForm):
    username = forms.CharField(label='Username')
    password = forms.CharField(widget=forms.PasswordInput, label='Password')
    password2 = forms.CharField(widget=forms.PasswordInput, label='Confirm Password')
    firstname = forms.CharField(label="Firstname")
    lastname = forms.CharField(label="Lastname")
    email = forms.EmailField(label="Email")
    email2 = forms.EmailField(label="Confirm Email")
    working_hours = forms.ChoiceField(label="Working hours", choices=workingHours)
    travel_distance = forms.IntegerField(label="Maximum travel distance (km)")
    phone_no = forms.CharField(label="Phone number")
    website_url = forms.CharField(label="Website", required=False)
    company_name = forms.CharField(label="Company name", required=False)
    occupation = forms.ChoiceField(label="Occupation", choices=occupation)
    town = forms.CharField(label="Town")
    county = forms.ChoiceField(label="County", choices=county)
    image = forms.ImageField(label="Upload Profile Image")
```

Figure 94: User Registration Form Attributes

```python
def login_view(request):
    next = request.GET.get('next')
    form = UserLoginForm(request.POST or None)
    if form.is_valid():
        username = form.cleaned_data.get('username')
        password = form.cleaned_data.get('password')
        user = authenticate(username=username, password=password)
        login(request, user)
        if next:
            return redirect(next)
        return redirect('/')

    context = {
        'form': form
    }

    return render(request, "login.html", context)
```

Figure 95: User Login View

```python
def tradesperson_edit_profile(request):
    data = Tradesperson.objects.get(username=request.user)

    if request.method == "POST":
        form = TradespersonEditProfileForm(request.POST, instance=data)
        form2 = EditProfileForm(request.POST, instance=request.user)
        if form.is_valid():
            if form2.is_valid():
                form.save()
                form2.save()
                return redirect('base_layout')

    else:
        form = TradespersonEditProfileForm(instance=data)
        context = {
            'form': form
        }
        return render(request, 'tradesperson_edit_profile.html', context)
```

Figure 96: Tradesperson Edit Profile View

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

Figure 97: Email Back-End Defined in settings.py

```
RUN apt-get -y update && apt-get -y upgrade && apt-get -y install python3-dev python3-setuptools python3-pip libgdal-dev nginx supervisor
RUN pip3 install --upgrade pip setuptools wheel

# Make a new direcctory on your target image and set it as your working dorectory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# COPY requirements.txt and RUN pip install BEFORE adding the rest of your code, this will cause Docker's caching mechanism
# to prevent re-installing (all your) dependencies when you made a change a line or two in your app.
COPY requirements.txt /usr/src/app
RUN pip3 install -r requirements.txt

# add (the rest of) our code
COPY . /usr/src/app

# Set up all the configfiles
COPY django_nginx.conf /etc/nginx/sites-available/default
COPY supervisor-app.conf /etc/supervisor/conf.d/

# Expose the image's ports. We'll bind different host ports to these later
EXPOSE 80
EXPOSE 443

# When a new container is created, we'll run supervisord to start uwsgi and nginx.
CMD ["supervisord", "-n"]
```

Figure 98: Dockerfile implemented to package the application to be run on Docker

```
def remove_favorite(request):
    if request.is_ajax():
        t = request.POST.getlist('remove_tradesperson')
        Favourites.objects.get(username=request.user, firstname=t[0], lastname=t[1]).delete()
        message = "Removing favourite using AJAX successful"
    else:
        message = "Not Ajax"
    return HttpResponse(message)
```

Figure 99: View to Remove Tradesperson from Favourites

```
class TestProjectHomePage(StaticLiveServerTestCase):

    def setUp(self):
        self.browser = webdriver.Chrome('functional_tests/chromedriver.exe')

    def tearDown(self):
        self.browser.close()
```

Figure 100: Functional Testing Setup