Causeway Coast & Glens Interactive Web Mapping (GIS)

A dissertation submitted in partial fulfilment of

The requirement for the degree of
MASTER OF SCIENCE in Software Development
in
The Queen's University of Belfast

By

Markus Condren

23rd September 2022

Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.
6. Software and files are submitted via Canvas.

I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.
I am aware of the disciplinary consequences of failing to abide and follow the School and Queen's University Regulations on Plagiarism.

Name: (BLOCK CAPITALS)	**MARKUS CONDREN**
Student Number:	**40128955**

Student's signature Markus Condren *Date of submission* **23/09/22**

Acknowledgements

I would like to thank the following people for helping me with this research project:

Dr David Cutting, my project supervisor, for his continued support, insight and ever calming influence. Our meetings on a Monday afternoon often acted as a tonic, just what was needed at the start of the week. Thank you for all your help.

My mother; Connie, father; Pat, and brother; Lukas for always offering an ear for me to brainstorm or vent, sometimes both, regardless of their level of interest. Thank you for your continued support and encouragement throughout all my education, but particularly for this last twelve months.

Mr Aidan McGowan, Dr Neil Anderson, Dr Leo Galway, Mr John Busch and Dr Sanmi Olade for their support and encouragement while teaching me the core modules of the MSc. Without their tutorage I would not have been able to attempt this project, thank you all.

Abstract

The Royal National Lifeboat Institution (RNLI) attended 17,000 incidents, giving aid to 30,000 people, along the UK shoreline in 2019. Concerned that this trend would continue, the RNLI launched a new campaign – Beach Smart. The campaign was designed to inform residents and visitors on how to enjoy the UK's coastline in a safe and environmentally friendly way. This dissertation explores the development of an app, as a hub for beach smart. The front-end is built using ReactJS, consists of an interactive map using Open Street Maps using LeafletJS and React-Leaflet, with styling using Bootstrap 5 and React-Bootstrap. The backend administrative portal and API written in PHP. This acts as proof of concept for the BeachSmart app, focussing on Portstewart's coastline and points of interest. The BeachSmart app is an excellent concept. It is certainly something that Northern Ireland could benefit from, given the importance the coast plays in public consciousness. The app developed during this project is by no means a complete product. It does, however, suggest that if such a project were to be developed professionally it could be an excellent application. One which could really enrich the coastal experience for locals and tourists alike.

Keywords: Beach safety; Ocean safety; Royal National Lifeboat Institution; Causeway Coast & Glens Council; Web Development; ReactJS; JavaScript; PHP.

Table of Contents

Introduction

In 2019, the Royal National Lifeboat Institution (RNLI) attended 17,000 incidents, giving aid to nearly 30,000 people. Of these, 200 alone were along the coast of Northern Ireland (CommunityAd, 2020). As a result, following the easing of lockdown restrictions in May 2020, fearing an influx of visitors to the coast, the RNLI and HM Coastguard launched a “beach smart” campaign (NIWorld, 2020) aiming to inform residents and visitors on how to enjoy the beaches and coastline of Northern Ireland, England, Wales and Scotland safely and with consideration of the natural environment. This was done in the hopes of reducing the load on their, already strained, resources. Comprising 12 district councils (Kelly and Whyte, 2019), Causeway Coast & Glens Council (CCG) oversees 11 beaches stretching from Benone to Cushendall (Causeway Coast & Glens Borough Council, 2022). There is limited in-depth information, that is easily accessible in one place online, about the coast in general but the beaches. This makes it difficult, particularly for elderly and people with disabilities, to access to required information to enjoy the coastal features safely.

RNLI crews gave aid to 53,665 people in 2021, up 60% from 33,546 in 2020 (RNLI, 2021). While 35 people required assistance from volunteer lifeboat crews *daily* in 2021, a 52% increase from 23 in 2020 (RNLI, 2021). It must be said that these figures relate to the UK in its entirety, with much of the higher lifeboat activity being seen in the south of England (Fig. 1). Despite this fact, as mentioned previously, there is a not insignificant number of incidents taking place along the coast of Northern Ireland, e.g., nearly 4,000 people saved via water rescue since 2011 (RNLI, 2021). Coupled with the wild and ragged nature of much of the North Coast, which remains largely unwatched by safety crews, it is worth trying to minimise incidents by whichever means.

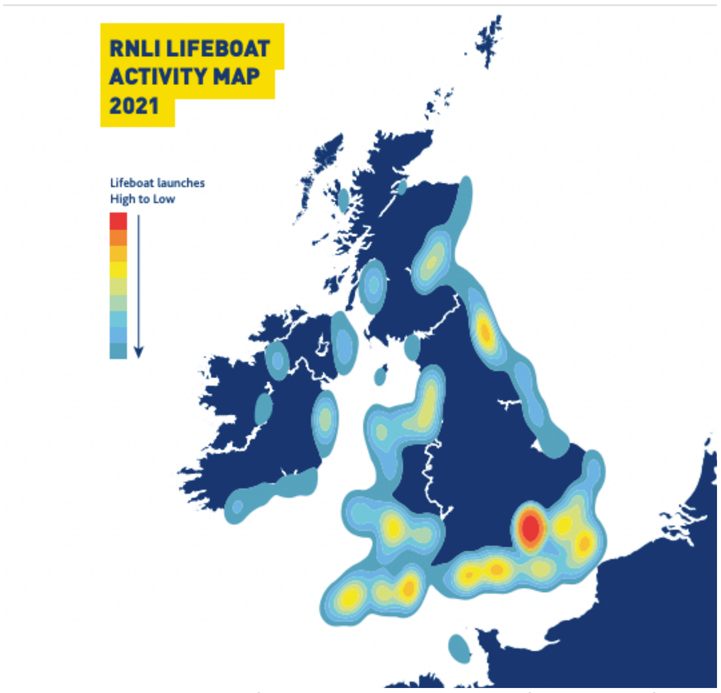


Figure 1 - RNLI Lifeboat Activity Map 2021 (RNLI, 2021)

It is hoped that by making this information easily accessible to more people, there will be fewer incidents requiring the attention of the RNLI or coastguard. This should, in turn, allow for resources to be further invested in improving beach safety for all, improving equipment and training, and potentially free up funds for investment in the personnel that carry out the rescue work. This may be particularly prudent, given that 95% of RNLI staff are volunteers (RNLI, 2021) and 92% of the money raised by the RNLI comes from charitable donations (RNLI, 2021), so there is never a guarantee of further funding.

This dissertation aims to describe the process of developing the BeachSmart web app – an information platform for the Causeway Coast & Glens Council (CCG), in conjunction with, and based on the beach smart campaign launched by, the RNLI.

Chapter 1: Understanding the Problem

The beach smart web app aims to provide a single portal for all things CCG. Users will be able to use the app to explore CCG beaches, coastline and towns, including local landmarks, entertainment, eateries and shops.

This, primarily, is aimed at improved beach safety education among locals, with an estimated 144,943 people currently living in the CCG area (NINIS, 2020) as of June 2020. The same technology may then be used for the rest of the United Kingdom, given that the beach smart campaign encompasses all the coastal territory under the remit of the RNLI.

Tourists, however, represent different challenges when it comes to beach safety as they may be unfamiliar with their surroundings, may be inexperienced with beach and sea safety concepts or may simply take more risks than usual due to their being on holiday.

Tourism spending accounts for 5.7% (£131.5 billion) of the entire UK's GDP, in the same year (World Travel & Tourism Council, 2022). In Northern Ireland alone, 56% of visitor trips are made up of external visitors (DfE, 2020). The COVID-19 pandemic did have a significant impact on tourism levels with fewer tourists coming to Northern Ireland due to pandemic and lockdown restrictions. Having said that, pre-pandemic there were 5.3m tourists visiting Northern Ireland, this figure (TourismNI, 2022). The COVID-19 pandemic saw significantly fewer tourists visit. Pre-pandemic tourism contributed £1bn to the NI economy, of which 70% was from external visitors (DfE, 2020). Therefore, it is important for the beach smart app to be rich in functionality that applies to tourists as well as locals, to take advantage of recovering tourism numbers. This could have a significant impact on the economic prosperity of the area in the coming years and help improve education around beach safety for all.

Following the analysis of the problem at hand, requirement prioritisation is the process of looking at the proposed features a client or customer wants to include in their product and deciding on how best to prioritise them during the development cycle. This is done to optimise the allocation of time and resources to the project so that a high-quality product may be brought to market in the quickest time possible, with as little waste as possible. Requirement analysis is oft considered one of the more important aspects of the development lifecycle (Ngo-The and Ruhe, 2005), with poor analysis having the potential to de-rail projects at any given time.

There are a few well used methods of requirement prioritisation widely used in industry. For the project, three were compared, Ranking, Kano Analysis and MoSCoW prioritisation.

Ranking involves creating a big list of potential requirements and ranking them from most to least important. Although simple in theory, ranking requirements in this way may become unmanageable as requirements are added and the project increases in complexity. It has been suggested to reserve this for very small projects as to process many requirements in one large list becomes inefficient once they grow beyond 7 (Miller, 1994). It was decided that Ranking would be inappropriate as the project would likely have more requirements and it did not feel appropriate when following an Agile methodology as requirements are listed by preference, but no indication is given as to the amount of preference between requirements; they are all weighted equally (Hatton, 2008).

The Kano Model was developed by Professor Noriaki Kano in 1984 (Kano, 1984; Wikipedia, 2022). Kano analysis looks at requirements for a project in terms of their effect on customer satisfaction. This is done by assigning each product requirement an attribute of one of the following. Threshold: the most basic requirements expected of the product, Performance: requirements that may not be wholly necessary but would improve the customer satisfaction of the product and Excitement: these are requirements which the client may not have even thought of but ones which would add a high level of satisfaction to the final product (MindTools, 2016). Requirement analysis using Kano analysis can then be ratified through standardised questionnaires, given to the development team, client group or prospective users, in order to statistically measure opinion on the matter (Kano, 2022). The fact that the Kano Model uses real customer data to analyse requirements is intriguing, this would offer a wide range of opinions from real-world users of various backgrounds and persuasions. In theory, this would lead to a more robust, customer-centric development cycle with more highly tuned requirements. In the case of this project, it may have been somewhat excessive. Unfortunately, the proposed contact at the council is off on long-term leave, this lead to Dr David Cutting acting as the client (who really was excellent in doing so) and meant there would be no access to the prospective users in the numbers required for accurate and valid conclusions to be drawn from any questionnaire-based survey – a minimum of 100 participants has been suggested as this number (Graglia, 2022; Bullen, 2022). Were this project to be carried out on an industrial scale, this may well have been the most appropriate form of requirement prioritisation due to the user-focussed nature of the project.

Finally, MoSCoW prioritisation. Developed in 1994 by Dai Clegg (Agile Business, 2022), the acronym MoSCoW denotes the four categories in which requirements can be placed, these describe the hierarchical importance of each requirement in terms of *must* have, *should* have, *could* have and *won't* have (yet), with the letter 'o' added in-between to aid pronunciation. Must have requirements are non-negotiable, they must be included for there even to be a minimum viable product. Should have requirements are features that would make for a nice addition to the project if they can be developed and integrated but won't cause it to fail if not. Could have requirements are ones which would add to the product, although less than should haves, but are also not essential to the completion of the project. Won't have requirements will not be included, this must be caveated with the fact that they are not bad or inconsequential requirements, they are simply ones which will not be included within the given timeframe of the project. Though these may be revisited as the project in maintained and further developed upon.

It was decided that MoSCoW prioritisation would be most appropriate for this project. It allows for clear and precise categorisation of requirements and does not necessarily rely on constant customer feedback to be effective. It also gives the opportunity to learn a new form of requirement analysis and prioritisation.

Table 1 - Final MoSCoW requirement prioritisation for BeachSmart

	Requirement
Must have	• Accessibility features for users with keyboard-only or visual/auditory impairments – to include light/dark themes and support for text-to-speech services.
	• An interactive map with pins displaying the beaches that the council oversees.
	• Users must be able to explore the map, to include zooming and panning.
	• A link for each beach to the relevant CCG website page.
	• Environment, historical and practical (accessibility, closure times etc.) information for a beach that is selected. To include beach status such as flag and water category. Either included in popup or available via external link.
	• Users must be able to easily contact the council for any extra information or to report anything.
Should have	• Local landmarks, businesses and sports facilities, highlighted with icons/pins – to include opening hours, contact details and links to external websites, where applicable.
	• A map filter allowing users to decide what type of amenity or attraction type they wish to browse.
	• Live weather and tidal data allowing users to safely explore the area.
	• Bus and train routes around the area with timetables (better UX) or links to external Translink website.
	• Should have external links to other websites for information on the area, beaches, lifeguards etc., this should open externally so as not to confuse any less technically minded users or cause someone to lose their place on the map.
	• Points of interest stored in a database and accessed via URL requests.
	• There should be the ability for an admin to access all points of interest as a list and be able to add, update or delete points of interest on the map.

Could have	• Users could be able to select items on the map, save to an itinerary and download as a pdf for future reference.
	• Users could pick a list of beaches, landmarks etc., they wish to visit and give their available time, start and end point. An algorithm would be used to calculate their most efficient route. This is perhaps a more tourist-oriented requirement but could be of great use to locals too.
	• Users could be able to create an account – allowing them to add map items to a watchlist, get notifications about events and save favourite routes – could sign up with email or Google account.
	• Users could report any rubbish, damaged property or anti-social behaviour to the council, perhaps uploading a photo from the app. This would be reported along with time, date and geolocation data, allowing the council to take quick and efficient action.
	• Users could report sightings of wildlife e.g., birds, whales, dolphins or seals, which occasionally show up on the North Coast. These could be pinned to a location to let others know where they may catch a glimpse. This could be used also to highlight areas where people have, for example, been stung by jellyfish. This could allow others to bathe and enjoy the beach safely and with greater peace of mind.
	• An API, likely built using PHP or NodeJS, allowing other developers to make use of the aggregated data used in the app.
	• Social media aggregators which display tweets or Facebook posts using pre-determined hashtags or the CCG social feeds, as per "plan your trip" page on the CCG website (RNLI, 2022).
Won't have yet	• Integration with Fitbit, Strava or similar, to save routes or walks and share progress with friends and family.
	• Social media integration allowing posts to be made direct from the web app.
	• Photos uploaded by viewers aggregated together and shown in the information section of each pin location as a slideshow with username and date below.
	• Augmented reality functionality, allowing users to preview the area and decide as to where they wish to visit.

Once requirements have been decided upon and prioritised, the technology best suited to the project must be chosen.

Technology considerations

The brief for this project was to create a web-based, interactive mapping application. Thus, technology was considered for front and back-end as well as the database element.

Front-End

Given the web-based nature of the project, the main structure of the website was developed in HyperText Markup language (HTML) 5, with styling coming from Cascading Style Sheets (CSS).

When considering a general plan for CSS, global variables to be used where possible for colours, fonts and any other constant values that are to be reused throughout the codebase. Where possible, height, width, margins, etc., to be controlled vw and vh, keeping them relative to the width and height of the given viewport. This will be important for maintaining responsiveness across the entire array of modern device screen dimensions. Likewise, font sizing to be controlled using relative units – rem or em for the same reason. The decision remained whether to use plain CSS or a library/framework. There are advantages and disadvantages to both; plain CSS allows for ultimate customisation and control over every aspect of the app's CSS and can avoid the added project bulk, given that some external frameworks and libraries often are massive files (Agrawal, 2021), which on a project of this scale may or may not affect performance. On the other hand, writing all the app's CSS from scratch would take a huge effort, with the added potential for bugs. The advantages of using a framework tend to revolve around the ease with which one can implement advanced features e.g., navbars, dropdown menus or nested elements which maintain their properties throughout (something which can be tricky when building CSS from scratch). Frameworks also allow for cross-browser support of the project's features and should minimise the occurrence of bugs as new versions of browsers are developed, and older versions defunct (Bose, 2022). The main advantage of using a framework in the case of this project is that they allow for the development of a modern, stylish user interface (U.I) that requires minimal external styling, with high-quality responsiveness across devices, with minimal effort and, quite often, well-written documentation (Coyier, 2008).

When considering the use of a CSS framework, there are many popular ones in the market. There were three considered for this project due to having previous experience working with each – Bulma, Tailwind CSS and Bootstrap.

Bulma is an open-source, class-based, mobile-first CSS framework (Bulma, 2022). Built on flexbox, it is responsive across various devices. It comprises CSS classes which are easy to implement in a project, including many components such as forms, navbars, menus etc., which are simple and sleek in appearance. Bulma also includes many optional Sass (an extension language of CSS) files which can be imported for additional functionality (Juviler, 2022).

Tailwind CSS is a lightweight, fast framework, described as 'utility-first'. Unlike Bulma and Bootstrap, Tailwind is not a U.I. kit but rather than have a default U.I. theme, tailwind uses 'widgets' and looks at the HTML and JavaScript code one has written to generate a static CSS file using the relevant class names (TailwindCSS, 2022). This in theory allows the quick creation of website U.I.s, with one commentator describing it as "a cool way to write inline styling" (Ekwuno, 2021). Tailwind has fast loading times but is not as robust as the likes of Bootstrap when it comes to deploying U.I. components which can be utilised site-wide (Ozanich, 2022).

Bootstrap is a mobile-first build which scales well on desktop. As is the case with Bulma, Bootstrap allows for a lightweight, responsive design with components that are easily customisable and offer high levels of functionality. Bootstrap also allows access to open-source icons for customising the look of the site. As these come with Bootstrap, they are free to use without additional accreditation, reducing the likelihood of legal complications arising from the finished product. Although external icons may be used, with accreditation. It is robust in its use of HTML, CSS and JavaScript component templates (Ozanich, 2022), and has a good track record in industry being, according to W3Techs, used by over 25% of websites (W3Techs, 2022).

As this project is primarily designed to be used as a web-app on a mobile phone, it was important to use a framework that is designed 'mobile-first', with high-quality and reusable U.I. components with functionality that reflects this. Tailwind CSS has many advantages, not least with its smaller loading times. It was decided though, that a U.I. kit framework such as Bulma or Bootstrap would be more appropriate. Bulma or Bootstrap would be perfectly adequate for this project but ultimately it was decided that Bootstrap would be used, primarily out of personal preference, having worked with both throughout the year.

Having decided on a CSS framework, the next technology to consider was JavaScript. JavaScript is a client-side scripting programming language - which can also be used in server-side applications, in non-browser environments e.g., Node.js - currently used in 98% of websites (W3Techs, 2022). It is a lightweight language which supports imperative, declarative and object-oriented programming (MDNWebDocs, 2022) and is compiled at run-time as opposed to pre-execution (Aycock, 2003). JavaScript is responsible for all the proposed interactive elements of the project and is the go-to in any modern, web-based application (Duraj, 2019). The question remains as to which 'flavour' of JavaScript would be most appropriate.

JavaScript written from scratch is known as 'Vanilla'. This refers to JavaScript that is written with no assistance from any external libraries or frameworks. Using Vanilla JavaScript for this project has its pros and cons.

Firstly, as Vanilla JavaScript is JavaScript in its purest form, it would offer an excellent learning experience. Unfortunately, due to timetabling pressures, we were unable to do JavaScript until the final week of our Web Development module. This meant that, regardless of how the project was to be developed, Vanilla JavaScript would have to be learned alongside whatever other technologies were chosen, even if it were not to be written entirely in Vanilla JavaScript.

Vanilla JavaScript is very light and quick compared to libraries and frameworks; this refers to the average file sizes of a project, the number of dependencies required and how quickly the files are compiled. An experiment run by Jeremy Wagner in 2020 found that Vanilla JavaScript is 30 times faster than react in terms of U.I. rendering and similarly, deals with changes in state much quicker (Wagner, 2020). This is an important consideration to make, having said that, on a project this size the speed increases may not outweigh the ease of use of a framework or library.

Writing Vanilla JavaScript takes time, in some cases a considerably longer development time than when compared to using a library of framework. This is because each function, component or feature must be written from the ground up (if you wanted to commit to pure Vanilla JavaScript). For a small project such as BeachSmart, this is maybe not a problem in the immediate term. Having said that, as there is a relatively type development period, any time-saved on more elementary features which are taken care of by frameworks, e.g., navbars or button-groups, may be worth considering.

Vanilla JS is not recommended when programming for cross-compatibility in legacy browsers such as Internet Explorer or older versions of Chromium-based browsers – this can lead to newer features or code may not run acceptably (MDN, 2022). This was more of an issue in the earlier days of JavaScript, although it is worth considering as nowadays as using vanilla JavaScript where the possibility exists that users may not be using the latest browsers – with, an admittedly small percentage, of 1.45% of people still using Internet Explorer on desktop (Dean, 2021). One would be forgiven for thinking this is not a significant percentage, but if extrapolated across a (hopefully) large user-base, this would lead to a significant number of users having issues with the project. This was less of a consideration when considering that the project is designed to primarily be used on mobile devices, where over 62.27% of people are using Chrome as their mobile browser (Dean, 2021).

An alternative to writing the project in Vanilla JavaScript is to use Typescript. Released in 2012 by Microsoft (Wikipedia, 2022), Typescript is open-source language that compiles to JavaScript. As of 2019, TypeScript was the 7th most-used and 5th in terms of usage adoption across the industry, respectively (GitHub, 2019). Despite being a separate programming language, TypeScript is a superset of JavaScript which means that any valid piece of JavaScript code is also valid TypeScript code. As with JavaScript, there are some advantages and disadvantages to using TypeScript.

Most of the advantages refer to the fact that using TypeScript allows us to define in strict terms what type any given variable is. JavaScript is loosely typed, meaning that a variable might either contain a number, text or Boolean value. TypeScript on the other hand, more similarly to something like Java, can be strongly typed. If we were to define a variable as type Boolean, TypeScript will throw an error when we try to assign a number to it. JavaScript on the other hand will run as normal with no errors. This can be advantageous when it comes to structuring your code and maintaining strict control over the types of variables being used. Additionally, TypeScript allows us to follow a stricter object-oriented programming structure with Classes, Enums etc as in Java. TypeScript might be considered a better option than JavaScript as it allows us to precisely define data types, which in turn may make managing and debugging the code more straight forward as type errors are more easily picked up by the developer. Since these errors are not generally found by the browser or end-user, having TypeScript flag such errors may be advantageous. TypeScript also, allegedly, makes developing in a team easier as there is less dependence on others while working on a specific feature due to the strict-typing nature (Swistak & Stempniak, 2022).

TypeScript does have its disadvantages, not least that it is rather complicated in how it is written, which can lead to a bloated code base as more code is required for a typical TypeScript file compared to JavaScript. (AltexSoft, 2022). TypeScript also has the extra step of the code being transpiled to JavaScript although this is not likely to make any significant difference in this project. The main disadvantage of trying to use TypeScript is that it would have to be learned from scratch, alongside learning JavaScript and potentially additional frameworks etc. This may or may not be worth it in the end considering that the project will not necessarily be following a strict object-oriented approach, so maybe the optional strong typing of TypeScript is not necessary in this instance.

A modern solution to many U.I. designs is by using a JavaScript framework or library. There are several such options – including but not limited to Angular, React, Svelte, or jQuery (Wikipedia, 2022). It was decided to focus on two – Angular and React.

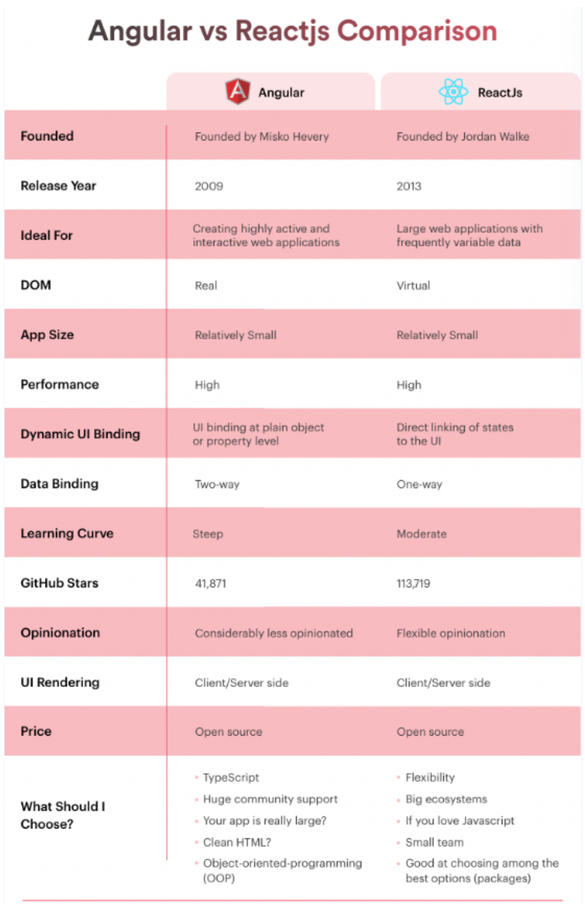
Angular is a JavaScript framework, created in 2010 and actively supported by Google. The most recent versions use TypeScript to create front-end components which can re-used throughout the project. When it was initially released, Angular provided an easy way to quickly build dynamic web pages – proving particularly effective in creating single-page applications. The main advantages of Angular are in the high reusability and readability of the code, as everything can be split into components. This also makes testing somewhat more straight forward as one can run individual tests to verify the performance of each part of each component. These factors make projects written using Angular much easier to maintain than if it were to be written in JavaScript alone (AltexSoft, 2022).

The disadvantages of using Angular tend to relate to the complex nature of its project structure, a main issue being that the management of components is a complicated and effusive, with as many as 5 files being required for a single component (AltexSoft, 2020). Coupled with a steep learning curve (alongside everything else) and the fact it uses TypeScript – although this is often given as an advantage, for this project it may not be given that TypeScript may have to be learned alongside development – it may prove to be too much to take on.

React is a JavaScript U.I. library, created in 2013. Similarly, to Angular, it allows for the creation of interactive and responsive U.Is. using components. React is declarative in that as a developer, one simply need write the code to describe how the Document Object Model (DOM) should look and react takes care of updating the system to render just what is needed at a given time (Garcia, 2022). Another benefit of React is its 'learn once, write anywhere' concept (Facebook, 2022); existing React code does not need to be rewritten when developing new features as React 'makes no assumptions' about the tech-stack used. This may be advantageous for future development of the project. React may also improve performance as it makes use of a virtual DOM, rather than Angular's real DOM. The virtual DOM exists in system memory, meaning that developers essentially create virtual components which are then converted to the DOM by React, giving rise to higher performance. React applications can also run on the server in node.js or React Native (for app development), this links to the 'learn once, write anywhere' concept but is also important for improving the search-engine optimisation (SEO) of the site. Poor SEO is a problem that can arise when using JavaScript frameworks due to search engines struggling to make sense of applications with lots of JavaScript, but as React renders components as a standard HTML page from the virtual DOM, this is less of an issue (JavaTPoint, 2022).

The disadvantages given for React are generally related to there being poor documentation and that the speed of development in React can be an issue for newer developers in a team (JavaTPoint, 2022). There is also the fact that it only renders the U.I. for an application and so one must decide on other technologies to complete the stack. The good thing about React in this regard is that it is not as strict for integrating with backend technologies as other libraries and frameworks sometimes can be. React is generally accepted as working with almost any backend technology (Livingston, 2016).

The advantages to using Angular or React are both similar – both allow for easy code-reusability, both allow for good management of code as most features are abstracted in individual components and they are both testing-friendly, particularly in terms of running unit tests. Although both may have been appropriate, it was decided to use React, as it is fast, reliable, scalable and doesn't necessitate the strict use of TypeScript. Both also have similar industry usage (Enlyft, 2022) so ultimately it came down to personal preference. An overview comparison is shown in figure 2.



Angular vs Reactjs Comparison

	Angular	ReactJs
Founded	Founded by Misko Hevery	Founded by Jordan Walke
Release Year	2009	2013
Ideal For	Creating highly active and interactive web applications	Large web applications with frequently variable data
DOM	Real	Virtual
App Size	Relatively Small	Relatively Small
Performance	High	High
Dynamic UI Binding	UI binding at plain object or property level	Direct linking of states to the UI
Data Binding	Two-way	One-way
Learning Curve	Steep	Moderate
GitHub Stars	41,871	113,719
Opinionation	Considerably less opinionated	Flexible opinionation
UI Rendering	Client/Server side	Client/Server side
Price	Open source	Open source
What Should I Choose?	• TypeScript • Huge community support • Your app is really large? • Clean HTML? • Object-oriented-programming (OOP)	• Flexibility • Big ecosystems • If you love Javascript • Small team • Good at choosing among the best options (packages)

Figure 2 - Comparison of Angular and React. Image used with permission from https://www.simform.com/blog/angular-vs-react/#:~:text=Angular%20is%20a%20Javascript%20framework,app%20with%20frequently%20variable%20data.

As this project is to be an interactive mapping platform, the next technology to consider was which mapping API would be used to gather the map information.

The obvious option is Google maps. Google maps is the most popular mapping service currently used, with 154.4 million monthly users. As of 2018, 67% of all users made use of Google maps for their mapping needs (Galov, 2022). There are several advantages to using Google maps, not least that it is highly regarded in industry and due to its high usage statistics, there would be next to no learning curve for most users. Google maps is relatively easy on data, using around 5MB per hour of navigating the maps, whilst also allowing maps to be downloaded and used offline, this is of particular importance for mobile users who may have limited data available (Galov, 2022). Finally, having access to multi-language support and street view may be an excellent addition to the project, were it to grow beyond its current scope (Dziuba, 2022).

There are, however, some disadvantages. Primarily that is quite expensive to use at larger scale, although Google offers a $300 free trial mode (Google, 2022), once site traffic increases it can become very expensive. This may be wholly worthwhile in a larger project with industrial backing, but for the sake of this project it may become prohibitively expensive with price-hikes not uncommon (Singh, 2018). A significant disadvantage in regards this project is that Google Maps API is neither open source nor particularly customisable (Dziuba, 2022). This would be quite the limiting factor in a project like this as an open-source API would provide higher levels of customisability.

The other option is to use Open Street Maps (OSM). OSM is an open-source mapping system. It is community driven and maintained which gives the option of maintaining accuracy for more obscure localities – as users can edit and confirm map details such as roads, paths or buildings which may be obscure or require local knowledge. This gives the developer more control as the map can be edited to remove any inaccuracies in the map. It comes with extensive and easy to understand documentation (OSM, 2022) which makes the learning curve more manageable. Some might suggest the main advantage is that it is free to use, although there are some limits on how often one can request data from the API before being blocked. The Open Street Maps API is also quite basic in its functionality, meaning that additional technology or services will be needed to get the most out of it (Dziuba, 2022).

It was decided that given the open source (and free) nature of OSM, it would be the most appropriate service to use. This, however, necessitates the use of some sort of mapping library. The most appropriate for use with OSM appears to be Leaflet JS.

Leaflet JS is an open-source framework that is designed to be mobile-friendly (LeafletGitHub, 2022). As this project will be developed “mobile-first” this is an important factor. LeafletJS works seamlessly with Open Street Maps, with an easy-to-follow guide for implementation of core features, which are designed to be extremely lightweight but there are several plug-ins available to add extra functionality (Leaflet, 2022). There is also a React Leaflet repository library which allows for easy integration of LeafletJS into a React project (ReactLeaflet, 2022). Thus, Leaflet proved to be the most appropriate library for use with OSM.

The final front end technology that may be of use is GeoJSON. GeoJSON is a standard format of JSON that is designed to encode and represent geographical features such as areas or lines along the map. It supports multiple geometries - Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon (GeoJSON, 2022). Using geojson.io, a website designed to allow the plotting of points on an OpenStreetMaps Mapbox map. This can be used to plot, in this case, the walking routes which are available in the various towns along the coast. It is hoped that these coordinates can be stored in the database alongside the start and end point pin marker of the walk, and the lines be shown on the map for users to explore before committing to trying the walk for themselves.

Back-End

The back-end API for this project will deal with accepting network requests from the front-end, extracting data from the database and returning it for render on the front-end. In the future, the API may be developed to generate API keys and allow other developers to make use of the BeachSmart dataset, once it became comprehensive enough to be of use. There were two main options for the backend API – develop full stack in JavaScript using Node JS for the backend, or build it in a separate programming language, most likely PHP.

Node JS is a runtime environment for Java Script, allowing it to be used on the server-side. Developing full stack in Java Script gives certain benefits such as high speed, good performance and good efficiency (TechStrikers, 2022). Node JS allows for fast processing of concurrent requests when compared to other API technologies such as Java or PHP (Peabody, 2017). Having said that, APIs built using Node have been known to be unstable and poor at handling calls to relational databases (Tondon, 2022) which may prove to be an issue.

Alternatively, PHP may be used as it is a reliable language for which to build APIs and is still heavily used in web development despite being 27 years old (Wikipedia, 2022). PHP is open source, fast even with slow internet connections, it has a large community behind it and is excellent for database connections (Volodymyr, 2021). Coupled with a gentler learning curve, having had some previous experience, PHP is likely more appropriate.

Database

There were two main options for database technology, MySQL and MongoDB.

MySQL is a widely used, open-source database management system. It offers cross-platform support and can be used with many different programming languages (TechStrikers, 2022) – this is particularly useful in web development where there may be many programming languages used in conjunction in a project. Secure socket layer makes transmitted data well protected, with MySQL being regarded as the most secure and reliable system for web applications. Although, MySQL may not be as efficient when the database is very large in size and lacks the debugging tools which can be found in other database management systems (TechStrikers, 2022), it is unlikely that this should be an issue in this project but is worth noting.

Mongo DB on the other hand is a no SQL database – that is rather than a relational data system as with MySQL, Mongo DB is an object-based system that employs the use of JSON objects. This may make it more flexible when it comes to searching for data and building a dataset, although given the amount of new technology already being used in this project, adding a new database technology may be unwise.

Both support JavaScript server-side technology. MySQL seems more appropriate for high traffic sites that require high levels of security. Mongo DB is said to be more applicable for sites with an analytical focus. As for which is better, MongoDB being part of the MERN stack containing React may be more appropriate, despite the steeper learning curve. However, given that a relational database is likely to be used in this project, MySQL would seem to be the most appropriate choice. This will be hosted on localhost by MAMP (XAMPP for Windows) and ported across to the EEECS servers upon submission.

The final technology stack chosen was a front end built in ReactJS, using the OpenStreetMap API alongside LeafletJS with a backend likely in PHP and a MySQL database using PHPMyAdmin as an interface for development.

Chapter 2: User Interface Design

It was important to consider the user-interface design in terms of user experience, design principles and accessibility.

Principles of User Interface design

There are several accepted principles of design when it comes to building a modern and accessible U.I. It is important to keep the interface clean and concise. The use of common design elements, such as buttons or forms so that each component within the U.I. works in tandem and does not confuse the user. Using hierarchy within font choice and component layout makes the page much easier to follow and can help funnel less-technical users to the features of the app which would be of most use to them (Usability, 2022). The use of colour should be considered carefully to highlight important information, contrast ratios between the chosen colours must also be in keeping with the Web Content Accessibility Guidelines of at least 4.5:1 for normal text (WCGA, 2019), this will be considered when choosing the projects colour scheme.

Front-end design

Font

When considering fonts, there were two options. Arial, Helvetica, sans-serif was gathered from CCG website using Font Finder chrome extension (Fig. 3), a simple, clean and crisp option that is easily recognisable, which scales well as device size changes.

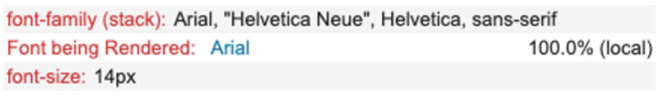

font-family (stack): Arial, "Helvetica Neue", Helvetica, sans-serif
Font being Rendered: Arial 100.0% (local)
font-size: 14px

Figure 3 - Font family gathered from CCG website using Font Finder (https://chrome.google.com/webstore/detail/font-finder/bhiichidigehdgphoambhjbekalahgha?hl=en)

The other option was to use a monospace font – fonts in which there is a constant space between each letter and the width of each letter is the same. Monospace fonts work well as they are very easily read on all manner of devices, they are simple, and uncluttered in design (PumpkinWebDesignManchester, 2019). Monospace fonts allow the user to browse without distraction while giving the app a less 'standard' feel. The monospace font chosen was 'Roboto Mono'. Roboto Mono is a beautiful, legible and readable font (Slant, 2022). The main disadvantages given relate to the fact it is not good for development and cannot be used in the terminal – this is, however, of no consequence to the font being used on the front and back end U.I.

Logo

There doesn't seem to be a beach smart logo as such. As this is a joint venture between Causeway Coast and Glens Council and the RNLI, these could be used. Although, since there was no direct contact with the council for this project, the challenge was taken to design a minimalist logo for the app. A few ideas were considered (Fig. 4a, 4b), designed using Canva. These tended towards the generic and were not wholly original, so ultimately a satellite image of Northern Ireland was imported into Gnu Image Manipulation Program (GIMP), the outline of the council bounds was traced to create an XCF vector image and then exported as a PNG (Fig. 5).

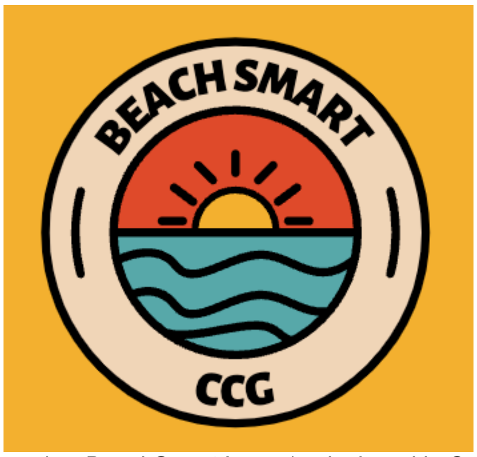


Figure 4a - BeachSmart Logo 1 - designed in Canva
https://www.canva.com/design/DAFM9NJ0ebU/H1Py7DY52HtG6_hkPdVPMQ/edit?layoutQuery=beach+logo

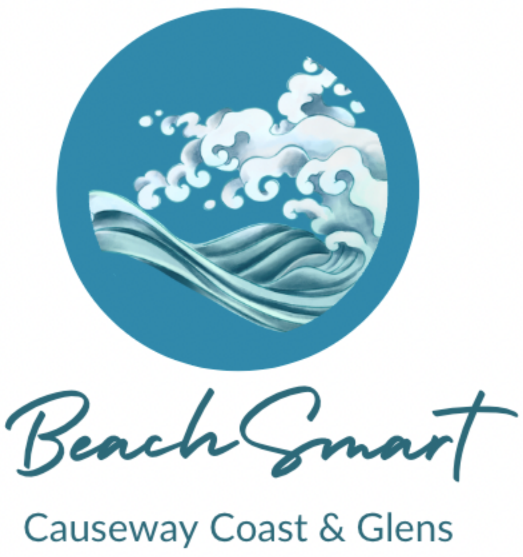


Figure 4b - BeachSmart Logo 2 - design in Canva
https://www.canva.com/design/DAFM9CZzWzE/lsORGSDRUaO7SogLpWoADg/edit?layoutQuery=beach+logo

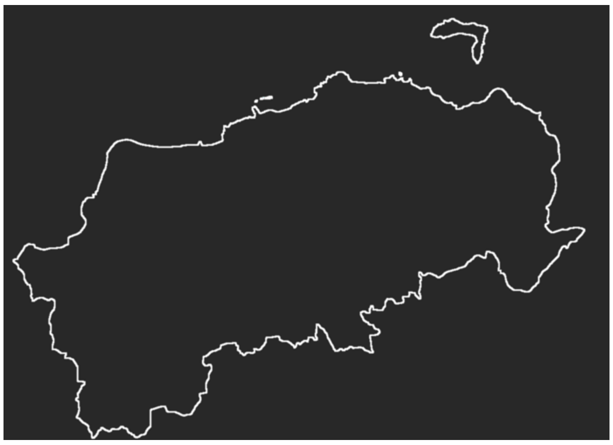


Figure 5 - Final logo design - White on dark background in this example but colour can be easily changed to fit whichever background. Designed using GIMP.

The logo has been designed so that text may be used in and around, although in this instance the minimalist outline will be used. This logo will be used as the navbar image and may, in future, be developed further to have options.

Colour scheme

At this point there are no definitive requirements for design aspects so initial thoughts are with the current colour palette for CCG site in mind. The Causeway Coast & Glens Council website uses a palette of - Jet: #333333ff; English-violet: #503250ff; English-violet-2: #6a4c6dff; Quinacridone-magenta: #85425cff; and White: #ffffffff (Fig. 6). Whilst warming and regal, this colour scheme seems inappropriate for an app surround beach safety.

Thus, taking one of these colours and using https://coolors.co, to find complementary colours, a palette was built for the BeachSmart site of – St-Patricks-blue: #23297A; Platinum: #E8E8E8; Dark Jungle Green: #0D160B; Little Boy Blue: #659DE1; and Shiny Shamrock: #66A182 (Fig. 6). It was important to have the colour scheme reference a blue theme. Initially for the obvious reason that blues and their complimentary colours remind us of the ocean but more importantly as they subconsciously create a calming atmosphere for users. Blue also tends to remind us of nature, due to the blues and greens we see each day moving through the world (PeldonRose, 2021). Blue-scale colours also appear, anecdotally, to be easier seen in low and high light conditions. Therefore, this seemed most appropriate for this project.

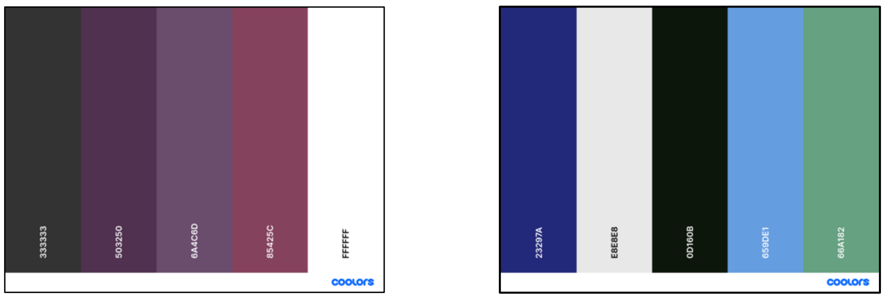


Figure 6 - CCG Homepage colour scheme (left) and chosen colour scheme designed using Coolors.co (right).

Accessibility considerations

The main accessibility concerns when designing this U.I. related to colour contrast ratios and accessibility within the code base using aria labels and semantic HTML tags. For this section, only the contrast ratios are relevant. The colour palette chosen has a contrast ratio of 15.07:1 for the most contrasting colours and 6.14:1 for the least (calculated using https://webaim.org/resources/contrastchecker/). It is important to note that the Dark Jungle Green and St Patricks Blue are not to be used together as their contrast ratio is only 1.46:1 – this is acceptable though as these are primary and secondary background colours and will only be used in conjunction with the lighter colours in the palette. A black or very dark grey to be used for text.

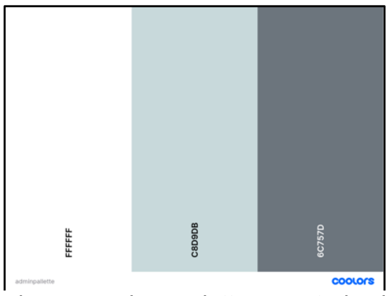


Figure 7 - Admin page colour palette - created using Coolors.co

Back-end design

The design of the back-end admin U.I. follows the same principles as mentioned above but follows a slightly simpler layout. As the purpose of the admin pages are to manage the data used by the front-end it seemed important to have as few bells and whistles as possible. It is not that less thought was put into the back end U.I. design but rather that the colour scheme and general layout required is simpler. A blue-grey scale colour scheme was selected (Fig.7) to make the U.I. clean but well contrasting, allowing administrators to do their work without distraction.

Wireframing

The next stage was wireframing. Wireframing is an important process in the design procedure. It refers to the process of creating the app layout and its various pages to show how the user journey may unfold when using the app. It is cheaper and easier to adapt than a coded prototype and allows for developers to better understand the potential UX of the app (Rees, 2022). Below are some screenshots of the more important pages from the wireframes. Shared links to the full Figma files for both front-end and back-end U.I. wireframes can be found in Appendix I.

Front-end

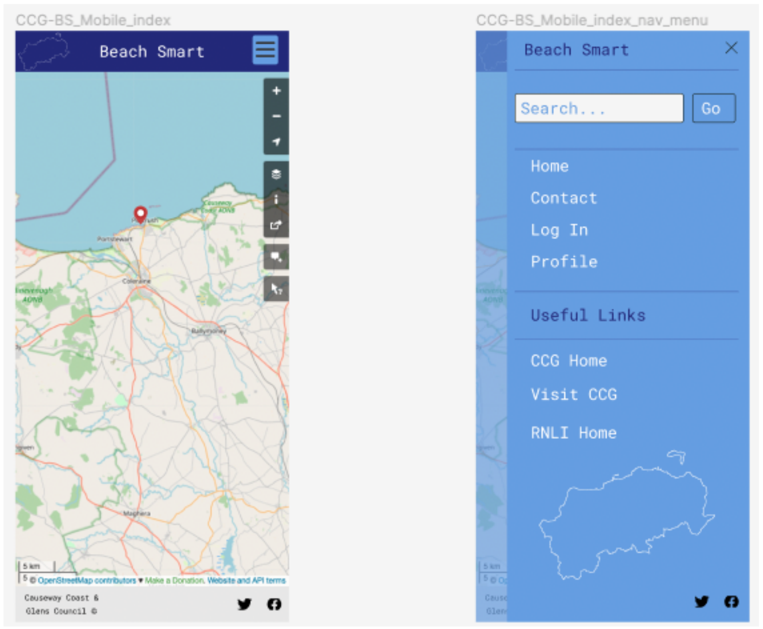
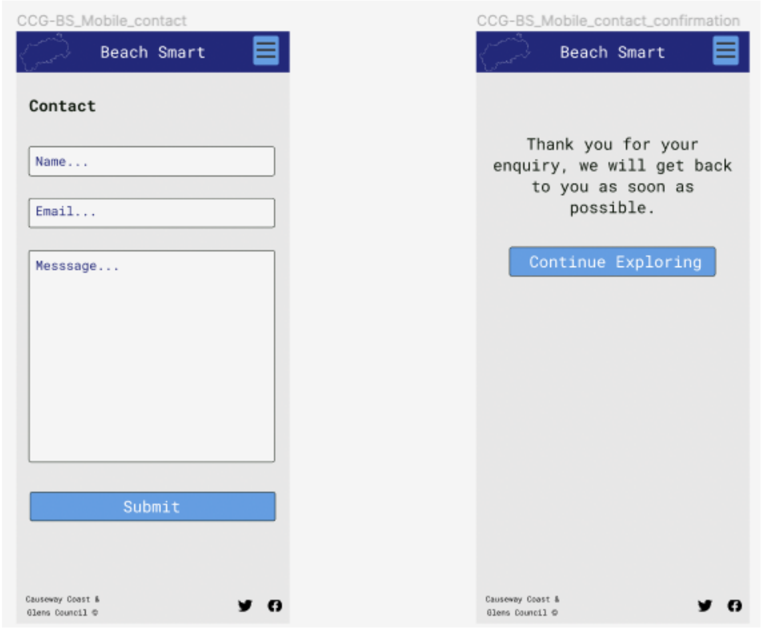


Figure 8 - BeachSmart Home and Nav dropdown



CCG-BS_Mobile_contact
Beach Smart
Contact
Name...
Email...
Messsage...
Submit
Causeway Coast & Glens Council ©
CCG-BS_Mobile_contact_confirmation
Beach Smart
Thank you for your enquiry, we will get back to you as soon as possible.
Continue Exploring
Causeway Coast & Glens Council ©

Figure 9 - BeachSmart Contact Page

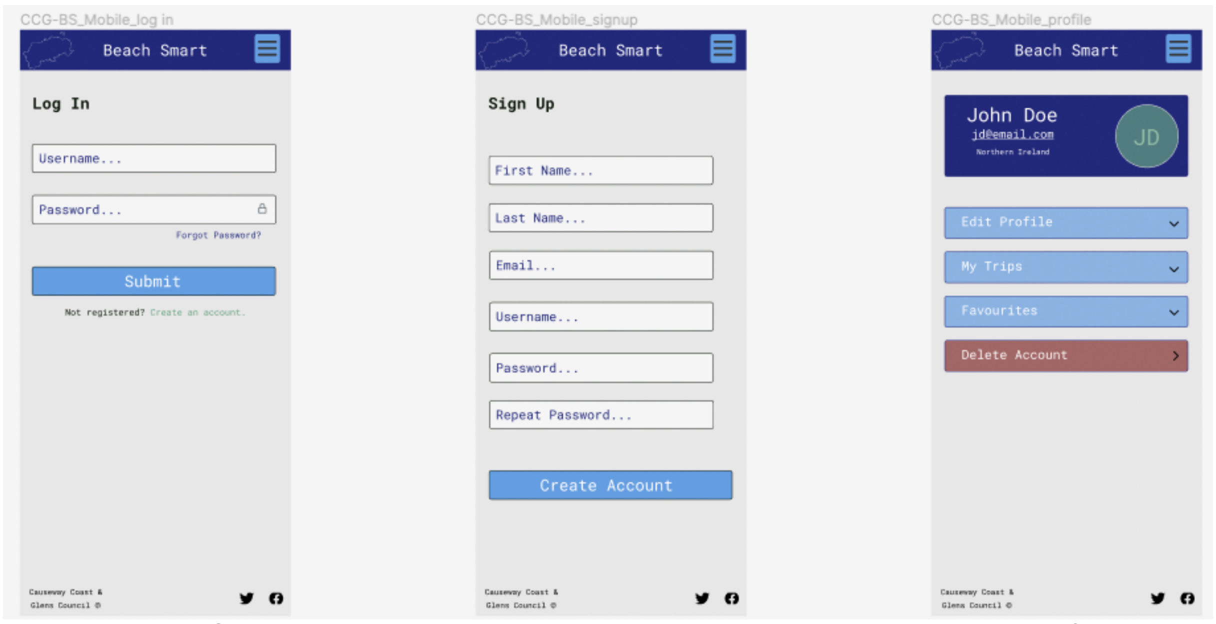
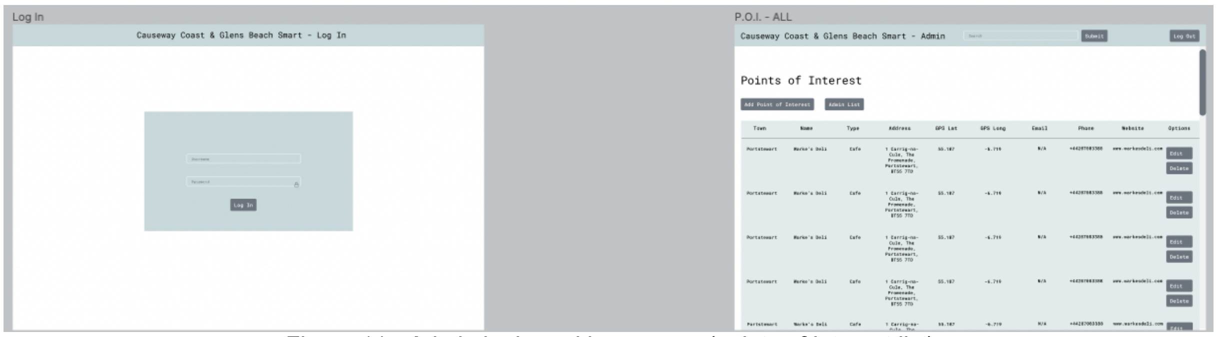


Figure 10 - BeachSmart user pages - Important to note that this may not be within the time scope of this project but was important to include in wireframe.

Back-end

Unlike the user site, the Administrator U.I. was designed with desktop in mind as administrative work tends to be done in an office on a desktop rather than on a mobile.



Town	Name	Type	Address	GPS Lat	GPS Long	Email	Phone	Website	Options
Portstewart	Morko's Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart, BT55 7TD	55.187	-6.719	N/A	+44287083388	www.morkosdeli.com	Edit Delete
Portstewart	Morko's Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart, BT55 7TD	55.187	-6.719	N/A	+44287083388	www.morkosdeli.com	Edit Delete
Portstewart	Morko's Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart, BT55 7TD	55.187	-6.719	N/A	+44287083388	www.morkosdeli.com	Edit Delete
Portstewart	Morko's Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart, BT55 7TD	55.187	-6.719	N/A	+44287083388	www.morkosdeli.com	Edit Delete
Portstewart	Morko's Deli	Cafe	1 Carrig-na-	55.187	-6.719	N/A	+44287083388	www.morkosdeli.com	Edit

Figure 11 - Admin login and homepage (points of interest list)

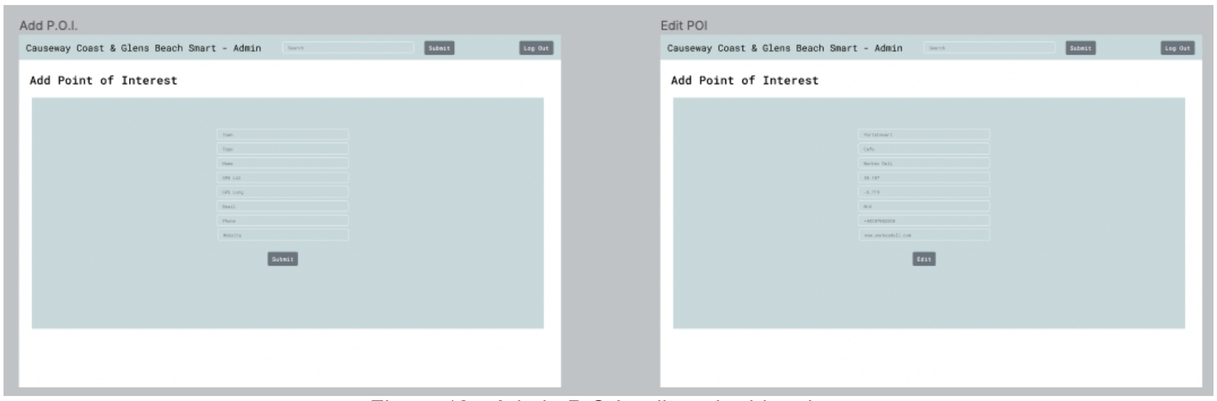
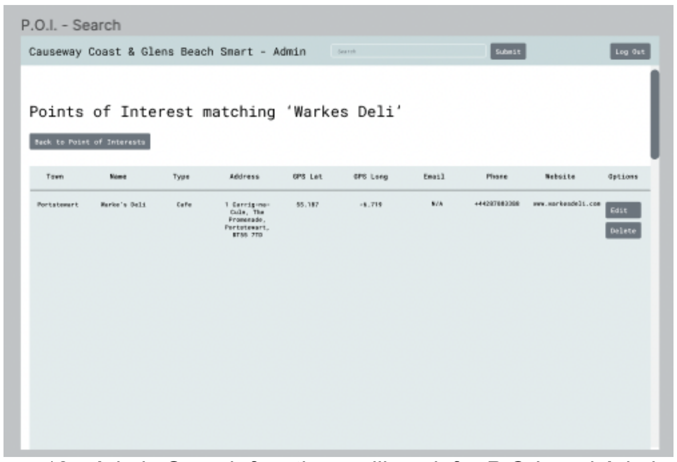


Figure 12 - Admin P.O.I. edit and add options

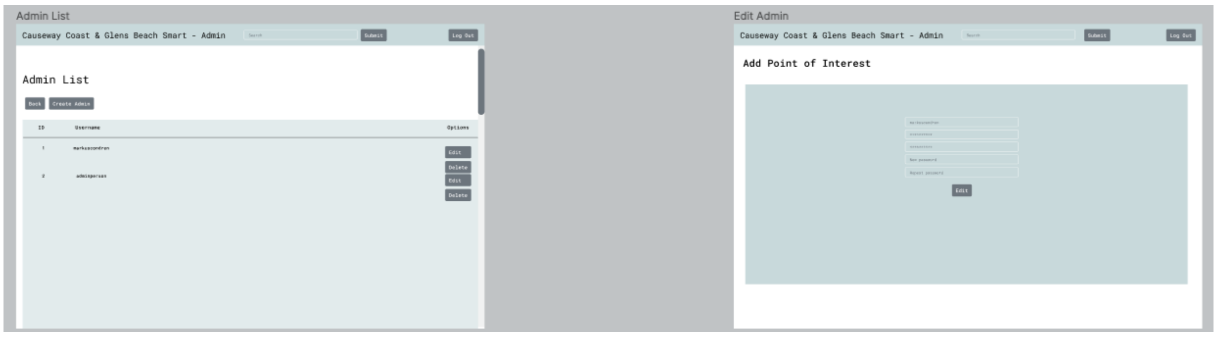

P.O.I. - Search

Causeway Coast & Glens Beach Smart - Admin

Points of Interest matching 'Warkes Deli'

Town	Name	Type	Address	GPS Lat	GPS Long	Email	Phone	Website	Options
Portstewart	Warke's Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart, BT55 7TD	55.187	-6.719	N/A	+442870833300	www.warkesdeli.com	Edit Delete

Figure 13 - Admin Search function - will work for P.O.I. and Admin list.


Admin List

Causeway Coast & Glens Beach Smart - Admin

Admin List

ID	Username	Options
1	markusconران	Edit Delete
2	adminperson	Edit Delete

Edit Admin

Causeway Coast & Glens Beach Smart - Admin

Add Point of Interest

Figure 14 - Admin list and edit admin option

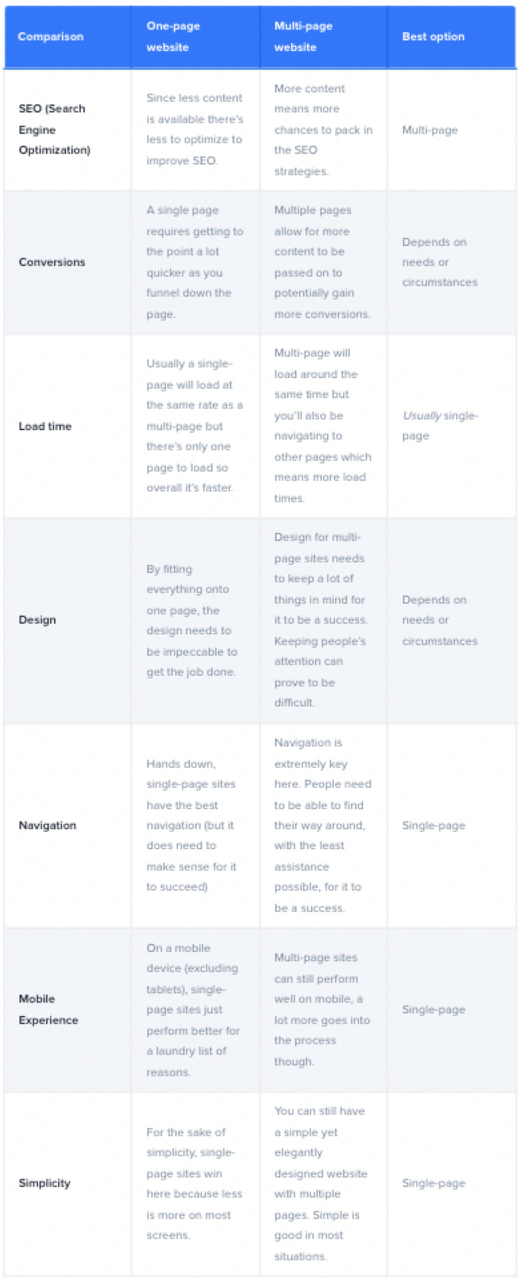
Chapter 3: Architecture design and algorithm explanation

When considering the architecture of this project, several factors were considered.

Front-end

It was decided that a single-page architecture would be most appropriate for this project. As the focus of the functionality surrounds the Map, having multiple extra pages could prove a distraction in the immediate term. Even though in the initial wireframing there are multiple pages for the likes of contact pages, for example, it is likely that these may be implemented to make use of the users contact apps on their phone, tablet or desktop. This allows for the map to remain unchanged when perusing the additional links or attempting to contact the council. This is perhaps more relevant to a discussion surround UX but was part of the thinking when deciding on a single-page structure. React is often seen as an excellent choice for large apps due to its ability to use global state variables, the simplicity with which one can expand and test a project and the fact that it is component based (Gleb, 2022).

Another, perhaps more important reason behind deciding on a single-page application is to do with reloading times. Since the app only requires a single HTML page to load, loading times are considerably quicker. This is particularly important in applications that require resource intensive actions such as rendering map tile images or sending multiple API calls to dynamically update the map display (CoreUI, 2022). A table of comparison can be seen in figure 15, this highlights that for the metrics important to this app; loading times, mobile UX and accessibility friendly navigation, a single-page architecture is most appropriate.

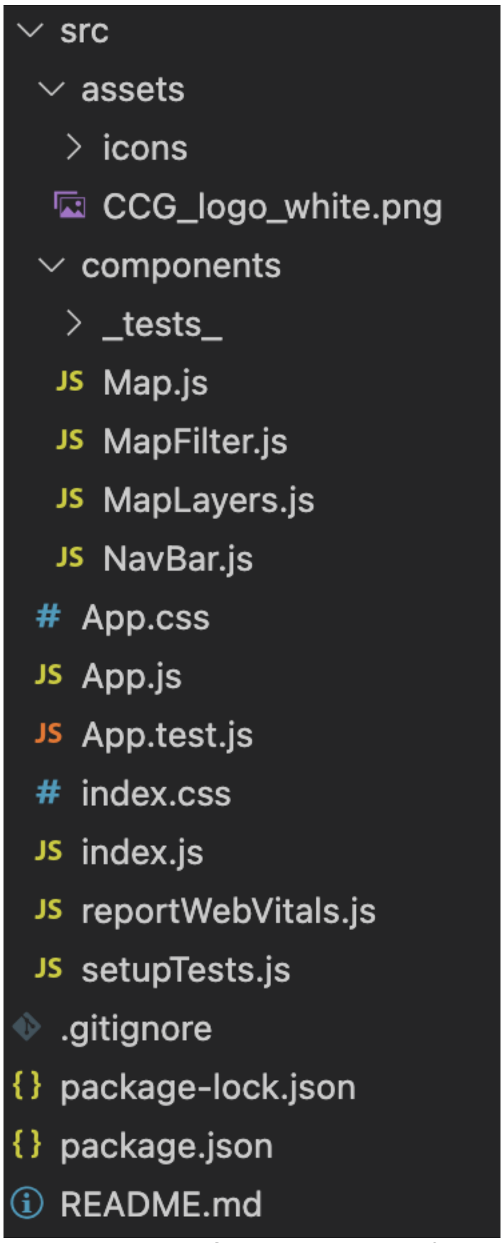


Comparison	One-page website	Multi-page website	Best option
SEO (Search Engine Optimization)	Since less content is available there's less to optimize to improve SEO.	More content means more chances to pack in the SEO strategies.	Multi-page
Conversions	A single page requires getting to the point a lot quicker as you funnel down the page.	Multiple pages allow for more content to be passed on to potentially gain more conversions.	Depends on needs or circumstances
Load time	Usually a single-page will load at the same rate as a multi-page but there's only one page to load so overall it's faster.	Multi-page will load around the same time but you'll also be navigating to other pages which means more load times.	*Usually* single-page
Design	By fitting everything onto one page, the design needs to be impeccable to get the job done.	Design for multi-page sites needs to keep a lot of things in mind for it to be a success. Keeping people's attention can prove to be difficult.	Depends on needs or circumstances
Navigation	Hands down, single-page sites have the best navigation (but it does need to make sense for it to succeed)	Navigation is extremely key here. People need to be able to find their way around, with the least assistance possible, for it to be a success.	Single-page
Mobile Experience	On a mobile device (excluding tablets), single-page sites just perform better for a laundry list of reasons.	Multi-page sites can still perform well on mobile, a lot more goes into the process though.	Single-page
Simplicity	For the sake of simplicity, single-page sites win here because less is more on most screens.	You can still have a simple yet elegantly designed website with multiple pages. Simple is good in most situations.	Single-page

Figure 15 - One-page website vs multiple pages comparison. Table taken from https://slickplan.com/blog/one-page-website-vs-multiple-pages#:~:text=Usually%20a%20single%2Dpage%20will,which%20means%20more%20load%20times. All rights reserved by original creator.

As it had been decided that the front-end was to be developed using React, the front end would naturally be component based. This means that, where possible, each individual feature is abstracted into a self-contained component, the name of which is then referenced in the App.js file which is ultimately rendered in the browser. There are several benefits to this type of file structure, some of which have been touched on already. Components form the 'building-blocks' of the app and allow for reusability, extensibility, replaceability, encapsulation, and independence (Gillin, 2022).

Reusability: once a component has been written, e.g., a log in box, it can be used in multiple places in a project (for users and admin for example), or can be used in multiple projects, without the need for any major changes to be made other than maybe specific references to files. Extensibility: two or more components may be used in conjunction with each other in the same or a different project. Say, for example, components have been created for a navbar and a search field, these can easily be combined to allow searches to be made within the navbar, as well as elsewhere in the project using the same component. Replaceability: if one has two components that carry out similar functions, they can be interchanged easily, allowing for the more appropriate component to be used. Encapsulation: an important concept in programming, this refers to the fact that when a component is created, the internal logic detail is hidden within the file and an interface is used to convey the component functionality wherever it is placed within the project. Independence: unless designed specifically to do so, components tend not to have high levels of dependency with other components. This is important when it comes to maintaining and further developing a project as a component written in one project can easily be used in a different context, either in a different place in the project or in a different project all together. The general file structure for the front-end source folder can be seen in figure 16.



```
∨ src
  ∨ assets
    > icons
      CCG_logo_white.png
  ∨ components
    > _tests_
      Map.js
      MapFilter.js
      MapLayers.js
      NavBar.js
    App.css
    App.js
    App.test.js
    index.css
    index.js
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

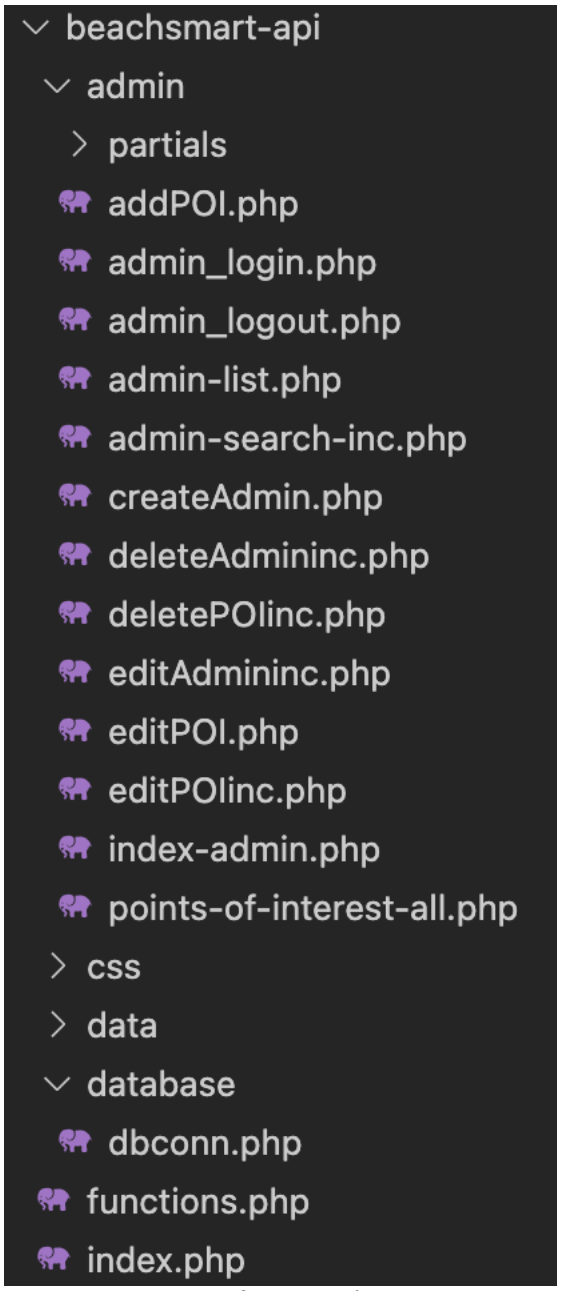
Figure 16 - BeachSmart Front-end src folder

Back-end API

The back end for this project is to be built as a REST API. A REST API follows specific architectural patterns such as having stateless communication or being able to cache data (Donovan et al., 2022). REST APIs receives requests in the form of JavaScript Object Notation (JSON), this is done via HTTP get and post requests sent from the client to the server. The API will then connect to the database, either locally or remotely, gather the required data and return the data in JSON form. Filtering of the data can either be done on the client or server.

It was important to have the API totally separate from the web app itself. This allows for modular separation of the two, keeping these separate can improve the readability and testing of both app and API (Washburn, 2019), which in turn should make it more easily maintained and developed in the future. Having a separate API also makes it reusable, this is perhaps not as likely to be a factor with this project's data as it is specific but nevertheless a standalone API can be used for other projects or by others. The final evolution of the API would be to generate random API keys and allow other developers to access the data stored in the BeachSmart database.

In this case, the API also houses the administrator front-end and logic files (Fig. 17).



```
beachsmart-api
  admin
    partials
    addPOI.php
    admin_login.php
    admin_logout.php
    admin-list.php
    admin-search-inc.php
    createAdmin.php
    deleteAdmininc.php
    deletePOIinc.php
    editAdmininc.php
    editPOI.php
    editPOIinc.php
    index-admin.php
    points-of-interest-all.php
  css
  data
  database
    dbconn.php
  functions.php
  index.php
```

Figure 17 - BeachSmart API folder structure.

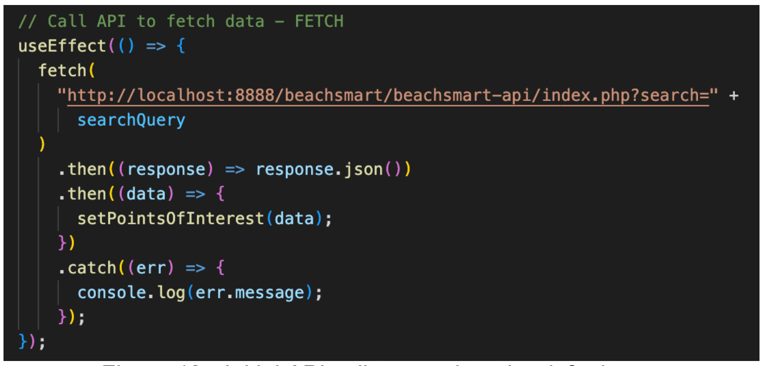
Chapter 4: Implementation

As this project did not include any significant experimentation, this chapter aims to describe the way in which the main components of the application have been implemented in code.

Front-end

Populating data from API

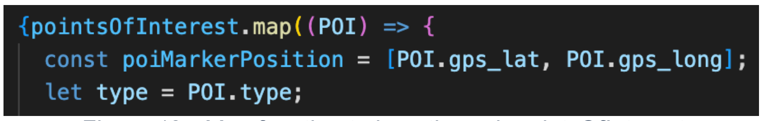
Upon first rendering, a useEffect fetch call is made the API endpoint. This invokes the returnMapData function (see below) which returns a JSON object containing all the data in the database relating to points of interest.



```
// Call API to fetch data - FETCH
useEffect(() => {
  fetch(
    "http://localhost:8888/beachsmart/beachsmart-api/index.php?search=" +
      searchQuery
  )
    .then((response) => response.json())
    .then((data) => {
      setPointsOfInterest(data);
    })
    .catch((err) => {
      console.log(err.message);
    });
});
```

Figure 18 - Initial API call to populate the default map

Once the data has been received by the front-end, it is stored in a useState constant; pointsOfInterest. This allows the data to be looped through using the JavaScript map function (Fig.19). Here the GPS coordinates are taken from the map function and combined as an array, to be included as a constant to define the position on the map for each point of interest. Type is also defined here as POI.type, this is to facilitate different map pin icons to be used dependent on point of interest type (Fig. 22).

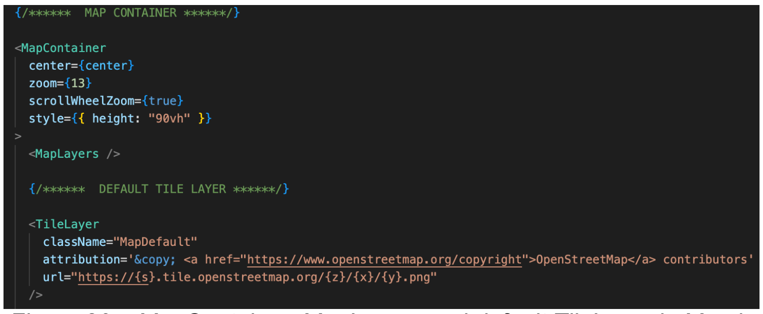


```
{pointsOfInterest.map((POI) => {
  const poiMarkerPosition = [POI.gps_lat, POI.gps_long];
  let type = POI.type;
```

Figure 19 - Map function to loop through pointsOfInterest

Map

Map.js holds the map container and all the relevant map information. The map is defined using the <MapContainer> tag from react bootstrap, this dictates the centre point at which the map is first rendered and defines zoom level and the ability for a user to scroll through the map. Below this, the MapLayers component is called which holds the authorisation information for adding additional map overlays, e.g., dark mode, to the map. Finally, we see the default tile layer, this is OSM's base map layer and is required before any additional overlays may be added (Fig. 20).

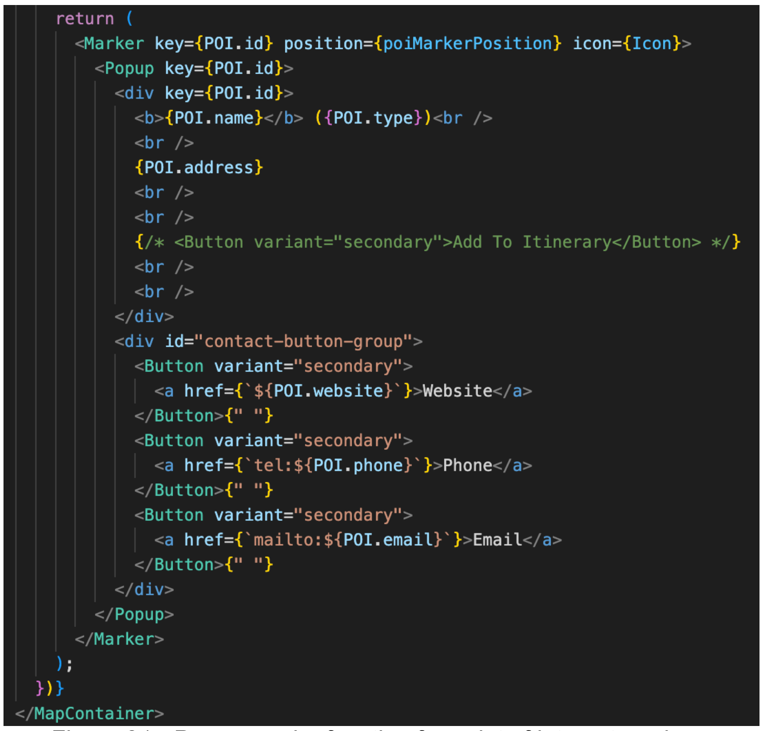


```
{/******  MAP CONTAINER ******/}

<MapContainer
  center={center}
  zoom={13}
  scrollWheelZoom={true}
  style={{ height: "90vh" }}
>
  <MapLayers />

  {/******  DEFAULT TILE LAYER ******/}

  <TileLayer
    className="MapDefault"
    attribution='&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
    url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
  />
```

Figure 20 – MapContainer, MapLayers and default TileLayer in Map.js

Pin icons

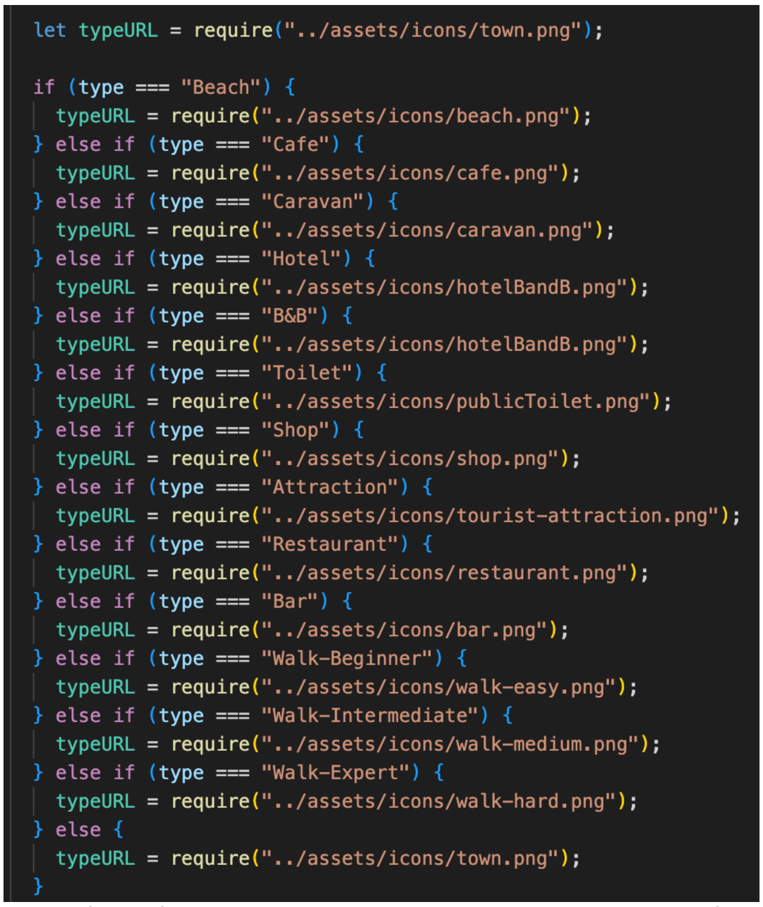
The data gathered by the fetch request in figure 18, once looped through, can then be displayed on the map. This is done by including a return statement within the map function which renders a popup for each point of interest and populates this with the required information for each point of interest (Fig. 21), this all occurs within the MapContainer tag. The button tags for Phone and Email open the user's relevant app, prepopulated with phone number or email respectively, to allow quick and easy contact options even for those who are less technically inclined.



```
      return (
        <Marker key={POI.id} position={poiMarkerPosition} icon={Icon}>
          <Popup key={POI.id}>
            <div key={POI.id}>
              <b>{POI.name}</b> ({POI.type})<br />
              <br />
              {POI.address}
              <br />
              <br />
              {/* <Button variant="secondary">Add To Itinerary</Button> */}
              <br />
              <br />
            </div>
            <div id="contact-button-group">
              <Button variant="secondary">
                <a href={`${POI.website}`}>Website</a>
              </Button>{" "}
              <Button variant="secondary">
                <a href={`tel:${POI.phone}`}>Phone</a>
              </Button>{" "}
              <Button variant="secondary">
                <a href={`mailto:${POI.email}`}>Email</a>
              </Button>{" "}
            </div>
          </Popup>
        </Marker>
      );
    })}
</MapContainer>
```

Figure 21 - Popup render function for point of interest markers.

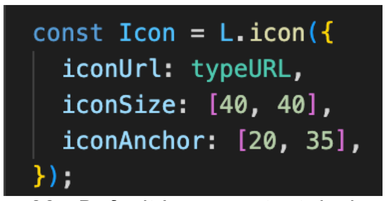
Map icons were downloaded as PNGs from www.flaticon.com/ and stored in assets folder in react src folder. Within the map function for looping through the points of interest data gathered from the API, there is if-else-if statement to ensure the correct icon is chosen for each type of attraction. This uses the variable *type* (Fig. 19) to change the typeURL of each point of interest to match the correct icon in the assets/icons folder (Fig. 22).



```
let typeURL = require("../assets/icons/town.png");

if (type === "Beach") {
  typeURL = require("../assets/icons/beach.png");
} else if (type === "Cafe") {
  typeURL = require("../assets/icons/cafe.png");
} else if (type === "Caravan") {
  typeURL = require("../assets/icons/caravan.png");
} else if (type === "Hotel") {
  typeURL = require("../assets/icons/hotelBandB.png");
} else if (type === "B&B") {
  typeURL = require("../assets/icons/hotelBandB.png");
} else if (type === "Toilet") {
  typeURL = require("../assets/icons/publicToilet.png");
} else if (type === "Shop") {
  typeURL = require("../assets/icons/shop.png");
} else if (type === "Attraction") {
  typeURL = require("../assets/icons/tourist-attraction.png");
} else if (type === "Restaurant") {
  typeURL = require("../assets/icons/restaurant.png");
} else if (type === "Bar") {
  typeURL = require("../assets/icons/bar.png");
} else if (type === "Walk-Beginner") {
  typeURL = require("../assets/icons/walk-easy.png");
} else if (type === "Walk-Intermediate") {
  typeURL = require("../assets/icons/walk-medium.png");
} else if (type === "Walk-Expert") {
  typeURL = require("../assets/icons/walk-hard.png");
} else {
  typeURL = require("../assets/icons/town.png");
}
```

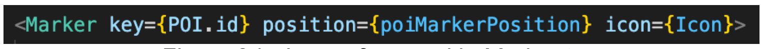
Figure 22 - If-else-if statement to assign correct pin icon to each point of interest.

To implement these as icons within the map, a default Icon constant needed to be declared (Fig. 23) which can then be referenced in the Marker tag within the map (Fig. 24).



```
const Icon = L.icon({
  iconUrl: typeURL,
  iconSize: [40, 40],
  iconAnchor: [20, 35],
});
```

Figure 23 - Default Icon constant declaration.



```
<Marker key={POI.id} position={poiMarkerPosition} icon={Icon}>
```

Figure 24 - Icon referenced in Marker tag.

Walking routes

Walking routes are shown by walking pins, of different colour depending on difficulty. These are assigned in the same way as the Icons for each point of interest using the if-else-if statement in figure 22. The actual routes were plotted in GeoJSON (Fig. 25) to highlight the route on the map, this file (portstewart-walks.geojson) is stored in the API data folder, unfortunately this was not implemented by submission (see evaluation)

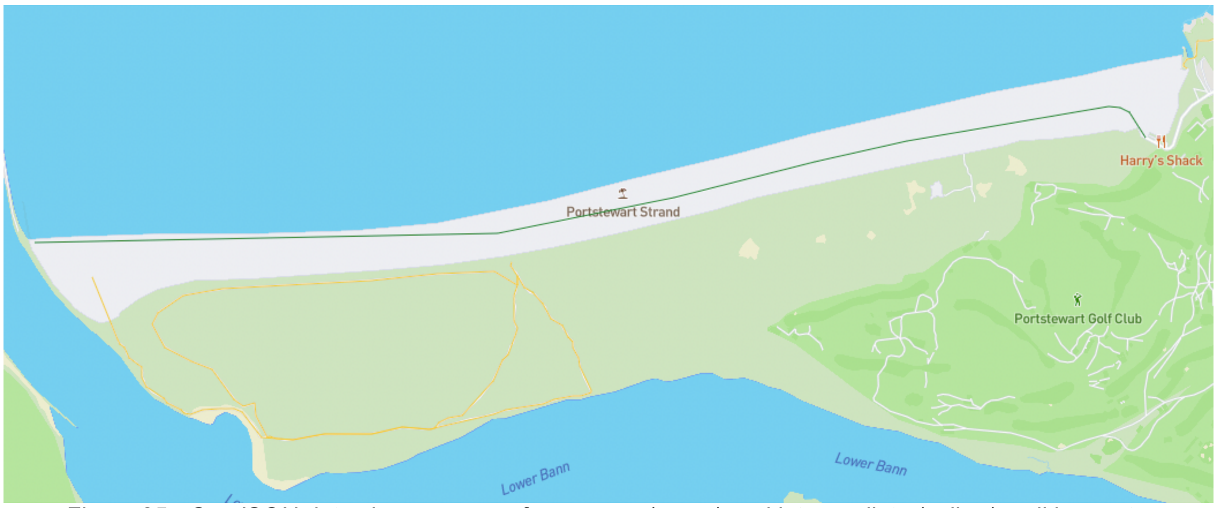


Figure 25 - GeoJSON data shown on map for an easy (green) and intermediate (yellow) walking route.

Map search and filter

Search query is saved from form, sent to server and triggers userSearchData function to return the relevant info, this updates dynamically as the user types (Fig. 26).

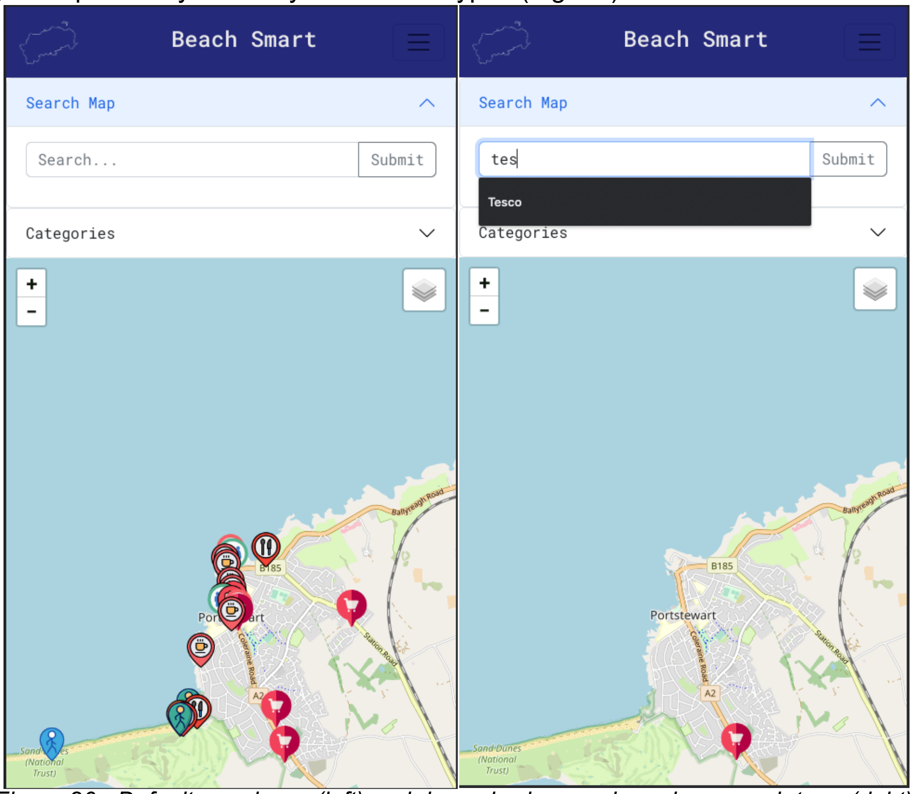


Figure 26 - Default map icons (left) and dynamic changes based on search term (right).

Initially checkboxes were preferred for filter but ended up being implemented as buttons (Fig. 27), which dynamically update the pins on the map in the same manner as the user search function.

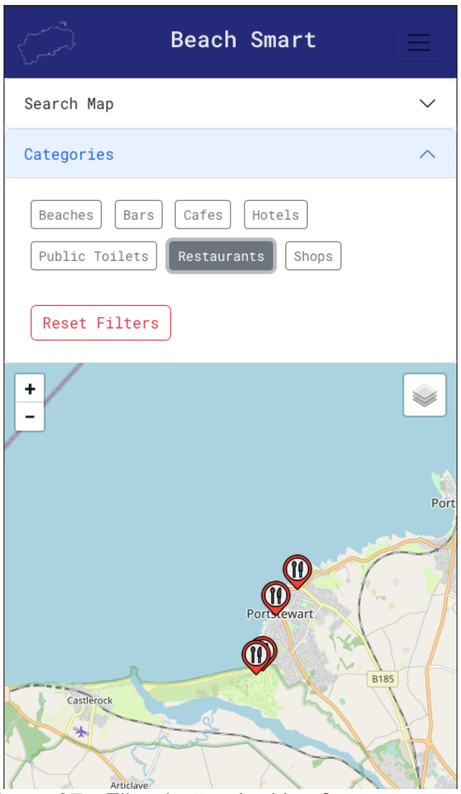
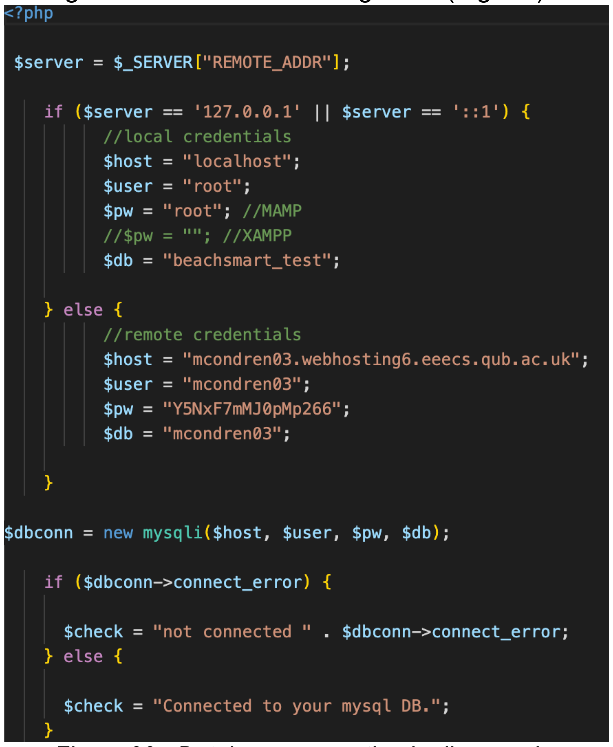


Figure 27 - Filter button looking for restaurants.

Back end

Database connection

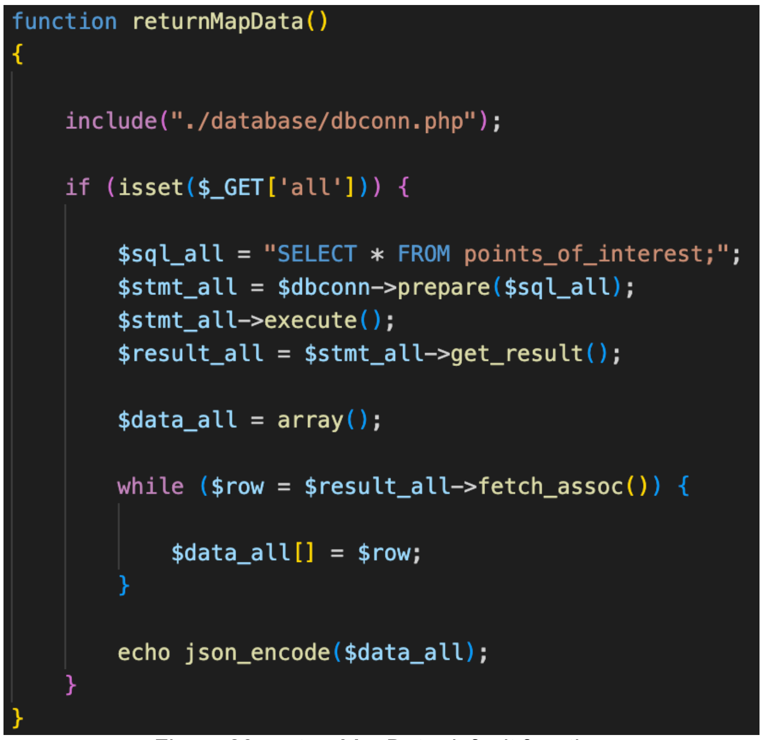
As this project was to be developed locally and hosted remotely for submission, there are several options for establishing connection to the server. The dbconn.php file has been written in such a way as to dynamically connect to the database whether on localhost or the EEEC server. This was achieved by using the $_SESSION PHP superglobal to grab the server address and change the database credentials depending on which server is being used (Fig. 28).



```php
<?php

 $server = $_SERVER["REMOTE_ADDR"];

    if ($server == '127.0.0.1' || $server == '::1') {
          //local credentials
          $host = "localhost";
          $user = "root";
          $pw = "root"; //MAMP
          //$pw = ""; //XAMPP
          $db = "beachsmart_test";

    } else {
          //remote credentials
          $host = "mcondren03.webhosting6.eeecs.qub.ac.uk";
          $user = "mcondren03";
          $pw = "Y5NxF7mMJ0pMp266";
          $db = "mcondren03";

    }

$dbconn = new mysqli($host, $user, $pw, $db);

    if ($dbconn->connect_error) {

      $check = "not connected " . $dbconn->connect_error;
    } else {

      $check = "Connected to your mysql DB.";
    }
```
Figure 28 - Database connection in dbconn.php

API Functions

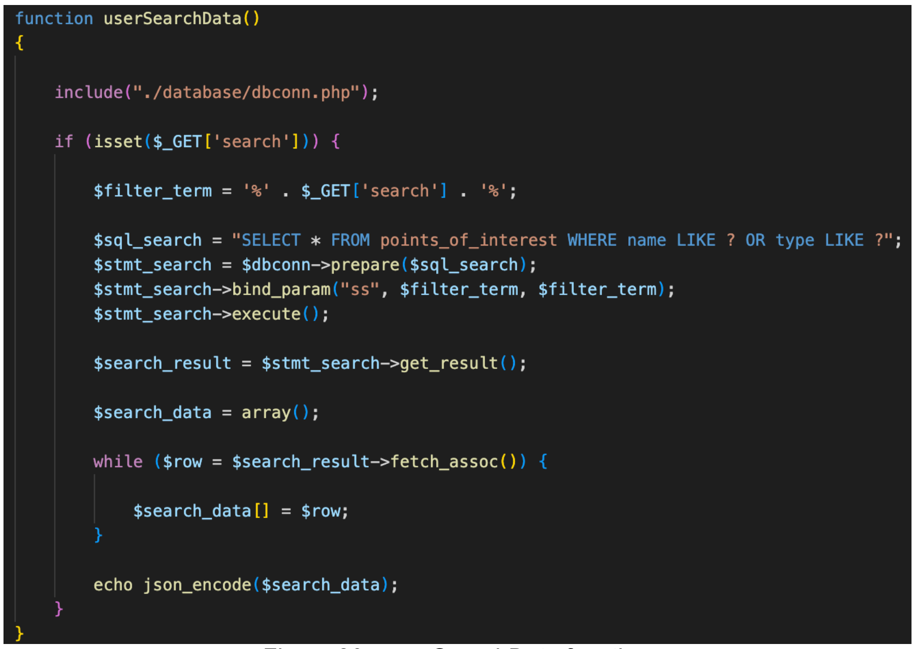
The logic for dealing with API requests are separated into a functions.php file. This kept the index page simple, calling these functions whenever the required post or get request had been made. Shown below are the returnMapData function (Fig. 29) which is responds to the first fetch request that is sent when the page is first rendered on screen (Fig. 18).



```
function returnMapData()
{

    include("./database/dbconn.php");

    if (isset($_GET['all'])) {

        $sql_all = "SELECT * FROM points_of_interest;";
        $stmt_all = $dbconn->prepare($sql_all);
        $stmt_all->execute();
        $result_all = $stmt_all->get_result();

        $data_all = array();

        while ($row = $result_all->fetch_assoc()) {

            $data_all[] = $row;
        }

        echo json_encode($data_all);
    }
}
```

Figure 29 - returnMapData default function.

The second function shown is userSearchData (Fig. 30), this function takes the value of the search bar when submitted, passes it to the API in the form of a get request and changes the filter term to whatever the user is searching for. The SQL uses a wildcard (%) search to get the points of interest which resemble the search term. This dynamically updates the pins shown on the map, and as returnMapData only fires once on render, userSearchData can continuously update what is shown on the map. The same function was used to facilitate the filter button options in figure 27.

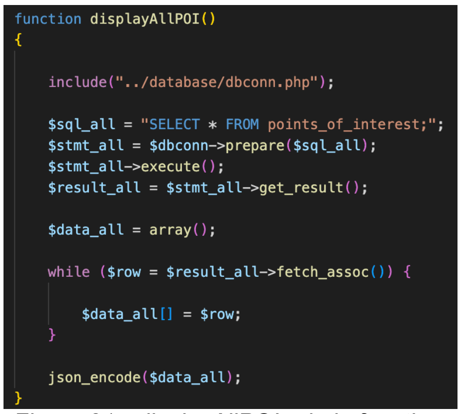


```php
function userSearchData()
{

    include("./database/dbconn.php");

    if (isset($_GET['search'])) {

        $filter_term = '%' . $_GET['search'] . '%';

        $sql_search = "SELECT * FROM points_of_interest WHERE name LIKE ? OR type LIKE ?";
        $stmt_search = $dbconn->prepare($sql_search);
        $stmt_search->bind_param("ss", $filter_term, $filter_term);
        $stmt_search->execute();

        $search_result = $stmt_search->get_result();

        $search_data = array();

        while ($row = $search_result->fetch_assoc()) {

            $search_data[] = $row;
        }

        echo json_encode($search_data);
    }
}
```

Figure 30 - userSearchData function.

Admin display

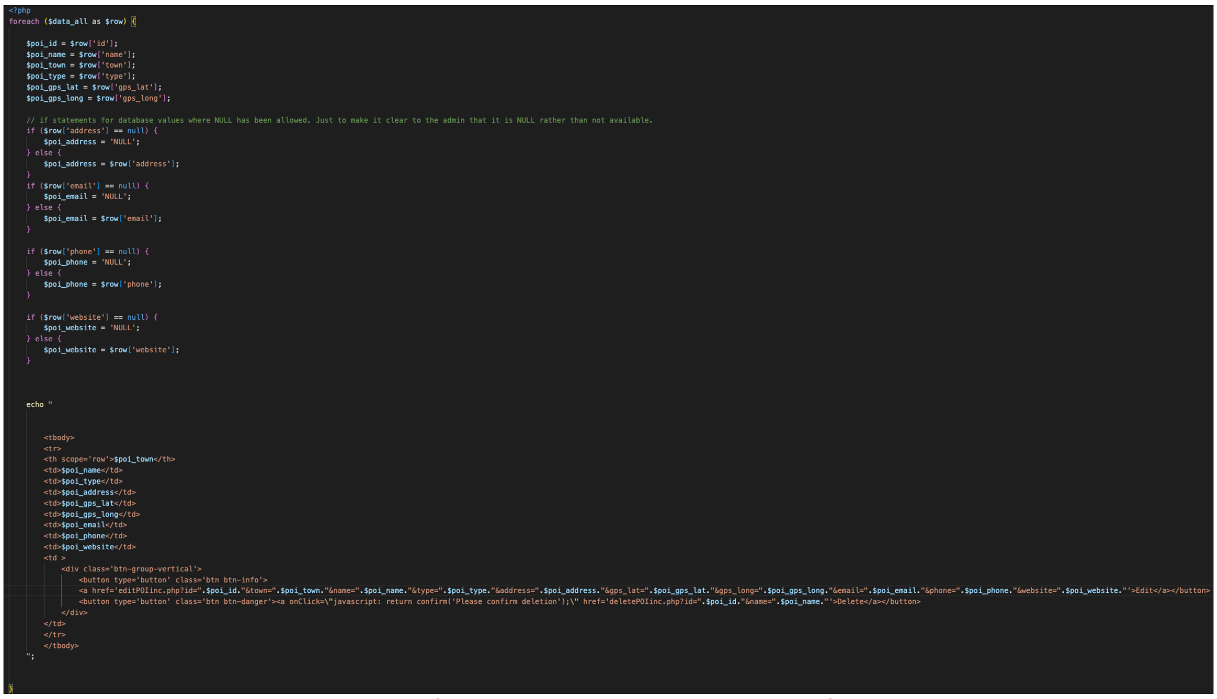
The list of Points of Interest that can be seen within the admin pages are simply gathered by SQL SELECT from the relevant part of the database (Fig. 31).



```php
function displayAllPOI()
{

    include("../database/dbconn.php");

    $sql_all = "SELECT * FROM points_of_interest;";
    $stmt_all = $dbconn->prepare($sql_all);
    $stmt_all->execute();
    $result_all = $stmt_all->get_result();

    $data_all = array();

    while ($row = $result_all->fetch_assoc()) {

        $data_all[] = $row;
    }

    json_encode($data_all);
}
```

Figure 31 - displayAllPOI admin function.

These are then rendered from PHP to be shown as a list, figure 32 shows how this occurs for the points of interest list. The list of Admins is done in the same manner.

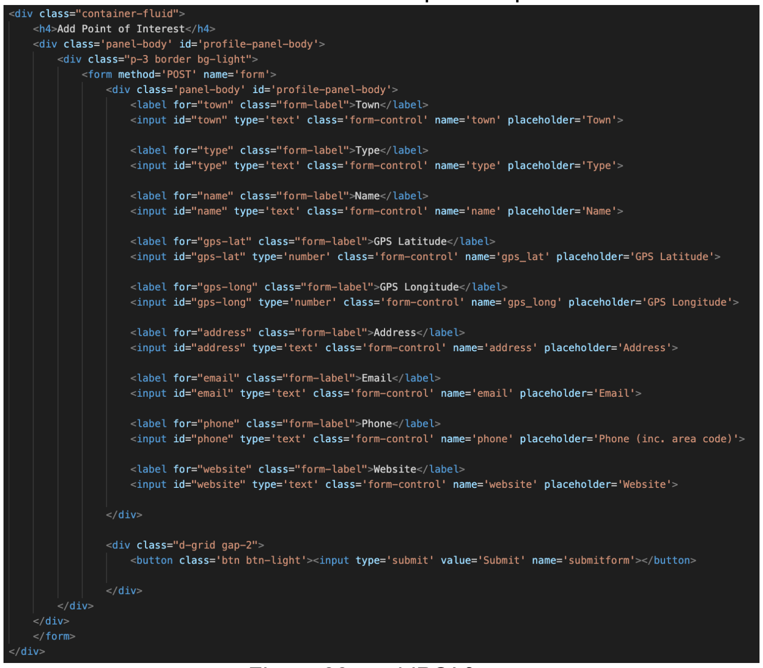


```php
<?php
foreach ($data_all as $row) {

    $poi_id = $row['id'];
    $poi_name = $row['name'];
    $poi_town = $row['town'];
    $poi_type = $row['type'];
    $poi_gps_lat = $row['gps_lat'];
    $poi_gps_long = $row['gps_long'];

    // if statements for database values where NULL has been allowed. Just to make it clear to the admin that it is NULL rather than not available.
    if ($row['address'] == null) {
        $poi_address = 'NULL';
    } else {
        $poi_address = $row['address'];
    }
    if ($row['email'] == null) {
        $poi_email = 'NULL';
    } else {
        $poi_email = $row['email'];
    }

    if ($row['phone'] == null) {
        $poi_phone = 'NULL';
    } else {
        $poi_phone = $row['phone'];
    }

    if ($row['website'] == null) {
        $poi_website = 'NULL';
    } else {
        $poi_website = $row['website'];
    }



    echo "


        <tbody>
        <tr>
        <th scope='row'>$poi_town</th>
        <td>$poi_name</td>
        <td>$poi_type</td>
        <td>$poi_address</td>
        <td>$poi_gps_lat</td>
        <td>$poi_gps_long</td>
        <td>$poi_email</td>
        <td>$poi_phone</td>
        <td>$poi_website</td>
        <td >
            <div class='btn-group-vertical'>
                <button type='button' class='btn btn-info'>
                <a href='editPOIinc.php?id=".$poi_id."&town=".$poi_town."&name=".$poi_name."&type=".$poi_type."&address=".$poi_address."&gps_lat=".$poi_gps_lat."&gps_long=".$poi_gps_long."&email=".$poi_email."&phone=".$poi_phone."&website=".$poi_website."'>Edit</a></button>
                <button type='button' class='btn btn-danger'><a onClick=\"javascript: return confirm('Please confirm deletion');\" href='deletePOIinc.php?id=".$poi_id."&name=".$poi_name."'>Delete</a></button>
            </div>
        </td>
        </tr>
        </tbody>
    ";


}
```

Figure 32 - Points of Interest being displayed on points-of-interest-all.php

P.O.I admin functions

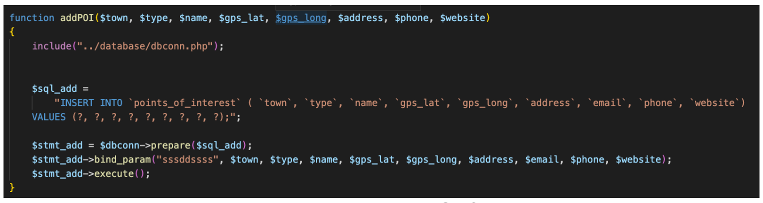
Administrators for BeachSmart have three main tasks in relation to managing the points of interest and admins from the admin page: add, edit and delete. These are taken care of by individual functions within functions.php. The functions for adding, editing and deleting are the same for points of interest and admins, just with different values – therefore just point of interest functions shown for the sake of example. To add a point of interest, the administrator fills the form from addPOI.php (Fig. 33), upon submission, this form data is sent to the backend via a post request.



```html
<div class="container-fluid">
    <h4>Add Point of Interest</h4>
    <div class='panel-body' id='profile-panel-body'>
        <div class="p-3 border bg-light">
            <form method='POST' name='form'>
                <div class='panel-body' id='profile-panel-body'>
                    <label for="town" class="form-label">Town</label>
                    <input id="town" type='text' class='form-control' name='town' placeholder='Town'>

                    <label for="type" class="form-label">Type</label>
                    <input id="type" type='text' class='form-control' name='type' placeholder='Type'>

                    <label for="name" class="form-label">Name</label>
                    <input id="name" type='text' class='form-control' name='name' placeholder='Name'>

                    <label for="gps-lat" class="form-label">GPS Latitude</label>
                    <input id="gps-lat" type='number' class='form-control' name='gps_lat' placeholder='GPS Latitude'>

                    <label for="gps-long" class="form-label">GPS Longitude</label>
                    <input id="gps-long" type='number' class='form-control' name='gps_long' placeholder='GPS Longitude'>

                    <label for="address" class="form-label">Address</label>
                    <input id="address" type='text' class='form-control' name='address' placeholder='Address'>

                    <label for="email" class="form-label">Email</label>
                    <input id="email" type='text' class='form-control' name='email' placeholder='Email'>

                    <label for="phone" class="form-label">Phone</label>
                    <input id="phone" type='text' class='form-control' name='phone' placeholder='Phone (inc. area code)'>

                    <label for="website" class="form-label">Website</label>
                    <input id="website" type='text' class='form-control' name='website' placeholder='Website'>

                </div>

                <div class="d-grid gap-2">
                    <button class='btn btn-light'><input type='submit' value='Submit' name='submitform'></button>

                </div>
        </div>
    </div>
    </form>
</div>
```

Figure 33 – addPOI form.

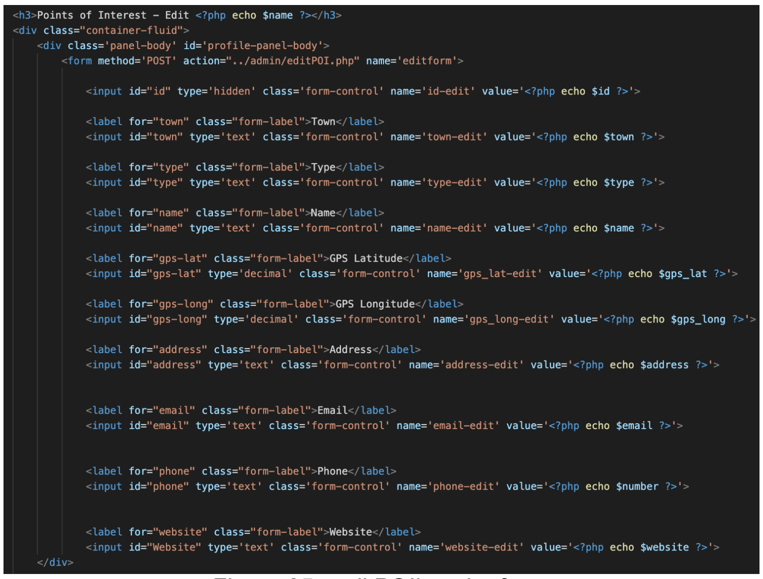
The form contents are passed into the addPOI function (Fig. 34) which sends a prepared SQL statement to the database to create the new point of interest.



```php
function addPOI($town, $type, $name, $gps_lat, $gps_long, $address, $phone, $website)
{
    include("../database/dbconn.php");


    $sql_add =
        "INSERT INTO `points_of_interest` ( `town`, `type`, `name`, `gps_lat`, `gps_long`, `address`, `email`, `phone`, `website`)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);";

    $stmt_add = $dbconn->prepare($sql_add);
    $stmt_add->bind_param("sssddssss", $town, $type, $name, $gps_lat, $gps_long, $address, $email, $phone, $website);
    $stmt_add->execute();
}
```

Figure 34 – API addPOI function.

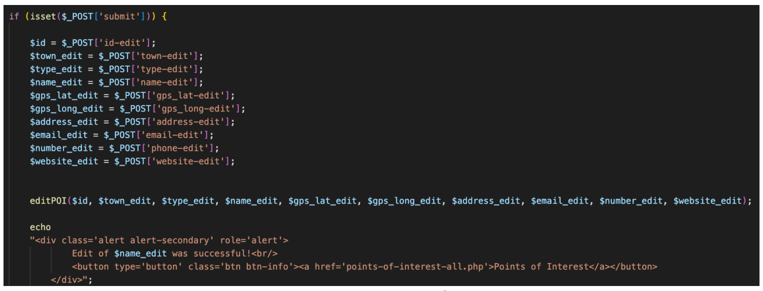
The edit function works in a similar way, when the admin selects the Edit button next to a point of interest (or admin), the information relating to id, name etc of that point of interest is passed to editPOIinc.php where it is displayed as values in a form (Fig. 35) – this allows the admin to see what current values are stored for the point of interest so they need only change those which are incorrect.



```html
<h3>Points of Interest - Edit <?php echo $name ?></h3>
<div class="container-fluid">
    <div class='panel-body' id='profile-panel-body'>
        <form method='POST' action="../admin/editPOI.php" name='editform'>

            <input id="id" type='hidden' class='form-control' name='id-edit' value='<?php echo $id ?>'>

            <label for="town" class="form-label">Town</label>
            <input id="town" type='text' class='form-control' name='town-edit' value='<?php echo $town ?>'>

            <label for="type" class="form-label">Type</label>
            <input id="type" type='text' class='form-control' name='type-edit' value='<?php echo $type ?>'>

            <label for="name" class="form-label">Name</label>
            <input id="name" type='text' class='form-control' name='name-edit' value='<?php echo $name ?>'>

            <label for="gps-lat" class="form-label">GPS Latitude</label>
            <input id="gps-lat" type='decimal' class='form-control' name='gps_lat-edit' value='<?php echo $gps_lat ?>'>

            <label for="gps-long" class="form-label">GPS Longitude</label>
            <input id="gps-long" type='decimal' class='form-control' name='gps_long-edit' value='<?php echo $gps_long ?>'>

            <label for="address" class="form-label">Address</label>
            <input id="address" type='text' class='form-control' name='address-edit' value='<?php echo $address ?>'>


            <label for="email" class="form-label">Email</label>
            <input id="email" type='text' class='form-control' name='email-edit' value='<?php echo $email ?>'>


            <label for="phone" class="form-label">Phone</label>
            <input id="phone" type='text' class='form-control' name='phone-edit' value='<?php echo $number ?>'>


            <label for="website" class="form-label">Website</label>
            <input id="Website" type='text' class='form-control' name='website-edit' value='<?php echo $website ?>'>
    </div>
```

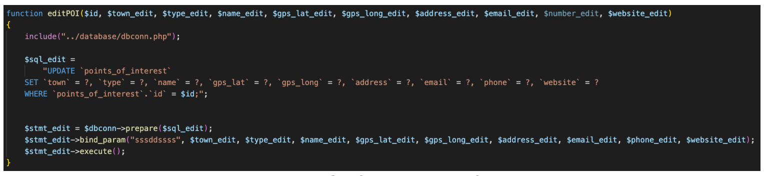
Figure 35 - editPOIinc.php form.

Once submitted, the values are passed to editPOI.php (Fig. 36) which calls the editPOI function to make the change in the database (Fig. 37), the user is then prompted that the edit was successful, if unsuccessful they will be funnelled back to points-of-interest-all.php to try again (Fig.36).



```php
if (isset($_POST['submit'])) {

    $id = $_POST['id-edit'];
    $town_edit = $_POST['town-edit'];
    $type_edit = $_POST['type-edit'];
    $name_edit = $_POST['name-edit'];
    $gps_lat_edit = $_POST['gps_lat-edit'];
    $gps_long_edit = $_POST['gps_long-edit'];
    $address_edit = $_POST['address-edit'];
    $email_edit = $_POST['email-edit'];
    $number_edit = $_POST['phone-edit'];
    $website_edit = $_POST['website-edit'];


    editPOI($id, $town_edit, $type_edit, $name_edit, $gps_lat_edit, $gps_long_edit, $address_edit, $email_edit, $number_edit, $website_edit);

    echo
    "<div class='alert alert-secondary' role='alert'>
            Edit of $name_edit was successful!<br/>
            <button type='button' class='btn btn-info'><a href='points-of-interest-all.php'>Points of Interest</a></button>
        </div>";
```

Figure 36 - editPOI.php



```php
function editPOI($id, $town_edit, $type_edit, $name_edit, $gps_lat_edit, $gps_long_edit, $address_edit, $email_edit, $number_edit, $website_edit)
{
    include("../database/dbconn.php");

    $sql_edit =
        "UPDATE `points_of_interest`
    SET `town` = ?, `type` = ?, `name` = ?, `gps_lat` = ?, `gps_long` = ?, `address` = ?, `email` = ?, `phone` = ?, `website` = ?
    WHERE `points_of_interest`.`id` = $id;";


    $stmt_edit = $dbconn->prepare($sql_edit);
    $stmt_edit->bind_param("sssddssss", $town_edit, $type_edit, $name_edit, $gps_lat_edit, $gps_long_edit, $address_edit, $email_edit, $phone_edit, $website_edit);
    $stmt_edit->execute();
}
```

Figure 37 - editPOI function in functions.php.

Admin Log In

It is important to discuss the log in function as admin passwords are encrypted using PHP built-in PASSWORD_DEFAULT encryption and stored in the database as varbinary. It is important to mention as this has been done simply to attempt to have some basic authentication within the project, this is by no means suggesting that the encryption is strong or that the log in system would hold up to any significant scrutiny.

It is used purely to demonstrate a vital aspect of running an online service. Such important data encryption of data between clients and servers is taken care of by Hypertext Transfer Protocol Secure (HTTPS) in conjunction with Secure Socket Layer (SSL) protocol, one should probably not use the encryption methods used in this project to secure confidential data unless it has been rigorously tested and confirmed as appropriate.

Database

The database for BeachSmart was designed and built using the PHPMyAdmin interface, using MAMP as a localhost. This was then uploaded to the EEECS server using FileZilla.

It was decided that using a relational database was most appropriate for this project. A basic entity discovery plan was carried out (Table. 2) to discover the basic relationships between the data, once complete, normalisation was to be considered.

Table 2 - Basic entity discovery of main Entities to be explored in this project.

Points of Interest	Administrator
• Name • Type • Address • GPS Latitude • GPS Longitude • Website • Phone • Email • Opening hours • Images	• Username • Password • Profile image (potential)

Normalisation

Important in designing a database is the maintenance of uniqueness within the data. This helps to minimise the amount of redundant data. This is particularly of concern in relational database systems such as this as unnecessarily repeated data can lead to inconsistencies in the stored data whilst increasing the potential for corruption. Normalisation is the process of separating entities into different tables following their normal forms to achieve this.

Edgar Codd (1970, 1971) pioneered the concept of first, second and third normal forms of data (1NF, 2NF, 3NF). These rules are as follows: - 1NF – Table should contain only atomic values, of the same domain. Columns should have unique names and the order in which data is stored is inconsequential. - 2NF – Table should be in 1NF and not have partial dependency. - 3NF – Table be in 2NF and not have transitive dependency. Normalisation can minimise any add, delete, or edit anomalies, reduce the need for database restructuring upon the addition of new data types, avoid bias when executing queries whilst ultimately allowing for a simpler and more informative database for developer and user alike (ITLEducationSolutions, 2009).

Data redundancies may, in some instances, improve the performance of SQL queries used as it can minimise the number of joins required to search through the data (MicroStrategy, 2022), this should be considered on an individual basis and the performance improvement weighed up against the disadvantages stated above. This is unlikely to be this case due to the scale of this project.

The practical consideration of a larger database and increased associated maintenance cost should be considered for future development of the project, therefore 3 normalisation options were explored for this project (Fig. 38, 39, 40).

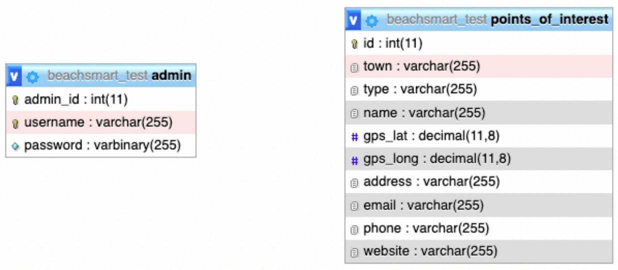


Figure 38 - No Normalisation of database data

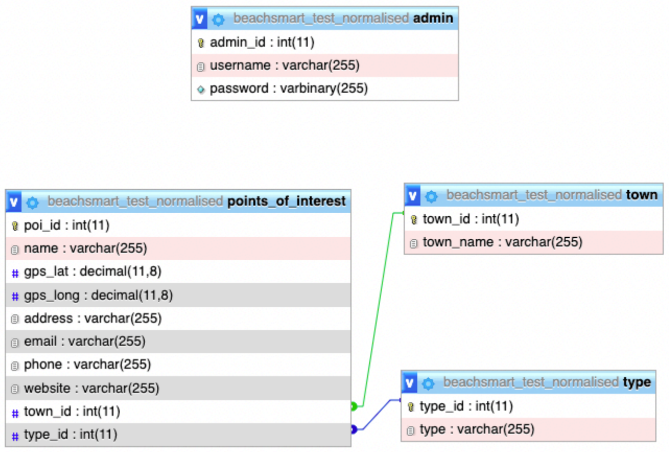


Figure 39 - Simple normalisation of data to include normalised town and type.

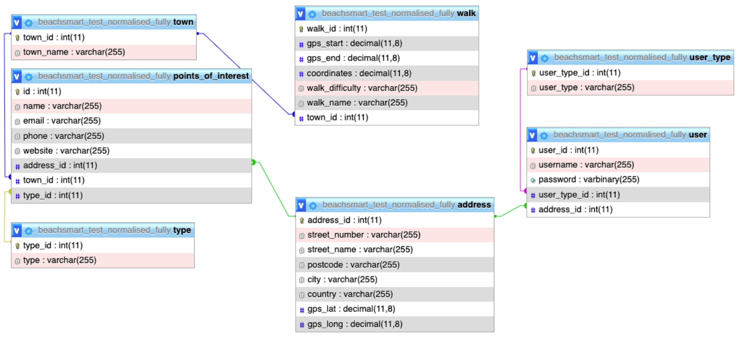


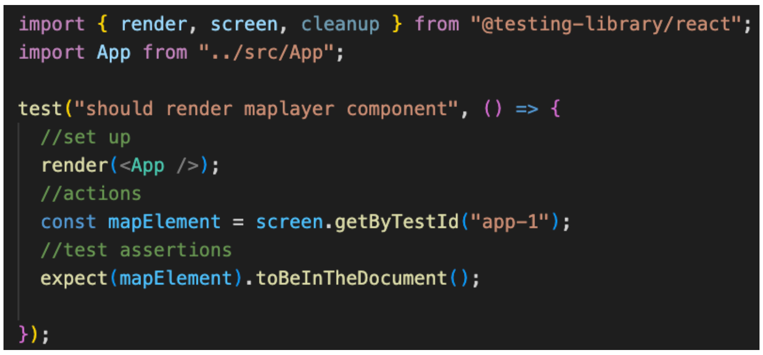
Figure 40 - Complete normalisation of the current data to include user table and type, town, user_type, address normalisation and a table for walking routes.

Ultimately, it was decided to keep the simplest option for this project due to the small size of the data at this stage of development. The SQL statements within the API functions are easily modified to include the additional joins required for a more normalised database structure, should the size and complexity of the dataset require it.

Chapter 5: Testing

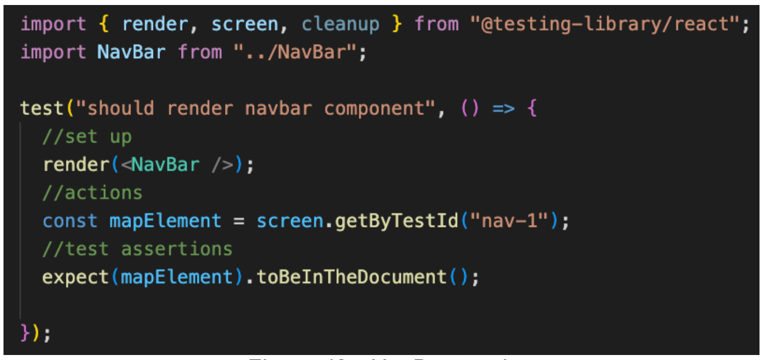
When running tests in a React app there are two broad categories: rendering component tests and running a complete app. It is recommended that this is done using Jest – a JavaScript test runner or the React-testing-library (React, 2022).

The following screenshots show the implemented component tests for this React app.



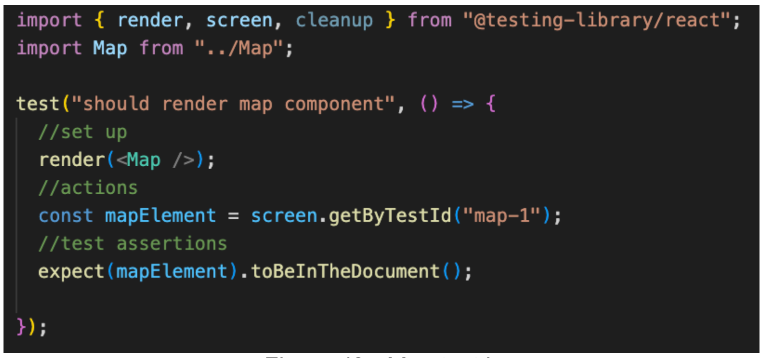
```
import { render, screen, cleanup } from "@testing-library/react";
import App from "../src/App";

test("should render maplayer component", () => {
  //set up
  render(<App />);
  //actions
  const mapElement = screen.getByTestId("app-1");
  //test assertions
  expect(mapElement).toBeInTheDocument();

});
```

Figure 41 - App.test.js



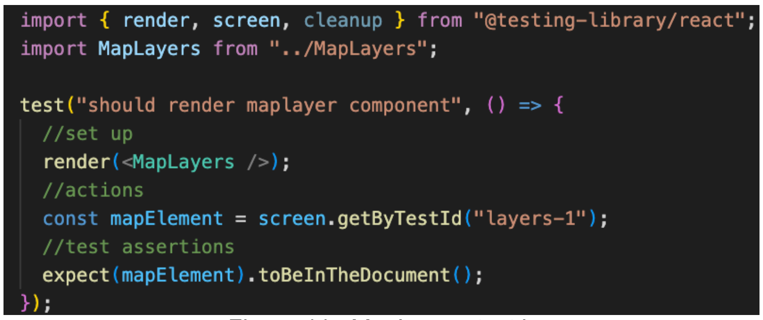
```
import { render, screen, cleanup } from "@testing-library/react";
import NavBar from "../NavBar";

test("should render navbar component", () => {
  //set up
  render(<NavBar />);
  //actions
  const mapElement = screen.getByTestId("nav-1");
  //test assertions
  expect(mapElement).toBeInTheDocument();

});
```

Figure 42 - NavBar.test.js



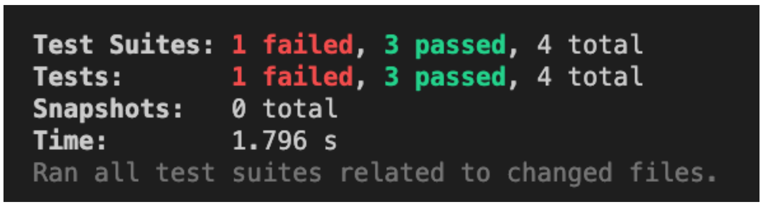
```
import { render, screen, cleanup } from "@testing-library/react";
import Map from "../Map";

test("should render map component", () => {
  //set up
  render(<Map />);
  //actions
  const mapElement = screen.getByTestId("map-1");
  //test assertions
  expect(mapElement).toBeInTheDocument();

});
```

Figure 43 - Map.test.js



```
import { render, screen, cleanup } from "@testing-library/react";
import MapLayers from "../MapLayers";

test("should render maplayer component", () => {
  //set up
  render(<MapLayers />);
  //actions
  const mapElement = screen.getByTestId("layers-1");
  //test assertions
  expect(mapElement).toBeInTheDocument();
});
```

Figure 44 - MapLayers.test.js



```
Test Suites: 1 failed, 3 passed, 4 total
Tests:       1 failed, 3 passed, 4 total
Snapshots:   0 total
Time:        1.796 s
Ran all test suites related to changed files.
```

Figure 45 - Test results

There appears to be no issues when running the app from a user point of view. Due to the simplicity of the app's functionality, there are no obvious areas where bottlenecks might arise.

Chapter 6: Evaluation and Conclusion

Requirement adherence

It seems most appropriate to begin the evaluation of this project by looking at how well it has fulfilled the initial requirements at this stage in development. Table 3 shows a summary of the requirements for this project, and which have been fulfilled at the time of submission.

Table 3 - Requirement adherence table.

	Requirement	Y/N
Must have	• Accessibility features for users with keyboard-only or visual/auditory impairments – to include light/dark themes and support for text-to-speech services.	Y
	• An interactive map with pins displaying the beaches that the council oversees.	Y
	• Users must be able to explore the map, to include zooming and panning.	Y
	• A link for each beach to the relevant CCG website page.	Y
	• Environment, historical and practical (accessibility, closure times etc.) information for a beach that is selected. To include beach status such as flag and water category. Either included in popup or available via external link.	Y
	• Users must be able to easily contact the council for any extra information or to report anything.	Y
Should have	• Local landmarks, businesses and sports facilities, highlighted with icons/pins – to include opening hours, contact details and links to external websites, where applicable.	Y
	• A map filter allowing users to decide what type of amenity or attraction type they wish to browse.	Y
	• Live weather and tidal data allowing users to safely explore the area.	Y*
	• Bus and train routes around the area with timetables (better UX) or links to external Translink website.	Y
	• Should have external links to other websites for information on the area, beaches, lifeguards etc., this should open externally so as not to confuse any less technically minded users or cause someone to lose their place on the map.	Y
	• Points of interest stored in a database and accessed via URL requests.	Y
	• There should be the ability for an admin to access all points of interest as a list and be able to add, update or delete points of interest on the map.	Y
Could have	• Users could be able to select items on the map, save to an itinerary and download as a pdf for future reference.	N

	• Users could pick a list of beaches, landmarks etc., they wish to visit and give their available time, start and end point. An algorithm would be used to calculate their most efficient route. This is perhaps a more tourist-oriented requirement but could be of great use to locals too.	N
	• Users could be able to create an account – allowing them to add map items to a watchlist, get notifications about events and save favourite routes – could sign up with email or Google account.	N
	• Users could report any rubbish, damaged property or anti-social behaviour to the council, perhaps uploading a photo from the app. This would be reported along with time, date and geolocation data, allowing the council to take quick and efficient action.	N
	• Users could report sightings of wildlife e.g., birds, whales, dolphins or seals, which occasionally show up on the North Coast. These could be pinned to a location to let others know where they may catch a glimpse. This could be used also to highlight areas where people have, for example, been stung by jellyfish. This could allow others to bathe and enjoy the beach safely and with greater peace of mind.	N
	• An API, likely built using PHP or NodeJS, allowing other developers to make use of the aggregated data used in the app.	N
	• Social media aggregators which display tweets or Facebook posts using pre-determined hashtags or the CCG social feeds, as per "plan your trip" page on the CCG website (RNLI, 2022).	N
Won't have yet	• Integration with Fitbit, Strava or similar, to save routes or walks and share progress with friends and family.	N
	• Social media integration allowing posts to be made direct from the web app.	N
	• Photos uploaded by viewers aggregated together and shown in the information section of each pin location as a slideshow with username and date below.	N
	• Augmented reality functionality, allowing users to preview the area and decide as to where they wish to visit.	N
REQUIREMENTS MET 13/24.	PERCENTAGE = 54%	
*APIs required payment for access to API key. See *External APIs* in Evaluation for more information.		

54% of requirements met is not a satisfying result. Some of the discussion below may highlight some reasons why this is the case. The requirements were set out in the hope of following the principles of good practice; necessary, verifiable and attainable (ReqExperts, 2022). As read above, for the most part these requirements fit the bill.

MoSCoW prioritisation was a new concept before beginning this project, one which has been very useful in helping form a clearer image of the importance of each requirement. It might have been advantageous to further refine the requirements to be more specific and have fewer dependencies on other requirements – an important aspect for requirements.

It must be said that the requirement analysis stage may have been more straight forward and worthwhile had there been contacts at the council who could give further insight into exactly what was needed. In an industry project, the requirement analysis often relies heavily on customer or user input and feedback. The lack of this was the reason for not using the likes of the Kano Method of prioritisation. On reflection, and with access to customer opinion, something like this would likely lead to a more robust set of requirements. The constant feedback from demoing iterations with the customer would have been beneficial in terms of developing a working prototype much quicker. This might have allowed more time to focus on details, some of which are lacking from this submission. It is important to mention that this is by no means an excuse, rather just that by working on a project like this without the ability to canvas customers and users, did lead to a less focused development cycle in terms of quickly iterating through versions.

Development Cycle

Starting development on the front-end

The plan for the development phase of this project was to begin by developing the front-end first (Appendix II). The idea was that, since this is a user-focussed project, that by having the front-end sorted and looking nice and professional, that the back end could be developed after, and everything would work nicely. The point of interest dataset was written as a JSON object and stored in the react app files. The idea was that the front end would take the data from the JSON file and use it to populate the map. Since the API would be fetching data from the database and returning encoded as JSON, anyway, connecting to the API once developed would be straightforward.

This was a poor plan in practice. The initial stages of development were enjoyable, making sure the colours work, the layout makes sense and was adequately responsive and playing with the OSM map layers and various other interesting map functions from LeafletJS. The problem arose once the design had been finalised and it was time to populate the data. It was simple enough to get all the points of interest to be displayed but when it came time to add more functionality, the API had to be developed.

The problem arose because much of the skeleton of the app had been developed using the JSON file. While this was no problem for just displaying the pins, when it came to accessing the data and running functions like looping through the mapped array or trying to filter which points of interest were to be rendered, issues kept popping up. This led to the remainder of the development period switching between front end and back-end API. It was a bit of a mess to be honest. Dr Cutting suggested that the API be developed using postman or the browser, and to ignore the front-end until one can be sure that the right data is being requested and sent. This made a huge difference to the speed and quality of development, though unfortunately many weeks were already lost to hopping between the two frantically.
In future projects such as this, the API shall be developed first. The API should be complete with solid architecture, functions that all work as intended and well written code before even beginning to think about developing the front end. By using postman to test the functionality of the API, to ensure that it reliably returns exactly the data requested, when it comes to frontend development any bugs or issues are guaranteed to be in the front end. The way this project was done, debugging took longer that it should as one was never sure if the error was on the front-end, back-end or both. Or perhaps neither of them. Which did derail the project on a few occasions.

Upon reflection the development cycle to be followed would be something along the lines of back-end API > API testing > back-end Admin portal > Admin portal testing > front-end UI development > front-end UI testing. Essentially back to front with how thus project was developed. On the evidence of how this project went, this could improve the final product significantly.

Technology choices

It was decided that this project should been seen as an opportunity to learn some new technologies and get some experience using industry standard techniques. As there were only a few days of JavaScript at the end of the web development module, it was important to use JavaScript extensively throughout the project given that it is such a popular language, with so many use cases. The natural progression for this was to use ReactJS to create the U.I., react had been lauded as an elegant way to create high level and professional U.I.s in JavaScript. This is certainly the case for individuals who know JavaScript. Unfortunately, it meant learning JavaScript from scratch using FreeCodeCamp, YouTube and documentation, while simultaneously doing the same with react, using Scrimba instead of FreeCodeCamp (Appendix. I). Once JavaScript and React had been partially learnt, it became apparent that TypeScript might be better to use, some of the better React-Leaflet tutorials used TypeScript. This led to some time exploring TypeScript and trying to learn it too. This turned out to be a mistake. Since TypeScript is a superset of JavaScript and both work in React, this meant trying to develop a project using three technologies that were related but not really knowing how to use any of them with a great deal of confidence. Many days were wasted trying to piece the three together, to the point that the decision was made to scrap TypeScript for now. On reflection, this decision should have been made after a single day of attempting to use typescript.

All this time spent on these technologies, meant that by the time API development started, the prospect of learning how to use Node.js from scratch too became unrealistic, hence it was decided to write the API and backend in PHP – a less fashionable, but reliable alternative. On one hand this was a disappointment as it was the plan for this project to be full-stack JavaScript app to take advantage of the benefits such as improved performance from a fully asynchronous project (BairesDev, 2022). On the other hand, it did mean that API development was somewhat more straightforward in the immediate term. It is obvious at this stage that had this project been done in HTML, CSS and Vanilla JavaScript with a backend in PHP, developed API first, that the final app would be richer in functionality and have met more requirements. Had this been the case, however, there would have been nothing new learnt (besides the vanilla JS). The idea was to push the boundaries to see how much could be learnt in the short period of time that the project ran, so by that metric, it has been a success. Moving forward, time will be spent on properly learning JavaScript before moving onto its supersets, but this will be a much more straightforward process thanks to the experience of the last few months.

Time spent researching technologies and following tutorials

In the initial GANNT chart for this project (Appendix II), 14 days were allocated to research of technologies. In the end, this ended up being more like three and a half weeks. This is maybe indicative of a more research heavy background in terms of undergraduate study, leading to less aversion to the 'rabbit-hole'. In all honesty it was more a case of being stuck in the loop of reading up on a technology, following tutorials on said technology and hoping to learn by 'doing'. In principle there is nothing wrong with this but given the time constraint, a much more focussed research period would be better. The issue I believe for this project is not quite knowing what one doesn't know yet, so having the conviction to decide what to research and when did prove difficult. This would perhaps have been less of an issue with more support from the council, but also speaks to the nature of learning new technologies on a tight time scale. The positive to take from this is that moving forward, learning new technologies, and properly learning the technologies used here is no longer a daunting prospect but rather an exciting one, as it is clearer now what should be learned and when.

U.I. Design

Front-end

The final front end U.I. follows the original wireframe well (Fig. 46). There are small differences such as the large search and filter sections and the footer. Having said that as far as sticking to the original 'customer' approved design, this is a successful U.I. The chosen colours work nicely and do not strain the eyes after prolonged use. The use of monospace Roboto Mono as a font has worked well. The text is easily legible with nods to both serif and sans-serif fonts in its shape. This makes it a nice option to appeal to most users across the board. As there are no real paragraphs in the app, the character spacing lends itself perfectly to short headings or small pieces of information as they can be read quickly and easily, particularly on smaller screens. On larger screens, it can look a bit stretched out if there are only small amounts of text but the high level of readability, make up for this.

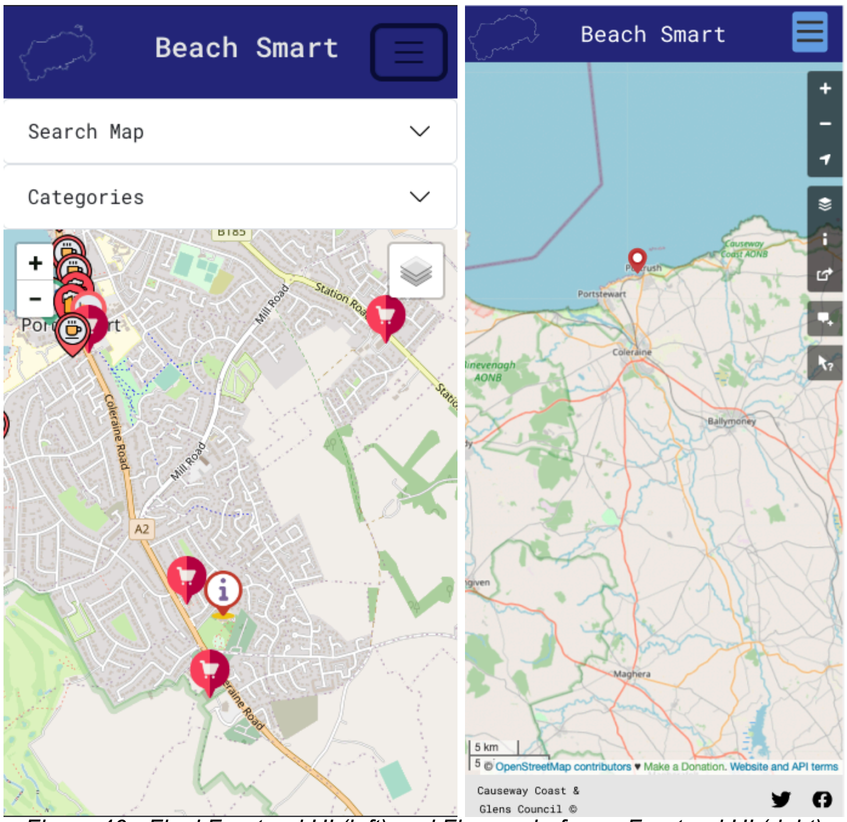

Beach Smart
Search Map
Categories
Beach Smart
Portstewart
Coleraine
Ballymoney
Maghera
5 km
© OpenStreetMap contributors ♥ Make a Donation. Website and API terms
Causeway Coast & Glens Council ©

Figure 46 - Final Frontend UI (left) and Figma wireframe Frontend UI (right)

The pin icons were taken from flaticon.com (attribution below map) and work very well. Initially they had been chosen in a black and grey colour scheme but once the dark mode and satellite layers were added, these were unreadable. It was then decided to have them be in different shades of red, to give high contrast regardless of background so they could be easily seen at any given zoom level. The walk pins are different, in that they are different colours depending on the difficulty of the walk. If this project were to be done again, these pin icons would be designed specifically for beach smart. As they are simply PNG images with their size defined in code, using something like Figma or GIMP would allow for beautiful pins to be made that were specific to the area. This would allow for all manner of types of point of interest to be included and could improve the general UX of the app and give it more of a custom-made flavour.

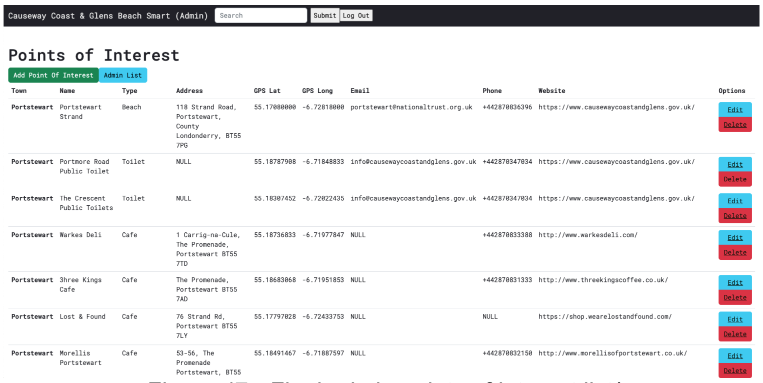
The contrast ratio of the burger menu is unacceptable, the lines should have been in the platinum colour that was set up as one of the colour variables, or little baby blue for the background, as it was in the wireframe. It appears that the bootstrap CSS styling for the burger menu has overridden the local stylesheet. The way the filter and search options are

The 'Search Map' and 'Categories' dropdown accordion components are ugly. They serve their purpose well, but they are not pleasing to the eye, and they take a significant percentage of the map's real estate on screen. This may pose a problem for older (or generally poorer in sight) individuals or those using smaller devices, with the focus supposed to be the map itself. This would have been better as a smaller button somewhere to open a pane or a Bootstrap modal component which could pop up on screen, allow the user to select their option and then apply, with the map updating dynamically in the background. This would have been a much more elegant solution. As for the search bar, this maybe should have been held in the navbar as a search icon, this would be cleaner and improve UX. Although, as part of the requirements of this project were surround accessibility, particularly for those with issues surround sight and reading. This meant that having the search and filter options large and easily visible took precedence over having them look neat and sleek. Given some extra time they could be nicer to look at and still fulfil the brief of accessibility, although as it stands now, they are acceptable, if not amazing.

Back-end

UI is acceptable, it is clean and simple, uses hierarchy and has common elements in terms of buttons and list style. While this ticks the boxes as far as design principles are concerned. The U.I. could, however, be more polished in terms of functionality.

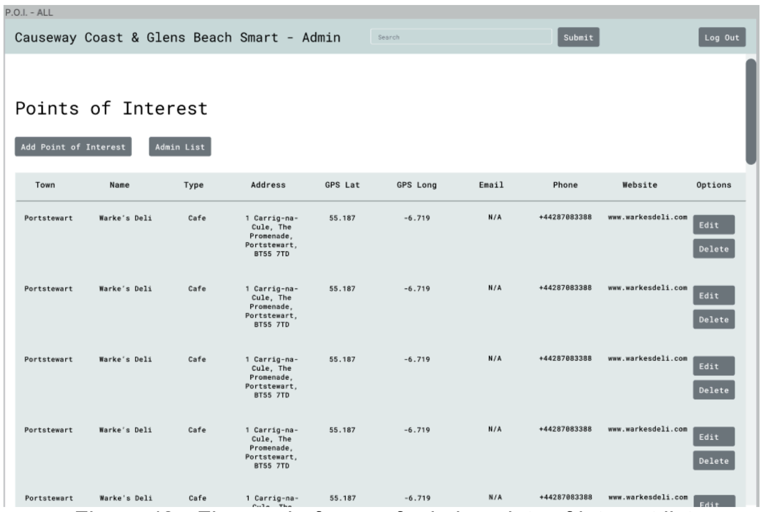
In terms of design, the admin portal follows the initial U.I. plan with fair accuracy (Fig. 47, 48). Therefore, the question remains, if perhaps the initial wireframe maybe could have been designed more thoughtfully or perhaps with more iterations to finalise a design before committing to development.



Points of Interest

Town	Name	Type	Address	GPS Lat	GPS Long	Email	Phone	Website	Options
Portstewart	Portstewart Strand	Beach	118 Strand Road, Portstewart, County Londonderry, BT55 7PG	55.17080000	-6.72818000	portstewart@nationaltrust.org.uk	+442870836396	https://www.causewaycoastandglens.gov.uk/	Edit Delete
Portstewart	Portmore Road Public Toilet	Toilet	NULL	55.18787908	-6.71848833	info@causewaycoastandglens.gov.uk	+442870347034	https://www.causewaycoastandglens.gov.uk/	Edit Delete
Portstewart	The Crescent Public Toilets	Toilet	NULL	55.18307452	-6.72022435	info@causewaycoastandglens.gov.uk	+442870347034	https://www.causewaycoastandglens.gov.uk/	Edit Delete
Portstewart	Warkes Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart BT55 7TD	55.18736833	-6.71977847	NULL	+442870833388	http://www.warkesdeli.com/	Edit Delete
Portstewart	3hree Kings Cafe	Cafe	The Promenade, Portstewart BT55 7AD	55.18683068	-6.71951853	NULL	+442870831333	http://www.threekingscoffee.co.uk/	Edit Delete
Portstewart	Lost & Found	Cafe	76 Strand Rd, Portstewart BT55 7LY	55.17797028	-6.72433753	NULL	NULL	https://shop.wearelostandfound.com/	Edit Delete
Portstewart	Morellis Portstewart	Cafe	53-56, The Promenade Portstewart, BT55	55.18491467	-6.71887597	NULL	+442870832150	http://www.morellisofportstewart.co.uk/	Edit Delete

Figure 47 - Final admin points of interest list/



Points of Interest

Town	Name	Type	Address	GPS Lat	GPS Long	Email	Phone	Website	Options
Portstewart	Warke's Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart, BT55 7TD	55.187	-6.719	N/A	+44287083388	www.warkesdeli.com	Edit Delete
Portstewart	Warke's Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart, BT55 7TD	55.187	-6.719	N/A	+44287083388	www.warkesdeli.com	Edit Delete
Portstewart	Warke's Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart, BT55 7TD	55.187	-6.719	N/A	+44287083388	www.warkesdeli.com	Edit Delete
Portstewart	Warke's Deli	Cafe	1 Carrig-na-Cule, The Promenade, Portstewart, BT55 7TD	55.187	-6.719	N/A	+44287083388	www.warkesdeli.com	Edit Delete
Portstewart	Warke's Deli	Cafe	1 Carrig-na-	55.187	-6.719	N/A	+44287083388	www.warkesdeli.com	Edit

Figure 48 - Figma wireframe of admin points of interest list

In terms of design alone, the admin portal is a success simply because it follows the original design well. Whether the original design is satisfactory is a different question. The intention was to get some experience with the design process as this is important in today's world. The experience of further learning how to use Figma, how wireframes work and in general to get a taste of the process of designing the user interfaces has been enjoyable and valuable. The same can be said for the experience of using GIMP to create logos and edit images. It is auxiliary technologies like these, that don't help with the coded development, but are equally as important when considering a project like this as a future complete, industrial level, product.

Map Features

Map

The map fulfils the requirements in as far as that users can interact with it by panning, zooming and moving. It also displays the relevant markers for beaches and points of interest. Generally, the map can be considered a success because of this. There is not much else that would be done differently as far as the base implementation of the map is concerned. It uses the default OSM map layer, although other standard layers are available from the leaflet-extras github (LeafletProvider, 2022). This is where the additional map layers such as dark mode were sourced, but they also include other base layers in different languages if necessary. Adding different language options would improve the appeal of the app to foreign visitors but it may be more elegant to set it up so that the entire page is translated, including the map details, into various languages, this is something to consider.

There are not too many qualms at this stage with the map implementation itself, other than how much of the screen it takes up (or doesn't, rather), as mentioned above.

Map filters

The map filter, or 'Categories' as it is named in the final submission, was an important UX requirement. Figure 50 shows how this was set up initially in the U.I.

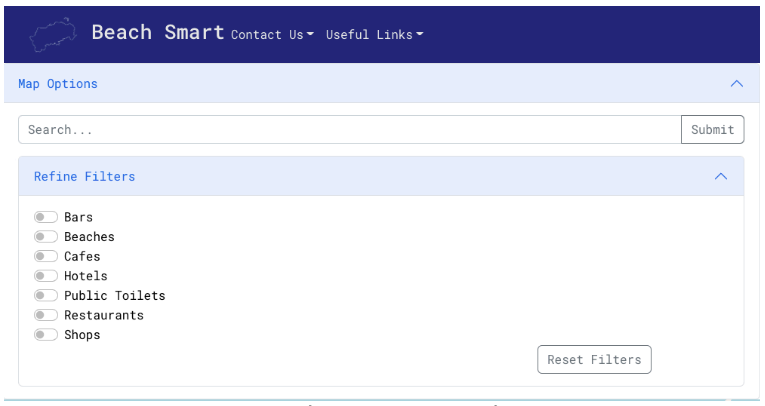


Figure 50 - Map filter checkbox in front end React app.

The plan was to save the state of each checkbox in an array of types. These were to be nested within the 'Map Options' accordion, to free up space and allow for more of the map to be on screen. This would have improved the usability of the map as there would be more screen real estate to play with. If checked, the array would contain the type assigned to each checkbox. This array would then be appended onto the fetch request to the API that dealt with populating the map. This could then be sent each time a checkbox was checked or unchecked, dynamically changing the pins that are displayed on the map in a similar way to the search function. Unfortunately, getting this to work proved challenging. It may be that the issue was linking the search function to the filter function. This meant that once the default map had been loaded, the only way to change it was to change the value of searchQuery, which did not want to accept an array. This is almost certainly down to not having found the solution in good time rather than there not being one. A better idea would have been to either have the searchQuery set up as an array to begin with, or better yet have the filter function first clear the map and then repopulate it by taking the array from the filter box and parsing it on the back end. This could then have been split into variables and conditionally added to a prepared SQL statement depending on the state of each checkbox that was passed through.

In the end the filter was implemented as a list of buttons with values corresponding to point of interest types (Fig. 51). These buttons, when clicked, change the value of searchQuery – the variable which is changed by the user search to update the map icons. Although this works ok, it is far from perfect. Firstly, it is no longer nested together with the search options. This was done with the intention of separating the two functions in the mind of user to make using the app for the first time more straightforward. Though, it is unsure whether this was even worth considering, perhaps more credit should be given to users. The way it is set up currently causes issues when one of the type values is included in a different point of interest – for example, clicking 'Bars' does show all the bars in the area, but also shows the 'Bar Mouth & Estuary Walk' popup. Changing the way the filter system is structured would avoid this sort of issue.

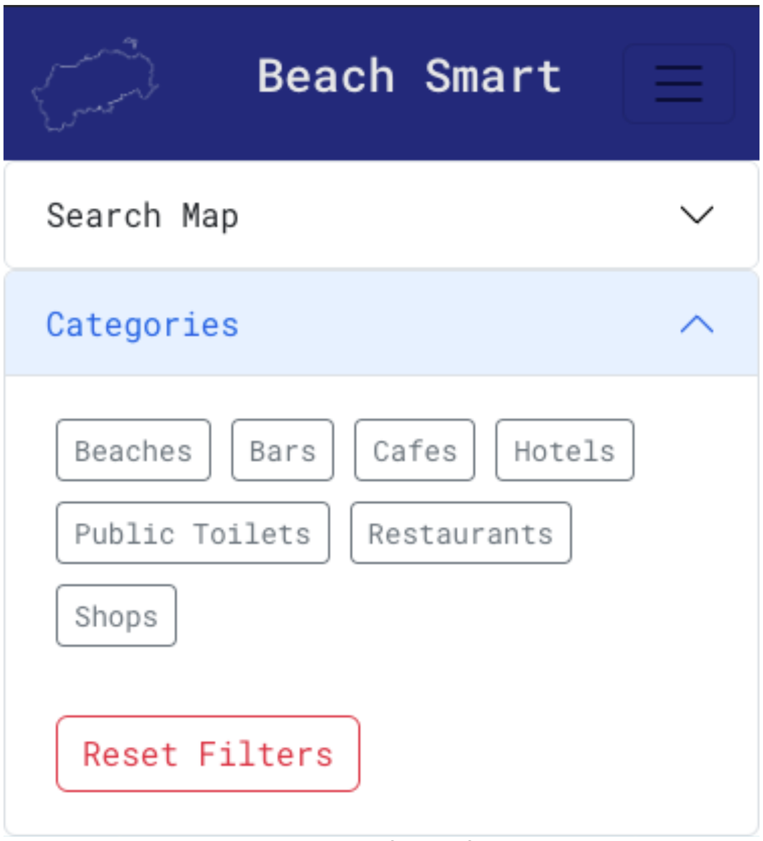


Figure 51 - Final checkbox filter in frontend React app

The ‘Reset Filters’ button simply refreshes the page, which resends the first fetch request that is fired upon starting the page. This returns all points of interest and returns this to its default state. A more eloquent way to do this would to have had a button that changed the searchQuery, either by having its value correspond to something that would return all points of interest, or more likely by having it send a get request to update the map icons. This would negate the need to refresh the page, which would make for a better UX.

Points of Interest Pop-up

Each point of interest on the map has a distinct icon based on their type. Once clicked on, a marker popup shows on screen with the details of each point of interest. These are taken by mapping through the pointsOfInterest array that is created from the fetch call to the API. These values are stored as variables and echoed out within the marker render as the list is looped through. This works well and is good for most points of interest. Users can easily contact representatives of each point of interest by phone, email or by visiting their website. Not all points of interest have phone, email and website information available though. Where this is the case, the marker should not have the respective button showing but rather should tell the user that there is no phone, email or website for a given point of interest. Likewise, for the likes of restaurants, bars and hotels, the plan was to have an image file saved to display on the marker. Early versions of the JSON object had this but there was inconsistency across the points of interest having usable imagery available. The plan was to go out and collect the missing pictures using a borrowed DSLR camera. Unfortunately, this became difficult due to extenuating circumstances. Therefore, it was left out at this stage.

There is room for improvement, particularly in the layout and complexity of the information in the markers for different types of points of interest. There should be a separate map function points of interest, beaches and walking routes. This would allow to have different popup marker contents for these different categories. For example, beaches could have additional environmental and safety information and walks would include length, difficulty, average time taken, and perhaps the ability to check the walk off as completed. The popups for the points of interest are acceptable, besides the lack of opening times. Opening times is an important factor but it can be found by following the website or contact us links, which doesn’t make for a great user experience by comparison. The basic implementation is good, but there is scope for significant improvement.

Walking Routes

Walking routes were meticulously plotted for well-known walks and hikes around the Portstewart area (Fig. 52). This was done using geojson.io, with the coordinates and line information stored in a GeoJSON format, this also calculates and stores the length of each walk, in various units (Fig. 53) which would have been perfect to use in the itinerary feature of the app moving forward.

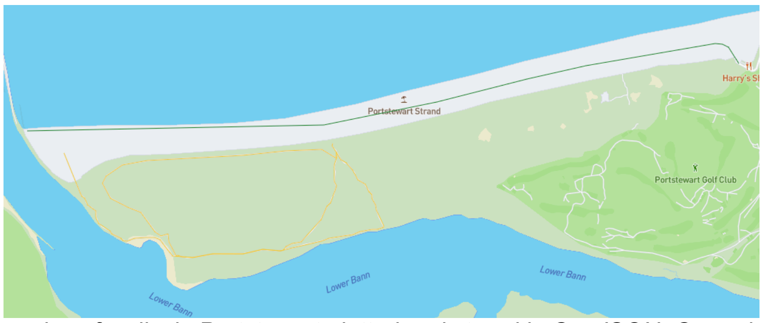
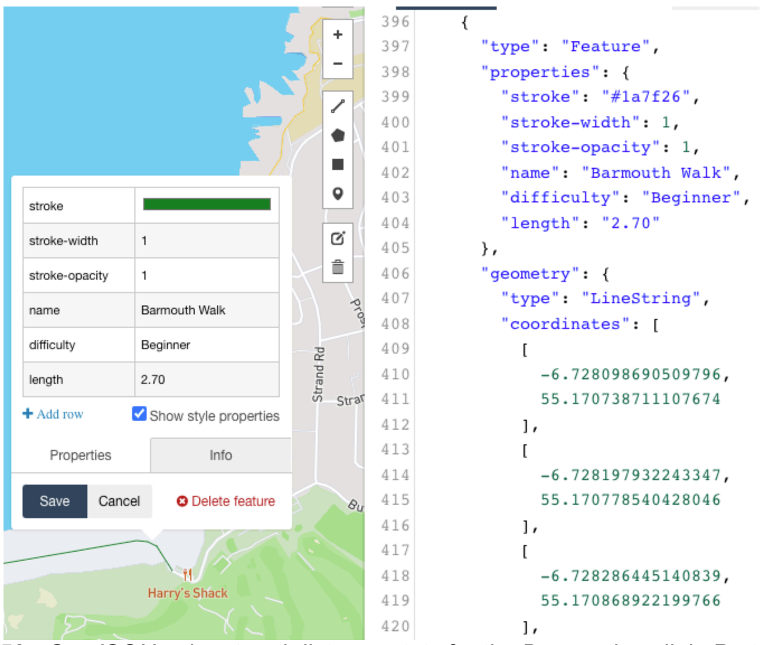


Figure 52 - Two examples of walks in Portstewart plotted and stored in GeoJSON. Green highlights a beginner walk, yellow indicates an intermediate walk.



stroke	🟩
stroke-width	1
stroke-opacity	1
name	Barmouth Walk
difficulty	Beginner
length	2.70

```
  {
    "type": "Feature",
    "properties": {
      "stroke": "#1a7f26",
      "stroke-width": 1,
      "stroke-opacity": 1,
      "name": "Barmouth Walk",
      "difficulty": "Beginner",
      "length": "2.70"
    },
    "geometry": {
      "type": "LineString",
      "coordinates": [
        [
          -6.728098690509796,
          55.170738711107674
        ],
        [
          -6.728197932243347,
          55.170778540428046
        ],
        [
          -6.728286445140839,
          55.170868922199766
        ],
```

Figure 53 - GeoJSON snippet and distance stats for the Barmouth walk in Portstewart.

There was much excitement upon discovering GeoJSON, it solves the problem perfectly of being in total control of plotting the walks and hikes in the area. This is great for somewhere like the North Coast as many of the best walking routes are not very well known, so with some local knowledge one could build up an extensive record of all the walks and hikes in the area, to be displayed on the map.

Unfortunately, this was not possible as, despite being a version of JSON there were issues with getting the data to display properly on the map. It is believed that this may be due to the database structure as there was no way in which to elegantly include the walk data. Therefore, it was included as a geojson file in the source code. As per figure 40, a database structure that includes a separate table for the walk data, coupled with separate map functions for the points of interest and walks, should allow this to be achieved without too much hassle. The geojson coordinate data would be displayed within the same function as the walk icons, rather than alongside the other points of interest.

Another option was to create vector or raster tile images using Mapbox to have the walking routes as extra layers – this is how it was going to be implemented as a workaround to the fact that it was not possible to get the geojson data to show on the map. It transpired that creating these map tiles was not as simple as laying out the walking routes. There wasn't enough time to implement this feature in the end. In all honesty, although it may have ticked a requirement, it also would not have been satisfactory. Ultimately, the walks must be integrated into the database and displayed from there, this would be the ideal option.

Itinerary

The decision was made to not have users at this stage of development, and instead to focus on the other functionality first. This meant that fulfilling the requirement to have an itinerary option became difficult. The itinerary needed users as it would have to use state to manage and save the points of interest or walks that the user wants to add to their itinerary. This could then have been saved to their profile and accessed later or downloaded as PDF or JPEG to the user's device for future reference or to print out, if they were so inclined. This would be an excellent additional feature to include in the future. The itinerary would have been disabled until a point of interest was added. It would then appear, perhaps in an accordion from the bottom of the map. This would store each point of interest until the user saved it or logged off.

From here the user could select the order in which they wish to explore their chosen points of interest and the app could algorithmically calculate the most efficient route to follow. The idea was to explore the slime mould algorithm. Slime mould refers to a group of eukaryotic, spore producing cells (Schapp, 2021), which when growing out from a nucleus in culture, forms connections across space that are so efficient they famously were used to redesign the Tokyo railway network (Christensen, 2010; Yong, 2021). This algorithm, or any other appropriate one, may have been used to help calculate these routes. This would also give an opportunity to gain experience using Python. This would have been a delightful feature, and incredibly fun to explore, but unfortunately just out of scope for this project.

Map Layers

Map Layers were included to enrich the functionality of the app while also not increasing the clutter on screen. They are housed in the upper right corner in the layers icon and appear when pressed. Figure 54 shows the options currently available.

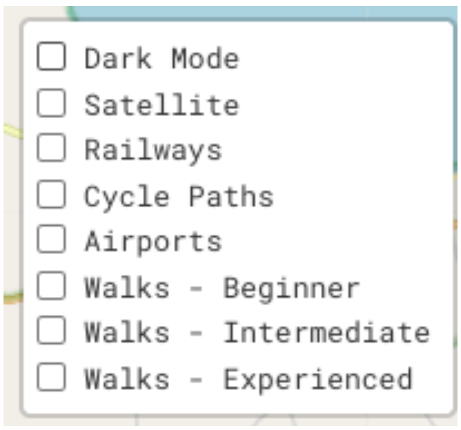


Figure 54 - Map Layer options.

The dark mode was important for the sake of accessibility as it drastically improves the readability of the map in low light conditions. A satellite layer is available, primarily to allow users to first explore the coast how it looks from above before they plan how they might explore along the ground. These layers are fit for purpose and look professional in as far as they are options which are available in industry standard mapping applications. Railways, Cycle Paths and Airports are included with tourists in mind rather than locals, it offers a quick and easy way to see, briefly what transport options are nearby. Unfortunately, it didn't get so far as to include the same data for bus routes as there was no map layer available for this, an option was to create our own. As mentioned above though, to create a custom map tile layer with this information would have taken longer than was left in the project. That's not to say that access to the Translink bus route data would have been allowed, without the explicit backing of the council. The walking route layers have been discussed above, in an ideal world they would not be included in the map layer options. Other than the walks, though, the map layers serve their purpose and improve the functionality of the app.

A welcome addition would have been the weather map layers which can show tidal flows, air pressure and precipitation levels, this would have been excellent for beach goers in general but for the large water-sports community that exists along the coast of Northern Ireland. There are map layer tiles available for these from the leaflet-extras github repository, but they require access to the relevant APIs via API key, so they have had to be omitted for now. Instead, the information is available via link. This would be a welcome addition to future iterations of the project.

React v React Native

When researching React for this project, React Native came up as an option. React Native is a React platform used to develop Android and iOS apps. This may be a better option for future development than standard React as users may be more likely to trust a 'normal' phone app rather than a web app as is the case in this project, with people viewing 4.2x more products within an app than on a mobile webapp, this is likely due to improved UX, performance speed and extra features (JMango360, 2022).

React Native apps are quick to develop, easily maintained and have good UX on apps of this scale – it also includes pre-made components, good debugging options and a large community from which to get assistance when needed. Having said that there are some issues, potentially with performance (Marcak, 2022).

It is unclear at this stage whether React Native would have been a better option rather than react, it may well just be a different option but is one that is worth exploring as a fully-fledged app, whether native or otherwise may allow more users to access the features rather than having to use the browser in their device.

Administrator Features

Points of Interest and Admin management

The administrative list of points of interest and admin lists are acceptable. The pros and cons of how they were implemented are essentially the same.

They both display all the required information, concisely. The options for editing and deleting entries work well and are generally acceptable in terms of UX. The admin list does not use space effectively. As it only displays id and username, there is no need to have the list sprawl across the page, perhaps a panel or having both lists on the same page split into tabs using Bootstraps pill components would have made for a more coherent look. Passwords were omitted for security reasons but there really should be an option for the admin to change passwords – these would simply have to be displayed as '******', for example, in a disabled input group which could be enabled by entering the admin password. Id should maybe not be included either for security reasons, this was included primarily for debugging but was left in as it looked better in space – possibly a mistake.

Moving forward, there would likely be more information in the admin table, such as address, date of birth or profile picture. Standard sort of information for such an application, the display of which on the admin portal might make it more intriguing. There would also ideally be user profiles stored in the database, with users being able to add all their information in the app, with the administrator being able to manage their details from the admin portal. Having the extra information would improve the U.I. and U.X. as the proportions would make more sense and there would be the opportunity for increased functionality – for example, user favourites, fitness app integration or interactive elements on the map that allow users to report issues or sightings of exotic wildlife. As the admin portal features rely on the front-end features for functionality, as the app itself improves in functionality, so to would the admin portal.

The inclusion of pagination for the list would be good moving forward, for this data set it is no problem as there are only a small number of database entries. Once this dataset has 50, 100 or even more entries, this list is no longer so good in terms of UX. Pagination should be straightforward to implement using JavaScript and Bootstrap's pagination components, something that could be added. The 'New Admin' and 'Add Point of Interest' functionality works well, with no issues.

The main issue with the admin features is the lack of authentication in terms of logging in (see below). Overall, the admin portal fulfils its basic tasks – to manage the data in the database without having access to it. Therefore, it is not a failure as such but there are many small tweaks that could be made to improve it significantly.

Log in

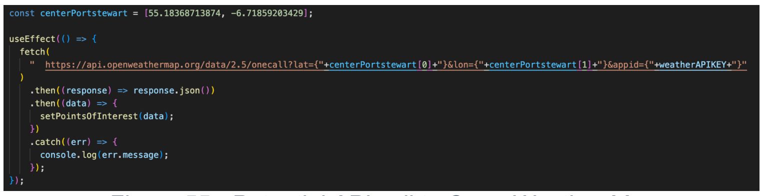
Unfortunately, despite working up until a week before submission, and being like the login system used in a previous project, the log in system ceased to work. It is unclear at this stage what the issue is, it may well be something simple that has been missed due to code blindness or in relation to the structure of the database tables. This is particularly disappointing as it is an important aspect. Were it to be done again, it would be on the same page as the admin home (points-of-interest-all.php) and would be dependent on session state. Upon opening the page, there would be no session variable set and thus the client would render the log in box. A function would confirm that the user exists and then another compares the passwords. Once the log in credentials have been confirmed, the client would then render the admin list and the remaining files within the admin would behave as usual.

As it is currently set up, the admin and API are in the same root directory, with little authentication. This is unacceptable really and will be a bug bearer for the coming months.

Miscellaneous

External APIs

One of the requirements of the project was to include tidal and wind information. The plan was to use the Open Weather Map API to request real-time data for weather and tides. This would have been done by getting a unique API key and setting the latitude and longitude to those of each beach and include a feed or dropdown within the popup marker. Figure 55 shows an example for Portstewart. The user could then access the current weather forecast for a given area.



```
const centerPortstewart = [55.18368713874, -6.71859203429];

useEffect(() => {
  fetch(
    "  https://api.openweathermap.org/data/2.5/onecall?lat={"+centerPortstewart[0]+"}&lon={"+centerPortstewart[1]+"}&appid={"+weatherAPIKEY+"}"
  )
    .then((response) => response.json())
    .then((data) => {
      setPointsOfInterest(data);
    })
    .catch((err) => {
      console.log(err.message);
    });
});
```

Figure 55 - Potential API call to Open Weather Map

Unfortunately, these appear to not be free services (OpenWeatherMap, 2022), and while not as expensive as something like Google Maps, it was decided that at this stage the project should not include paid services were possible as this is something that would have to be decided upon by the customer, to fit within their budget. The work-around for this was to include the relevant websites in the 'useful links' dropdown menu in the navbar. While far from ideal, this seemed the most appropriate without access to the real-time data. An option, given a longer time would maybe be to develop a proprietary weather/tidal API as part of the BeachSmart product.

Testing

Testing is the most certainly the area of this project which leaves the most to be desired. The plan was to write simple unit tests for each component and complete an app-wide end-to-end test. It is now clear that this should have been done alongside developing the components, testing was left to late by which time it was a confusing time to try and get them working. This resulted in the unit tests being desperately simple and not being robust enough to test the inner workings of the app, even though all appears to work as planned when running the app. End-to-end testing should also have been carried out using Jest but regrettably time ran out in the end.

The inclusion of react-leaflet appears to have brought up some issues regarding "Error: Uncaught [Error: No context provided: useLeafletContext () can only be used in a descendant of <MapContainer>]". Despite some research online, a solution was not found but it appears to be in relation to dependencies installed when installing react-leaflet. In the future, a test-driven development cycle could avoid such issues. In test driven development the development cycle starts by writing tests that fail first and then writing the code so that they pass. This ensures that all code written matches the test suites and can only pass if they have fulfilled the requirements. More learning and research are required on testing moving forward, current testing knowledge and ability is well below par.

There are no back-end test suites at present. This is an oversight, stemming from the front-end first approach which was taken throughout this project. Therefore, back-end testing was somehow omitted from the plan. This compounds the feeling of dissatisfaction surrounding the testing section of this project – it simply must be better in future projects. A welcome learning opportunity all the same.

Adherence to project brief

“Using interactive web-based mapping, develop a range of features to support the development of a BeachSmart campaign informing residents & visitors of environmental issues they should be aware of before using the renowned beaches & coast of the Causeway Coast and Glens.”

The final question to ask is how closely the final product follows the project brief. It must be said, difficult as it is, that it is not so clear that it does.

As discussed above, too much time was spent thinking about all the additional features, starting and restarting repeatedly. When the app should have been developed with the environmental factors around the beaches in mind. There should be more functionality that relates to this. More information in the popup for the beach, pins for the sort of wildlife that is found in certain areas, specific points around the coast where it might be dangerous, tips on how to be safe at the beach (perhaps in a learning portal of some description). This should maybe have been done for all the beaches in the area before then adding other features like the points of interest, searches or the admin features. Not enough time was spent during the development cycle, stopping and checking how well the requirements and project brief are being met at each stage, as the pressure increased, this became worse. As a result, it is felt that the project has veered somewhat from its original purpose, although if these features were to be added now, the app would suddenly be quite a good match to the brief. Although this is disappointing, comfort is taken from the good aspects of the app, while lessons have been learned from the bad.

Conclusion

The submission of this project and dissertation is somewhat bitter-sweet.

On one hand, it can be considered a failure for several reasons. Only 54% of requirements having been met is disappointing. Even the features that do fulfil requirements, many on reflection, should have been done differently. Failing to implement the GeoJSON data as planned or running out of time to implement the itinerary features due to a lapse in time management may be seen as in and of themselves enough to not label BeachSmart a success.

On the other hand, there are reasons why this project may be considered a success. Despite 54% of requirements being met one might argue that what has been submitted could be considered a sort of minimum viable product. Which, after just over three months of development, a large portion of which was spent learning numerous new technologies and how to use various new applications while attempting to develop the project concurrently, may be seen as somewhat of a personal triumph. Despite how disappointed one might be with the outcome. This introduction to such a wide variety of concepts and technologies will be invaluable in the future.

At this stage the project simply is what it is, if not what it could be. It is essentially a proof of concept for BeachSmart Portstewart, which is not necessarily anything to be ashamed of. This project has been a valuable experience, one that will be most meaningful moving forward, regardless of the ultimate outcome. This is a project that has great potential, even as an industry standard product (given a blank slate and a development team) and is a concept which very well might prove to be a hit in the future. Although the final product is not fully fit for purpose at present, much has been learned along the way.

A chance to start over again tomorrow would certainly yield a significantly better and more polished product, borne from the mistakes made here. Overall, it has been a challenging project mentally, and at times physically. A challenge which having now been completed, one can have no regrets in attempting and (in some respects) overcoming.

Bibliography

Agile Business, 2022. *Chapter 10 - MOSCOW Prioritisation*. [online] agilebusiness.org. Available at: <https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation> [Accessed 21 August 2022].

Agrawal, S., 2021. *The World Of CSS. Should you write your own CSS or use any Library?* [online] DEV Community 👩‍💻👨‍💻. Available at: <https://dev.to/theshubhagrwl/the-world-of-css-should-you-write-your-own-css-or-use-any-library-4i57> [Accessed 22 August 2022].

AltexSoft, 2020. The good and the bad of angular development. *AltexSoft*. Available at: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/ [Accessed September 21, 2022].

Aycock, J., 2003. A brief history of just-in-time. *ACM Computing Surveys*, 35(2).

BairesDev, 2022. Pros and cons of full stack JavaScript development. *BairesDev*. Available at: https://www.bairesdev.com/javascript/pros-cons-full-stack/#:~:text=Thanks%20to%20Node.,run%20multiple%20independent%20processes%20simultaneously. [Accessed September 23, 2022].

Bose, S., 2022. *Top 5 CSS Frameworks for Developers and Designers | BrowserStack*. [online] BrowserStack. Available at: <https://www.browserstack.com/guide/top-css-frameworks> [Accessed 22 August 2022].

Bullen, P., 2022. *How to choose a sample size (for the statistically challenged) - tools4dev*. [online] tools4dev. Available at: <https://tools4dev.org/resources/how-to-choose-a-sample-size/#:~:text=The%20minimum%20sample%20size%20is,to%20survey%20all%20of%20them.> [Accessed 20 August 2022].

Bulma, 2022. *Overview*. [online] Bulma.io. Available at: <https://bulma.io/documentation/overview/> [Accessed 21 August 2022].

Causewaycoastandglens.gov.uk. 2022. *Beaches - Causeway Coast & Glens Borough Council*. [online] Available at: <https://www.causewaycoastandglens.gov.uk/see-do/beaches> [Accessed 30 August 2022].

Christensen, B., 2010. *Slime mold network engineering: Science fiction in the news*. Available at: http://www.technovelgy.com/ct/Science-Fiction-News.asp?NewsNum=2756 [Accessed July 16, 2022].

CommunityAd, 2020. *RNLI and HM Coastguard launch beach safety campaign*. [online] CommunityAd.co.uk. Available at: <https://www.communityad.co.uk/rnli-hm-coastguard-launch-beach-safety-campaign-urging-parents-protect-families-save-lives-coast-summer/> [Accessed 09 August 2022].

CoreUI, How to create a single page application using react.js · coreui. · *CoreUI*. Available at: https://coreui.io/blog/how-to-create-a-single-page-application-using-reactjs/ [Accessed September 22, 2022].

Coyier, C., 2022. *What Are the Benefits of Using a CSS Framework? | CSS-Tricks*. [online] CSS-Tricks. Available at: <https://css-tricks.com/what-are-the-benefits-of-using-a-css-framework/> [Accessed 22 August 2022].

Dean, B., 2021. *85+ Web Browser Market Share & Usage Statistics (2022)*. [online] Backlinko. Available at: <https://backlinko.com/browser-market-share> [Accessed 20 September 2022].

Developer.mozilla.org. 2022. *Handling common JavaScript problems - Learn web development | MDN*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/JavaScript> [Accessed 21 September 2022].

DfE, 2020. *Tourism | Department for the Economy*. [online] Economy. Available at: <https://www.economy-ni.gov.uk/topics/tourism> [Accessed 20 August 2022].

Donovan, J.A.-Y. and R. et al., 2022. Best practices for REST API design. *Stack Overflow Blog*. Available at: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/ [Accessed September 22, 2022].

Duraj, M., 2019. *How JavaScript Implementation Adds Dynamic Interactivity*. [online] pluralsight.com. Available at: <https://www.pluralsight.com/guides/how-javascript-implementation-adds-dynamic-interactivity> [Accessed 1 September 2022].

Dziuba, A., 2022. Choosing a map API for your next app: Mapbox, google maps, or openstreetmap? *Relevant Software*. Available at: https://relevant.software/blog/choosing-a-map-amapbox-google-maps-openstreetmap/ [Accessed September 21, 2022].

E. Codd, "A relational model of data for large, shared data banks", Communications of the ACM, vol. 13, no. 6, pp. 377-387, 1970.

Enlyft, REACT commands 6.26% market share in software frameworks. *Enlyft*. Available at: https://enlyft.com/tech/products/react [Accessed September 21, 2022].

Facebook, Facebook/react: A declarative, efficient, and flexible JavaScript library for building user interfaces. *GitHub*. Available at: https://github.com/facebook/react [Accessed September 21, 2022].

Galov, N., 17+ google maps statistics to survey in 2022. *WebTribunal*. Available at: https://webtribunal.net/blog/google-map-statistics/#:~:text=Google%20Maps%20has%20154.4%20million,pieces%20of%20information%20per%20day. [Accessed September 21, 2022].

García, A., 2022. Why is react declarative? A story about function components. *Medium*. Available at: https://medium.com/trabe/why-is-react-declarative-a-story-about-function-components-aaae83198f79 [Accessed September 21, 2022].

GeoJSON, Geojson. *GeoJSON*. Available at: https://geojson.org/ [Accessed September 23, 2022].

Gillin, P., 2022. What is component-based architecture? *Mendix*. Available at: https://www.mendix.com/blog/what-is-component-based-architecture/#:~:text=The%20benefits%20are%20reduced%20development,replacing%20components%20without%20major%20disruption. [Accessed September 22, 2022].

GitHub, The state of the octoverse: The state of the Octoverse explores a year of change with new deep dives into writing code faster, creating documentation and how we build sustainable communities on github. *The State of the Octoverse | The State of the Octoverse explores a year of change with new deep dives into writing code faster, creating documentation and how we build sustainable communities on GitHub.* Available at: https://octoverse.github.com/ [Accessed September 21, 2022].

Gleb, K., 2022. How to set up a scalable react architecture: Tips and best practices to follow. *Geniusee*. Available at: https://geniusee.com/single-blog/react-architecture-best-practices-and-tips#:~:text=A%20good%20practice%20while%20working,to%2C%20without%20affecting%20the%20architecture. [Accessed September 22, 2022].

Google, *Google Maps Platform Billing | google developers*. Available at: https://developers.google.com/maps/billing-and-pricing/billing#monthly-credit. [Accessed September 21, 2022].

Graglia, D., 2022. *How many survey responses do I need to be statistically valid? | SurveyMonkey*. [online] SurveyMonkey. Available at: <https://www.surveymonkey.co.uk/curiosity/how-many-people-do-i-need-to-take-my-survey/> [Accessed 20 August 2022].

Hatton, S., 2008. *Choosing the "Right" Prioritisation Method*. [ebook] Crawley, Western Australia: Highly denormalized schema: Enhanced query performance, Doc-archives.microstrategy.com, 2021. [Online]. Available: https://docarchives.microstrategy.com/producthelp/10.4/ProjectDesignGuide/WebHelp/Lang_1033/Content/Proj ectDesign/Highly_denormalized_schema__Enhanced_query_perform.htm. [Accessed: 26-Nov- 2021

I. T. L. Education Solutions Limited (2009). Introduction to Information Technology. Pearson Education India. p. 522. ISBN 978-81-7758-118-8.

JavaTPoint, Pros and cons of reactjs - javatpoint. *www.javatpoint.com*. Available at: https://www.javatpoint.com/pros-and-cons-of-react [Accessed September 21, 2022].

JMango360, 2022. Mobile app versus mobile website statistics: 2020 and beyond. *JMango360*. Available at: https://jmango360.com/mobile-app-vs-mobile-website-statistics/#:~:text=Mobile%20Apps%20vs.,-Mobile%20Websites%3A%20User&text=That's%20because%20consumers%20prefer%20apps,the%20most%20after%20news%2Dapps. [Accessed September 18, 2022].

Juviler, J., 2022. *The Bulma CSS Framework: What It Is and How To Get Started*. [online] Blog.hubspot.com. Available at: <https://blog.hubspot.com/website/bulma-css> [Accessed 23 August 2022].

Kano, 2022. *Kano+ | Kano studies online*. [online] Kano.plus. Available at: <https://www.kano.plus/#kano> [Accessed 31 August 2022].
Kelly, C. and Whyte, N., 2019. *Causeway Coast and Glens District Council*. [online] Ark.ac.uk. Available at: <https://www.ark.ac.uk/elections/nlgccg.htm> [Accessed 9 August 2022].

Kin, K; Schaap, P (2021). "Evolution of Multicellular Complexity in The Dictyostelid Social Amoebas". *Genes*. **12** (4): 487. doi:10.3390/genes12040487. PMC 8067170. PMID 33801615

Leaflet, Documentation - leaflet - a JavaScript library for interactive maps. *Leaflet*. Available at: https://leafletjs.com/reference.html [Accessed September 21, 2022].

Leaflet, Leaflet/leaflet: 🍃 javascript library for mobile-friendly interactive maps 🇺🇦. *GitHub*. Available at: https://github.com/Leaflet/Leaflet [Accessed September 21, 2022].

LeafletProvider, *Leaflet provider demo*. Available at: https://leaflet-extras.github.io/leaflet-providers/preview/ [Accessed September 23, 2022].

LeSuer, S., 2022. One page website vs Multiple Pages Pros & Cons: Which is best? *Slickplan*. Available at: https://slickplan.com/blog/one-page-website-vs-multiple-pages#:~:text=Usually%20a%20single%2Dpage%20will,which%20means%20more%20load%20times . [Accessed September 22, 2022].

Livingston, M., 2016. Which technology stack / frameworks does react.js fit most ... - quora. Available at: https://www.quora.com/Which-technology-stack-frameworks-does-React-js-fit-most-naturally-into [Accessed September 21, 2022].

Marcak, M., 2022. React native pros and cons - 2022 updated. *Pagepro*. Available at: https://pagepro.co/blog/react-native-pros-and-cons/ [Accessed September 17, 2022].

MDNWebDocs, 2022. *JavaScript | MDN*. [online] Developer.mozilla.org. Available at: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> [Accessed 1 September 2022].
Miller, G., 1994. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 101(2), pp.343-352.

MindTools, 2016. *Kano Model Analysis: Delivering Products That Will Delight*. [online] mindtools.com. Available at:

<https://www.mindtools.com/pages/article/newCT_97.htm#:~:text=What%20Is%20Kano%20Model%20Analysis,it's%20also%20about%20customers'%20emotions.> [Accessed 19 August 2022].
Ngo-The, A., Ruhe, G. 2005. Decision Support in Requirements Engineering. In: Aurum, A., Wohlin, C. (eds) Engineering and Managing Software Requirements. Springer, Berlin, Heidelberg. Available at: < https://link.springer.com/chapter/10.1007/3-540-28244-0_12> [Accessed 20 August 2022].

NINIS, 2020. *NINIS: Northern Ireland Neighbourhood Information Service*. [online] Ninis2.nisra.gov.uk. Available at: <https://www.ninis2.nisra.gov.uk/public/AreaProfileReportViewer.aspx?FromAPAddressMulipleRecords=Causeway%20Coast%20And%20Glens@@Causeway%20Coast%20And%20Glens@22?#:~:text=The%20estimated%20population%20of%20Causeway,73%2C104%20(50.4%25)%20were%20female.> [Accessed 30 August 2022].

NIWorld, 2020. *New RNLI and HM Coastguard Campaign*. [online] northernirelandworld.com. Available at: <https://www.northernirelandworld.com/news/new-rnli-and-hm-coastguard-campaign-2860751)> [Accessed 6 August 2022].

Noriaki Kano, Nobuhiko Seraku, Fumio Takahashi, and Shin-ichi TSUJI 1984. 'Attractive Quality and Must-Be Quality,' *Journal of the Japanese Society for Quality Control*, Vol. 14, No. 2, pp. 147-156.

OpenWeatherMap, Pricing. *OpenWeatherMap*. Available at: https://openweathermap.org/price [Accessed September 23, 2022].

OSM, API. *API - OpenStreetMap Wiki*. Available at: https://wiki.openstreetmap.org/wiki/API [Accessed September 21, 2022].

Ozanich, A., 2022. *TailWind CSS vs Bootstrap: What is The Difference & Which One is Best?*. [online] Blog.hubspot.com. Available at: <https://blog.hubspot.com/website/tailwind-css-vs-bootstrap> [Accessed 21 August 2022].

Peabody, B., 2017. Server-side I/O performance: Node vs. PHP vs. Java vs. go. *Toptal Engineering Blog*. Available at: https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go [Accessed September 21, 2022].

PeldonRose, The benefits of blue in design. *Peldon Rose*. Available at: https://www.peldonrose.com/news-insight/features/the-benefits-of-blue-in-design/ [Accessed September 22, 2022].

Pumpkin Web Design Manchester, 2019. Advantages of monospaced typeface in web design. *Pumpkin Web Design Manchester*. Available at: https://www.pumpkinwebdesign.com/web-design-manchester/advantages-of-monospaced-typeface-in-web-design/ [Accessed September 22, 2022].

R. Rustin, Data base systems, 6th ed. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

React, Testing overview. *React*. Available at: https://reactjs.org/docs/testing.html [Accessed September 22, 2022].

ReactLeaflet, React leaflet documentation. *React Leaflet*. Available at: https://react-leaflet.js.org/ [Accessed September 21, 2022].

Rees, D., What is wireframing. *Experience UX*. Available at: https://www.experienceux.co.uk/faqs/what-is-wireframing/ [Accessed September 22, 2022].

ReqExperts, *Writing good requirements*. Available at: https://reqexperts.com/wp-content/uploads/2015/07/writing_good_requirements.htm#:~:text=A%20good%20requirement%20states%20something,is%20not%20a%20good%20requirement. [Accessed September 23, 2022].

RNLI, 2021. *RNLI lifeguards in Northern Ireland rescue 3,701 people in a decade of patrols*. [online] rnli.org. Available at: <https://rnli.org/news-and-media/2021/june/23/rnli-lifeguards-in-northern-ireland-rescue-3701-people-in-a-decade-of-patrols> [Accessed 18 August 2022].

RNLI, 2021. *RNLI: Annual Report and Accounts 2021*. [ebook] Royal National Lifeboat Institution (RNLI), pp.5,6,9. Available at: <https://rnli.org/about-us/how-the-rnli-is-run/annual-report-and-accounts> [Accessed 22 August 2022].

RNLI, 2021. *Running Costs*. [online] rnli.org. Available at: <https://rnli.org/about-us/how-the-rnli-is-run/running-costs#:~:text=The%20RNLI%20is%20funded%20primarily,RNLI%20receives%20no%20government%20funding.> [Accessed 20 August 2022].
School of Computer Science and Software Engineering, The University of Western Australia. Available at: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4483241> [Accessed 20 August 2022].

Singh, J., 2018. Google maps API price hike is threatening the future of some companies. *Gadgets 360*. Available at: https://gadgets360.com/apps/features/google-maps-apis-new-pricing-impact-1907242#:~:text=Currently%2C%20Google%20Maps%20API%20fees,to%20100%2C000%20per%2024%20hours. [Accessed September 21, 2022].

Slant, Google Roboto Mono review. *Slant*. Available at: https://www.slant.co/options/287/~google-roboto-mono-review [Accessed September 22, 2022].

Swistak, T. & Stempniak, A., 2022. What is typescript? pros and cons of typescript vs. JavaScript. *stxnext*. Available at: https://www.stxnext.com/blog/typescript-pros-cons-javascript/ [Accessed September 21, 2022].

TechStrikers, MySQL Advantages and Disadvantages. *MySQL Advantages and disadvantages*. Available at: https://www.techstrikers.com/MySQL/advantages-and-disadvantages-of-mysql.php [Accessed September 21, 2022].

TechStrikers, The Good and the Bad of Node js Web App Development *TechStrikers*. Available at: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/ [Accessed September 21, 2022].

Tondon, S., 2022. The Pros and cons of Node.js Web App Development: A detailed look. *Medium*. Available at: https://javascript.plainenglish.io/the-pros-and-cons-of-node-js-web-app-development-a-detailed-look-c91a22f013c [Accessed September 21, 2022].

TourismNI, 2022. *Tourism Performance Statistics*. [online] Tourism NI. Available at: <https://www.tourismni.com/industry-insights/tourism-performance-statistics/> [Accessed 15 June 2022].

Usability, 2014. User interface design basics. *Usability.gov*. Available at: https://www.usability.gov/what-and-why/user-interface-design.html [Accessed September 22, 2022].

VisitBritain, 2014. *Britain's visitor economy facts*. [online] VisitBritain.org. Available at: <https://www.visitbritain.org/visitor-economy-facts> [Accessed 18 August 2022].

Volodymyr, 2021. 25 pros and cons of using PHP for web development. *SapientPro*. Available at: https://sapient.pro/blog/25-pros-and-cons-of-using-php-for-web-development/ [Accessed September 21, 2022].

W3techs, 2022. *Usage Statistics and Market Share of Bootstrap for Websites, September 2022*. [online] W3techs.com. Available at: <https://w3techs.com/technologies/details/js-bootstrap> [Accessed 1 September 2022].

W3Techs, 2022. *Usage Statistics of JavaScript as Client-side Programming Language on Websites, September 2022*. [online] W3techs.com. Available at: <https://w3techs.com/technologies/details/cp-javascript/> [Accessed 1 September 2022].

Wagner on Aug 18, J., 2020. Radeventlistener: A tale of client-side Framework Performance: CSS tricks. *CSS Tricks*. Available at: https://css-tricks.com/radeventlistener-a-tale-of-client-side framework-performance/ [Accessed September 21, 2022].

Washburn, 2019. 4 reasons to separate your web app's UI and api. *michaelwashburnjr.com*. Available at: https://michaelwashburnjr.com/blog/4-reasons-web-app-separated-frontend-backend [Accessed September 22, 2022].

Wikipedia, 2022. Comparison of JavaScript-based web frameworks. *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Comparison_of_JavaScript-based_web_frameworks [Accessed September 21, 2022].

Wikipedia, 2022. *Kano model - Wikipedia*. [online] En.wikipedia.org. Available at: <https://en.wikipedia.org/wiki/Kano_model#:~:text=The%20Kano%20model%20is%20a,customer%20preferences%20into%20five%20categories.> [Accessed 18 August 2022].

Wikipedia, 2022. PHP. *Wikipedia*. Available at: https://en.wikipedia.org/wiki/PHP [Accessed September 21, 2022].

Wikipedia, 2022. TypeScript. *Wikipedia*. Available at: https://en.wikipedia.org/wiki/TypeScript [Accessed September 21, 2022].

World Travel & Tourism Council, 2022. *2022 Annual Research: Key Highlights*. [ebook] World Travel & Tourism Council. Available at: <https://www.statista.com/statistics/598093/travel-and-tourism-gdp-total-contribution-united-kingdom-uk/> [Accessed 20 August 2022].

Yong, E., 2021. Slime mould attacks simulates Tokyo rail network. *Science*. Available at: https://www.nationalgeographic.com/science/article/slime-mould-attacks-simulates-tokyo-rail-network [Accessed September 17, 2022].

Appendices

Appendix I - Additional Reference Material

API URL

- http://mcondren03.webhosting6.eeecs.qub.ac.uk/beachsmart/beachsmart-api/admin/points-of-interest-all.php

Figma Files

- Front End - https://www.figma.com/file/49tYoextNSqm2byD2Ql4Tv/CCG-BeachSmart-Wireframe_v01?node-id=0%3A1
- Back End - https://www.figma.com/file/wHY9bBGSdm5sSe2tUqOcmO/CCG-Admin-Wireframe?node-id=0%3A1

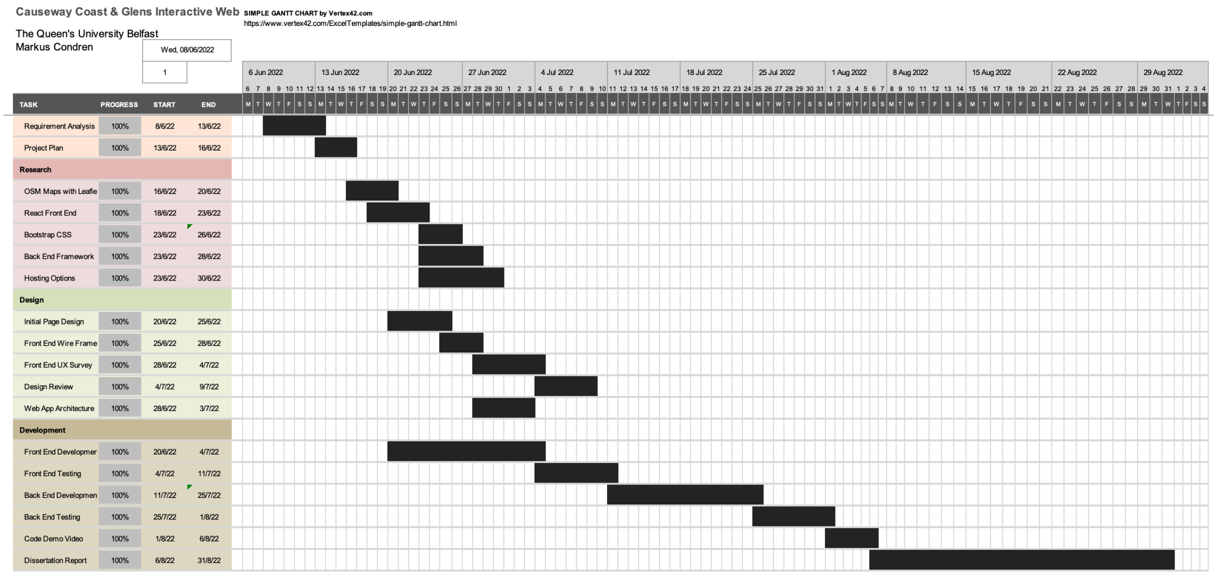
Courses

- FreeCodeCamp JavaScript - https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/
- Scrimba React - https://scrimba.com/learn/learnreact

Video References

- Figma
  - Figma UI Design Tutorial: Get Started In Just 24 Minutes! - https://www.youtube.com/watch?v=FTFaQWZBqQ8
  - Free Figma Tutorial: Designing Wireframes with Figma - https://www.youtube.com/watch?v=6t_dYhXyYjI
  - How to Wireframe with Figma - https://www.youtube.com/watch?v=k6ALKYIh2A8
- GIMP
  - How to Customize the GIMP User Interface - https://www.youtube.com/watch?v=UrKoTLt8ROc
  - GIMP basics Use Guides To Trace A Image - https://www.youtube.com/watch?v=sKcMNWsi4-E
  - GIMP Tutorials Playlist from TJFREE - https://youtube.com/playlist?list=PLqazFFzUAPc4vITMJaF3Fnqh3pccSMnC4

Appendix II – GANNT Chart



Causeway Coast & Glens Interactive Web

SIMPLE GANTT CHART by Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

The Queen's University Belfast
Markus Condren

Wed, 08/06/2022
1

TASK	PROGRESS	START	END
Requirement Analysis	100%	8/6/22	13/6/22
Project Plan	100%	13/6/22	16/6/22
Research			
OSM Maps with Leafle	100%	16/6/22	20/6/22
React Front End	100%	18/6/22	23/6/22
Bootstrap CSS	100%	23/6/22	26/6/22
Back End Framework	100%	23/6/22	28/6/22
Hosting Options	100%	23/6/22	30/6/22
Design			
Initial Page Design	100%	20/6/22	25/6/22
Front End Wire Frame	100%	25/6/22	28/6/22
Front End UX Survey	100%	28/6/22	4/7/22
Design Review	100%	4/7/22	9/7/22
Web App Architecture	100%	28/6/22	3/7/22
Development			
Front End Developmer	100%	20/6/22	4/7/22
Front End Testing	100%	4/7/22	11/7/22
Back End Developmen	100%	11/7/22	25/7/22
Back End Testing	100%	25/7/22	1/8/22
Code Demo Video	100%	1/8/22	6/8/22
Dissertation Report	100%	6/8/22	31/8/22