Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Fuelinator

View on Github

By

Cian Hession

for

Gerard Harrison, Examination Board

April 24, 2023

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

B.Sc. (Hons) in Software Development

Contents

```
www.GitHub.com/CianHession/Fuelinator
```

Chapter 1

Introduction

Figure 1.1: Fuelinator Logo

I am a 4th Year Student Studying Software Development at Atlantic Technological University Galway, this dissertation is for my final year project, my project is called Fuelinator for the Module Applied Project And Minor Dissertation. where the module has offered me experience working on a project similar to one encountered in the software industry, in collaboration with academic supervisors and technical staff.

So, What IS Fuelinator you might be asking yourself, well in simplest terms, it is an application that allows the people of Ireland to find the cheapest petrol or diesel prices and the station offering these prices in Ireland, or if they wish, they can select any county in Ireland and find the stations offering the lowest prices for both fuel types, the user sees the map of Ireland, along with all the stations we have data for, which are marked with fuel icons, the user can interact with the map, but zooming in and out, moving the map in any direction and allowing the user to click a marker and the marker then shows to the left of the map the stations name, address, and its prices.

I would like to think the idea seems fairly basic, however, I strongly believe it offers an exceptionally useful purpose and the concept of this app is fairly unique to Ireland as the only other service I was able to find that is offering this service within Ireland is a website called 'pumps. ie' which seems to be outdated and in my opinion an application is more relevant than a web-based service, the only positive to using a web-based service is the fact you don't need to download anything to use, whereas for the application you are required to download it first.

Fuelinator was developed using React.js, which then uses the Google Maps API to provide a map interface, it also uses MongoDB which is for storing data in this case all the data in relation to the stations, Firebase which is developed by Google and offers very effective user authentication, which is a massive element of nearly all modern services, and finally, Node.js which allows me to connect my front end; the map with these other services. The application's main purpose is to allow the users to find the cheapest fuel prices of any county in Ireland or of Ireland itself, making this application aimed at use by Irish-based users. The user is given the ability to select between finding the cheapest diesel price or petrol prices in Ireland. If however, the user wants to find a county's cheapest price, they can use a drop-down menu allowing them to select the county they wish. When the user selects a county, the markers for the other counties are hidden and the user can see a visual representation of markers for fuel stations of the selected county and update fuel prices if they are not correct or have changed, the user can also select any marker on the map and then they will be shown the data for the station they clicked on.

The idea came to me following the increasing escalation of the Russia and Ukraine war, causing fuel prices to skyrocket globally. I am currently working in a petrol station, I was thinking to myself what is something most humans would find useful on a daily basis but something that is not really out there? and then it dawned on me, from interacting with customers on a daily basis, mainly about fuel prices, an application which would let the user know the fuel prices before arriving, so if they don't like a stations price, they can browse around, near to the location and compare prices.

Now I am going to walk you through the development and thought process of how the application came together.

The first thing I knew I had to do was, create a user interface to show the user fuel stations, so I decided to use Google Maps, why? To me this was the best way to show fuel stations, as we probably use Google Maps more than we even realize, so I felt it would be appropriate to provide a sense of familiarity to the user, once this was done, I then needed a way to actually get data for fuel stations, simple data such as the name of the station, its location, so I used python to scrape through the Google Places API, county by county, taking this information and putting into a JSON File, then I needed a way for my Map to be able to read this data and then do something with it, so I created a MongoDB Collection, which I used my Node.js server to push the data from JSON to this collection, so once I was satisfied with the data I had scraped, I put all the data into one big JSON file, and read it into the server, and I made a fuel station Schema, then pushed the data to MongoDB, so now I was faced with, how do I show this on the map? I was able to do this by looping through the station's data, finding the longitude and latitude from the data I scraped, and for the coordinates putting an icon instead of the provided Marker Icon provided by Google Maps to represent a station.

That was going great, I had a front end that worked as I hoped it would, but then I needed to add more functionality, such as being able to click a marker, the marker then showing the station data for the marker, I eventually got it working and was powering on with developing, but then I remembered the app should allow the user to select a county, as that was part of the aim of the application, so I made an array for all 32 counties on the Island of Ireland, then I created methods for if a county is selected we would only show markers for that county based on the station's address, mainly a hidden variable called county which is stored with the station data on MongoDB, this would be key to doing this, as it allowed the filtering to be rather easy, and hiding other counties markers, making the app feel effective.

Now that the front end was completed with the back end having been developed according to the required functionality, I felt something was missing from the app, and I was correct, I again used a Python script that went through the large JSON file, I coded it to input diesel and petrol prices for every single station, whilst keeping true to; in Ireland, Diesel is cheaper than Petrol and to also be generated in a range from 1.00 - 2.10, so the prices appear realistic enough, whilst keeping true to Diesel being cheaper, after that was done and I read it in, I wanted to let the user to be able to update the prices, with the aim of people keeping the prices updated.

Once I was satisfied that had been completed and the application functioned correctly, I wanted and needed to create a way for user accounts to be created, so I looked up some ways of doing that and picked Firebase for 2 reasons, the first being it was a new technology for me to use and the second being the fact Google developed Firebase, given I was using Google Maps/Console, I figured I would stick to using them for this part.

Once all that was done, I was happy to mark the project as being completed, at least what I would like to refer to as version 1 completed, I have aims to flesh out this project and expand its functionality in the future, one such function which I feel is mandatory would be for allowing the user to search locations by using Auto-Complete, and filtering out non-searched based markers so for instance if I selected Tuam, Co. Galway only the markers for stations in Tuam would appear just like what happens when you select a county in the current state, another piece of functionality I hope to add would be the ability to compare prices of one station to another, I feel this isn't overly important as you can do this manually with clicking markers or you could just use the filter by cheapest option. I think it would also be important to add some sort of label to indicate when prices were last updated and a warning if the price had passed 24 hours without being updated, that the price may no longer be accurate as fuel stations can change their prices at any given time, I think having the ability to upvote or downvote submitted prices would be great in showing the accuracy of the price, this would go hand in hand with this feature, meaning other users of the application, can verify if they visited a station and the submitted price was correct, and once a price has updated the counter for thumbed up/down is reset, and voting again can begin, that's just some of the ideas I have of right now, there is definitely more features that could be added and I look forward to discovering them and eventually implement them, I know fleshing it out would require me to expand on the users account, maybe giving them a dashboard, I also need to add a way for the user to reset their password, but if I really want this app to be high quality, I would be willing to do that.

In order to use Fuelinator users can create an account and then login to their account, I made this a requirement in order to be able to use Fuelinator, so long as the user was registered, they then can use the application fully and submit prices for any of the stations marked on the map.

Upon completing the project prices have gone back to almost normal standard pre-global hike, I felt the application served a good purpose regardless as it allows for budgeting if needed, but more importantly, providing the user the station's prices before arriving.

MongoDB is used to store the stations' data, as of now I have 1365 stations, whilst there are many more In Ireland, I set out with an aim to constantly update the application even after this project is due, and hope to deploy it as an actual published application.

Firebase will be used for User Authentication, meaning Account Creation and then logging in, Firebase also allows users to contact me with any queries, almost like contacting me through a business email but instead logs these queries in a collection for real-time data on Firebase, and if necessary I can reach them on their provided email.

Node.js Will be used for connecting the front end and MongoDB, this allows me to pull and push data from this data store.

The only thing the user requires is a Google Cloud Membership, If I do make this into an actual application post-graduation, I would of course provide my own API Key for Google Maps and Console so that the user doesn't need to do anything extra, but for now, the user can claim a free API Key and a Google Cloud membership trial for 3 months with 300 dollars trial credits, which is more than enough to use this application, the reason I don't provide my own key right now is that it would eat into my own pockets, and throughout the development of this application I have had to pay small fees due to using the API for testing and using the application, I would probably incorporate adverts, mainly non-content covering adverts so the content of the application is not covered but is somewhere to generate revenue or seek partnerships with fuel companies so they can buy into Fuelinator and in turn we could work together to provide live price updates, keeping data 100 percent accurate, without relying on users of the application to keep prices up to date, for example say Circle K liked the idea, we could work together to have all registered Circle K stations on the map, and not allow users to update their prices, but instead Circle K feed real time data through some means to my server, providing the best possible implementation of tracking prices.

And with that being said, that is the introduction section for Fuelinator complete! I hope you enjoyed reading, and are eager to continue reading on to learn more technical reasoning and an advanced breakdown of the final product, and ultimately come to understand the reasoning behind why I chose to do certain things over others, and what worked and did not work.

Chapter 2

Methodology

As outlined in the introduction section, I had a clear plan in my mind, first was to create a nice user interface, which was implemented with Google Maps and React, and then a way of using data to represent fuel stations, which was done via MongoDB and Node.js, but I then decided I wanted the users of the application to actually have and use accounts in order to prevent just anyone updating fuel prices.

Below is the methodology for how I handled all this.

The Fuelinator React application is a web-based tool for finding and comparing fuel prices at various fuel stations in Ireland from a collection of more than 1300 stations. The application uses the Google Maps API to display a map with markers representing fuel stations which have been allocated unique icons which allow users to interact with these markers to view and update fuel prices for the station marker they click on. The methodology for this dissertation will focus on the design and implementation of the Fuelinator application, including the technologies and tools used, the data sources and processing, and the user interface and interaction.

Technologies and Tools: The Fuelinator application is built using React, a popular JavaScript library for building user interfaces, and it uses several other libraries and tools to enhance its functionality. The main technologies and tools used in the development of the application are:

React: React is a JavaScript library for building user interfaces. It allows developers to create reusable UI components and provides a fast and efficient way to update the user interface when the application state changes.

google-maps-react: google-maps-react is a library that provides a simple and easy-to-use interface for integrating Google Maps into a React application. It allows developers to display maps, markers, and other map-related components and provides event handlers for interacting with the map.

Bootstrap: Bootstrap is a popular CSS framework that provides a collection of responsive and customizable UI components. It is used in the Fuelinator application to create a clean and modern user interface with consistent styling.

react-router-dom: react-router-dom is a library for handling routing in a React application. It is used in the Fuelinator application to handle navigation between different views and components.

fetch API: The fetch API is a modern JavaScript API for making HTTP requests. It is used in the Fuelinator application to fetch data from the backend server, such as fuel station data and updates.

Data Sources and Processing: The data for the Fuelinator application is obtained from a backend server through HTTP requests. The backend server provides RESTful APIs for retrieving and updating fuel station data. The data is stored in a MongoDB database, which is a popular NoSQL database that stores data in a flexible, JSON-like format called BSON.

The Fuelinator application uses the fetch API to make HTTP requests to the backend server and retrieve fuel station data. The data is returned in JSON format, which is a lightweight data-interchange format that is easy to parse and manipulate in JavaScript. The application uses the JSON data to populate the map with markers representing fuel stations and to display fuel prices and other information in the InfoWindow component when a marker is clicked.

The application also allows users to update fuel prices by submitting a form with the updated prices. The form data is sent to the backend server as a JSON payload in the request body, and the server updates the corresponding fuel station record in the database with the new prices. The updated data is then returned to the client as a JSON response, and the application uses the response data to update the UI with the new prices.

User Interface and Interaction: The user interface of the Fuelinator application is designed to be simple and intuitive, with a map displaying fuel stations as markers, and an InfoWindow component that shows detailed information about a selected fuel station when its marker is clicked. The application provides several features for users to interact with the map and the markers, including:

Marker Click: When a user clicks on a marker representing a fuel station, the application fetches the fuel station data from the backend server and displays it in the InfoWindow component. The user can view the current fuel prices and other information, and can also update the fuel prices by submitting a form in the InfoWindow.

Form Submission: The application allows users to update fuel prices by submitting a form in InfoWindow. The form includes input fields for entering the new prices for each fuel type, as well as a submit button to send the data to the backend server. When the form is submitted, the application sends an HTTP request with the updated data to the server, which updates the corresponding fuel station record in the database with the new prices.

Search Functionality: The application provides search functionality that allows users to search for fuel stations by their county. The search function filters by comparing the address and the county variable which I obtained from the Google Places API to match the user's request for finding stations on the map.

Filtering: The application provides a filtering functionality that allows users to filter fuel stations by county. The filtering function updates the markers on the map to show only the fuel stations that match the selected criteria, hiding non relevant markers.

In conclusion, the Fuelinator React application is a web-based tool that provides users with an easy and intuitive way to find and compare fuel prices at various fuel stations in Ireland. The application uses several technologies and tools, such as React, Google Maps API, Bootstrap, and fetch API, to enhance its functionality and provide a responsive and modern user interface. The data for the application is obtained from a backend server through HTTP requests and is stored in a MongoDB database. The user interface of the application is designed to be simple and intuitive, with several features for interacting with the map and the markers, such as marker click, form submission, search functionality, filtering, and location-based services.

The Node.js server was built using the Express framework, which serves as an API for accessing and modifying data on a MongoDB database. The server listens on a port, which is either specified in an environment variable or defaults to 3001.

The server uses the Mongoose library to connect to the MongoDB database and define a schema for the FuelStation objects that will be stored. The schema specifies the fields that each FuelStation object should have, including the name, address, latitude, longitude, rating, county, petrolPrice, and dieselPrice.

The server loads a JSON file of fuel station data and iterates through each station in the file. For each station, the server checks if there is already a FuelStation object in the database with the same name and address. If there is, it does nothing. If there isn't, it creates a new FuelStation object with the data from the JSON file and saves it to the database.

The server provides several endpoints for accessing and modifying the FuelStation objects in the database. The endpoint /API/fuelstations returns a list of all the FuelStation objects in the database. The endpoint /API/fuelstations/:id returns a single FuelStation object with the specified ID. The endpoint /API/fuelstations/:id/ allows the petrolPrice and dieselPrice fields of a FuelStation object to be updated.

In addition to these endpoints, the server provides a /API/key endpoint that returns the Google Maps API key that the client will need to access the Google Maps API. The server also provides a default endpoint that returns a simple message indicating that the API is running.

Overall, the server is built using a RESTful architecture, where each endpoint maps to a specific CRUD operation (Create, Read, Update, Delete) on the FuelStation objects in the database.

The User Authentication is done via Firebase which represents a React component that renders a login/signup form, allowing users to either login to an existing account or create a new account. The code uses a variety of React hooks, including useState and useNavigate, as well as an external Firebase library for user authentication.

The useState hook is used to manage the state of several variables, including loginEmail, loginPassword, signupEmail, signupPassword, signupPasswordError, errorMessage, and showLogin. These variables are used to control the behavior of the form and manage user input.

The useNavigate hook is used to manage programmatic navigation within the app. In this case, it is used to redirect the user to a protected route (/fuelinator) after successful login or signup.

The auth, createUser, and loginUser functions are imported from the external Firebase library and used to handle user authentication. The handleLogin and handleSignup functions are defined to handle user input and perform the appropriate authentication and navigation actions based on the user's actions.

The handleToggle function is defined to allow users to switch between the login and signup forms by toggling the value of showLogin.

The component's return statement contains the JSX code that defines the structure and behavior of the login/signup form. Conditional rendering is used to display either the login or signup form based on the value of showLogin.

Overall, this component demonstrated how I made the user authentication through a clean approach to build a login/signup form in a React application.

Chapter 3

Technology Review

The main code for the application called Fuelinator is developed using React, fuelinator is a function that renders a Google Map using the 'Google-maps-react' package. It also provides a search feature to allow the users of the application to find fuel stations on the map, whilst also using Firebase for user authentication, as well as cloud storage and messaging. Node.js was used in conjunction with MongoDB and Express to handle data messaging to and from Fuelinator.

I used Visual Studio Code as my IDE for the process of development, Visual Studio Code is a free and open-source code editor developed by Microsoft. It's available for Windows, macOS, and Linux and is designed to be lightweight, fast, and highly customizable to match your needs and preferences.

Visual Studio Code is popular among developers for several reasons:

It supports a wide range of programming languages and frameworks, including JavaScript, Python, PHP, and many more.

It has a rich extension ecosystem that allows developers to add new features and functionality to the editor.

It has a built-in debugger that makes it easy to find and fix errors in code.

It has a powerful IntelliSense feature that provides code completion, syntax highlighting, and other helpful hints as you type.

It's free and open-source, making it accessible to developers of all skill levels and budgets.

However, there are some downsides to Visual Studio Code as well:

Some users have reported performance issues when working with larger projects or code bases.

The user interface can be overwhelming at first, and it may take some time to customize it to your liking.

Some users have issues with the Git integration and other features, although these issues are usually fixed quickly with regular updates.

Overall, Visual Studio Code is an excellent code editor that's highly recommended by many developers, myself included. However, like anything in life, it may not be the best fit for everyone, and users should evaluate it based on their individual needs and preferences, but I highly enjoy using it and recommend it.

I decided to use React as I had planned on using Algolia with my code, I was planning on using this due to it offering to search, and it also is extremely fast, I also planned to use Mapbox for the front-end development but decided I wanted to either use the Google Maps API or else leaflet.js, ultimately I decided upon using Google Maps, as I felt we use Google Maps far more than we realize and felt it offered a sense of familiarity to the users, whilst being enjoyable reading the react documentation for the google-maps-react package and trying to implement it into my code, I did not get time to use these technologies enough to use them with my project sadly but I have began using them and will be keen on using them more and probably implement more functions and features to my code post submission in order to complete the project at a higher standard for my CV.

As mentioned, Google Maps renders markers using the Google Maps JavaScript API. Map, Marker, and InfoWindow components from Fuelinator are used from the google-maps-react library to render the map and the markers.

To render a marker, I created a Marker component and provided its position prop with the location of the marker, which can be a google.maps.LatLng object or an object with lat and lng properties, which is what I used in this case. I could also have provided other props to customize the marker, such as an icon which I have implemented, label, title, and onClick, which again I have used.

I used 1 useEffect hook, which created a script element that was dynamically added to the DOM in order to load the Google Maps JavaScript API with the API Key. Once the API loads, the Google object is set in the state using the setGoogle function.

The Map component is then rendered with the Google object and other props such as initialCenter, zoom, and style, which I Set at the center point of Ireland, zoomed out. The Marker component is rendered inside the Map component using a map function to iterate over the fuelStations array and create a marker for each station. The selectedMarker state is used to determine which marker to highlight, and the handleMarkerClick function is used to handle the marker click event and show an InfoWindow with additional information about the station.

When I had decided that I was going to be using React, despite it not being a new technology in relation to our course, I thoroughly enjoyed using it in previous years, I mainly went with React as it is one of the most popular JavaScript libraries for building user interfaces and is used by many companies and organizations worldwide. React's popularity is due in part to its ease of use, modularity, and the large range of tools and libraries which have been developed around it.

Additionally, React's virtual DOM allows for efficient updates to the UI, making it particularly well-suited for large, complex applications. React is also supported by Facebook, which provides ongoing development and support for the library, I suppose when I started developing this project, I had thought it might have been fairly complex and felt React fitted the requirements to be able to handle any complex issues I threw at it.

I loved using React and learning how to use some of the packages offered, I would genuinely be interested in beginning my career using React, it is so simple to use and can be fairly effective and powerful if used properly, however, React isn't flawless, below I will go over some pros and cons to using it.

Advantages of using React for building user interfaces:

Component-based architecture: React's component-based architecture allows developers to break down UIs into smaller, reusable pieces of code. This makes it easier to manage complex UIs and encourages code reusability.

Virtual DOM: React's virtual DOM allows for efficient updates to the UI without having to re-render the entire page. This leads to improved performance and faster UI updates. Large community: React has a large and active community of developers who contribute to the framework and create open-source libraries that can be used to extend its functionality.

Reusability: Since React components are modular and reusable, they can be used in multiple places throughout an application, leading to less code duplication and easier maintenance. Cross-platform development: React can be used to develop both web and mobile applications, allowing developers to write code once and deploy it across multiple platforms.

Disadvantages of using React for building user interfaces:

Steep learning curve: React has a higher learning curve compared to other frameworks, especially for developers who are new to the concept of virtual DOM and JSX syntax.

JavaScript-heavy: React is a JavaScript library, so developers need to have a good understanding of JavaScript to work with it effectively.

Lack of opinionated structure: Unlike some other frameworks, React does not have a strong opinion on how developers should structure their code. This can lead to inconsistencies across projects and make it harder to onboard new developers.

Examples of how React's modularity and virtual DOM were used to improve the performance and scalability of Fuelinator:

I used React's modularity to import only the necessary components from external libraries such as Google Maps and Bootstrap. By importing only what is needed, the application loads faster and reduces the overall bundle size.

The virtual DOM, on the other hand, ensures that only the necessary components are re-rendered when there is a change in the application state. In my code, React tracks changes to the state and updates only the necessary components in the DOM. For example, when the selected marker state is updated, React updates only the InfoWindow component related to the marker rather than re-rendering the entire map. This ensures that the application is efficient and performs well even when dealing with a large number of components.

The useEffect hook is also used to improve the performance of the application. The hook is used to fetch data from the backend and update the state accordingly. The handleMarkerClick function, for instance, fetches data from the backend when a marker is clicked and updates the state with the new data. Similarly, the useEffect hook used to update the selected marker when the selectedCounty changes, ensures that only the necessary components are re-rendered.

I had considered using Python to create this project too but felt it would be graded similarly to me using React, given we have used it in previous years, and even used it this semester in the course for Theory Of Algorithms, I also considered using Ruby-On-Rails, but again we were going to be using this technology for our module Software Engineering.

Now Looking at the code, I will walk through it a little bit.

Fuelinator uses the useState hook to manage its state, including: google: An object representing the Google Maps API, initialized as null. fuelStations: An array of fuel station objects, initialized as an empty array. selectedMarker: An object representing the selected marker on the map, initialized as null. selectedCounty: A string representing the selected county for the search feature, initialized as an empty string. editMode: A boolean representing whether or not the user is in edit mode, initialized as false. The component also uses the useEffect hook to update the state in response to changes in the fuelStations and selectedCounty variables.

Fuelinator includes several helper functions, which over the duration of the course has been key in our development skills progression as they are so useful as they offer re-usability, whilst being easy to read and offer encapsulation if editing is necessary, below are the helper methods I created:

1) handlePetrolPriceChange: A function that updates the petrolPrice property of the selectedMarker object.

2) handleDieselPriceChange: A function that updates the dieselPrice property of the selectedMarker object.

3) findCheapestDieselPrice: A function that finds the fuel station with the cheapest diesel price in the selected county (or in all counties if none is selected), and updates the selectedMarker object accordingly.

4)findCheapestPetrolPrice: A function that finds the fuel station with the cheapest petrol price in the selected county (or in all counties if none is selected), and updates the selectedMarker object accordingly.

5)handleFormSubmit: A function that updates the petrol and diesel prices of a fuel station using a PUT request to the server, and updates the state accordingly. handleMarkerClick: A function that fetches the petrol and diesel prices of a fuel station using a GET request to the server, and updates the state accordingly.

Finally, the Fuelinator component renders a Map component from the google-maps-react package, along with several Marker and InfoWindow components representing the fuel stations on the map. The component also renders a search form that allows the user to filter fuel stations by county and find the cheapest petrol and diesel prices in the selected county.

The findCheapestDieselPrice() and findCheapestPetrolPrice() functions were created to find the cheapest fuel prices for diesel and petrol, respectively. These functions are called when the user selects a county, or when the application first loads if no county is selected.

Both functions use a loop to iterate over the fuel station array, comparing the fuel prices of each station to a variable that is initially set to the maximum possible number. If the price of the current station is lower than the current minimum, the minimum is updated to the price of the current station, and the current station is saved as the station with the cheapest fuel price so far.

If the selectedCounty variable is set to a county, the loop only considers fuel stations in the selected county. Otherwise, all fuel stations are considered.

Once the cheapest fuel station is found, the setSelectedMarker() function is called with the cheapestDieselStation or cheapestPetrolStation as an argument, depending on which function is called from the users selection.

These functions are used in the useEffect() hook that is called when the fuelStations or selectedCounty variables are updated. This hook is used to update the selected marker to the cheapest fuel station in the selected county or in all fuel stations if no county is selected.

In addition to these functions, there are two event handler functions, handlePetrolPriceChange() and handleDieselPriceChange(), that update the selectedMarker object with the new fuel prices entered by the user. These functions are called when the user enters new fuel prices into the input fields on the marker info window.

Finally, the handleFormSubmit() function is called when the user submits the form to update the fuel prices of the selected fuel station. This function sends a PUT request to the server with the updated fuel prices and then updates the fuelStations and selectedMarker objects with the new prices.

In Order for Fuelinator - The Front end to work, I needed a server to handle code and functions whilst also pulling and pushing data to and from MongoDB.

I was able to do This by creating my server using Node.js, I used the server in order so that it created an API for fuel stations. I used the Express framework to handle HTTP requests and responses, and Mongoose to interact with a MongoDB database.

I was contemplating using MySQL for this but felt Node preformed better for React and required one less install in order to use Fuelinator.

The server starts by importing the required modules and setting up the server to listen on a port. It then connects to the MongoDB database using a connection string.

The fuel station schema is defined using Mongoose, and a model is created from this schema. The application then populates the database with fuel station data by iterating through a JSON file combinedFormattedStations.json and saving each station to the database. If a station with the same name and address already exists in the database, it is not saved again to avoid data overloading and also errors for example if I clicked the marker twice it might return different data for the same marker.

The server defines several HTTP routes for retrieving and updating fuel stations. It also defines a route for retrieving a Google Maps API key which is got from the .env file, I used this to protect my key from being pushed to GitHub.

The app.get('/', ...) route responds with a simple message indicating that the server is running, saying "This is the Fuelinator API".

Overall, the server was used to provide a basic API for retrieving and updating fuel station data and serves as an example of how to use Node.js, Express, and Mongoose to build a web application.

I again would strongly recommend using Node as a small projects server, it is fast to interact and send data to where you need it, and there is plenty of documentation and videos to help you if you need it.

Using the React.js front end and Node.js back end, I was able to create exactly what I had envisioned in my mind.

I also used Firebase as I wanted to learn new technologies, and I needed this technology to handle user Authentication, upon researching ways to handle this, I read about a few technologies that were new to me, here are some of the options I considered along with their benefits:

AWS Amplify: Amplify provides an easy-to-use set of tools and services for building scalable and secure cloud-powered applications. Amplify offers built-in authentication features that allow you to quickly and easily add user sign-up, sign-in, and other authentication flows to your React applications.

Auth0: Auth0 is a flexible and scalable platform that provides authentication and authorization as a service. Auth0 supports a variety of authentication methods, including email/password, social login, multi-factor authentication, and more. Auth0 also provides a variety of SDKs and libraries for integrating authentication into your React applications.

Okta: Okta is a cloud-based identity and access management platform that provides a range of authentication and authorization features. Okta supports a variety of authentication methods, including social login, multi-factor authentication, and more. Okta also provides a variety of SDKs and libraries for integrating authentication into your React applications.

Cognito: Amazon Cognito is a fully managed service that provides authentication, authorization, and user management for web and mobile applications. Cognito supports user sign-up, sign-in, and access control, as well as social identity providers such as Google, Facebook, and Amazon.

However, as mentioned in the Introduction I decided upon Firebase due to the fact Google developed and supported it, keeping in tune with the Google Maps API, I was pleasantly surprised at just how easy Firebase was to use, the documentation was very informative whilst providing exactly what you need to find solutions and methods to use it, I was able to connect my React Application and Firebase with very little code, which majority was found and provided within the documentation, on top of that, there were many tutorials and active forums to get support using Firebase if needed.

I was also really impressed with the ability to handle users needing to contact me, using real-time data, the user can submit a contact form which sends data to Firebase, where I can get everything I need, including their email, name, and inquiry, I would suggest using Firebase to any start-up company looking for a useful tool, whilst I used Firebase mainly for user authentication, It can also be used to do more such as cloud hosting, messaging. It really just helps us, developers, time and effort whilst developing our mobile and web applications quickly and easily.

I also attempted to deploy my react app to GitHub Pages to offer the user a trial/live demo of the app, so if they liked it, they could decide to keep using it via GitHub pages or just download the project and run it locally, GitHub Pages is a nice piece of technology as it uses GitHub to deploy the app, it saves you needing to get hosting and then probably pay to host your app online, GitHub pages just take your published code from GitHub and hosts it on your own unique GitHub Pages URL, I also tried using Twilio to host my React app but considering GitHub Pages is free and easy to use it's a very efficient service.

Finally, I used GitHub from start to finish in the process of developing this application, in my opinion, GitHub is the most powerful tool a developer can use, it is the best way to host and store files dynamically, not only can an entire team with hundreds of developers work on one project at one time, you can view data such as how many commits are made, how many insertions and deletions are made, and more importantly, you have the ability to look at a previous versions/commits source code, so if you go wrong, you can view the working version, and revert the code that breaks, if cannot find a suitable fix for any issues encountered. On top of all this, you can have a discussion forum on your code, where users can submit issues or other developers may give you suggestions, advice, and fixes for any issues you may have.

Even better, you can use GitHub as a portfolio of your work, you can show off projects, assignments, and anything else to potential employers, where you can show your progression of work throughout a period of time, for example, I will be using Fuelinator on my CV to show off the work I completed over a span of a few months, hoping to impress them and get a job.

It's as simple as this if you are a developer, you NEED GitHub.

I also used JIRA to help keep on track with what I was doing, It helped me lay out tasks I needed to do, whilst also letting myself put a time on when everything should be done by, as in task to task, when they needed to be completed, As it was not my first time using JIRA, I liked it, but I remember having to spend a lot of time in my spare time when first using JIRA to understand it and get benefits out of it, whilst I find it handy and know a lot of development teams use JIRA, I personally don't enjoy using it too much, but can easily see why teams use it.

As mentioned above I messed around with MapBox and Twilio, I also briefly looked at Algolia, I actually really liked Algolia, the user interface seemed more bright and vibrant than the likes of MongoDB, and far easier to navigate through but felt it would be better used for sorting different data, larger data for usage in something like a store locator capable of handling online orders or something in a similar vein.

MapBox I just felt put me through far more hoops and hurdles to access the same sort of output Google Maps offered, but worse, I mean sure, it is better for building objects as you can do this yourself from your inputs, compared to a predefined API and package, but I was mostly confused using this service.

Twilio was going to be used to host my react app, but was not sure if I needed this, so ultimately didn't try it out enough to get my app deployed.

Chapter 4

System Design

Looking at the System Design, we will start at Fuelinator, the main code. It can be broken down into the following sections:

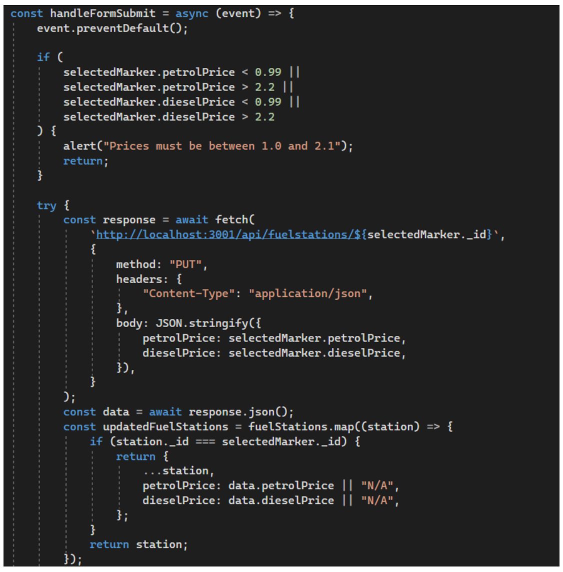
1) State Management: The component uses React's built-in state management system through the useState hook to manage various state variables such as "google", "fuelStations", "selectedMarker", "selectedCounty", and "editMode". These state variables are used to keep track of the state of the application, such as the Google Maps instance, the fuel station data, the currently selected marker on the map, the selected county for filtering, and the editing mode of the fuel station data.

2) Side Effects: The component uses React's useEffect hook to handle side effects, such as fetching data from APIs and updating the state based on changes in the component's props or state variables. For example, it fetches fuel station data from an API, updates the selected marker based on the selected county, and updates the fuel station data when the form is submitted.

3) Google Maps API: The component uses the "google-maps-react" library to interact with the Google Maps API. It renders a map component with markers representing fuel stations on the map. It also renders an info window with a form for editing the fuel station data when a marker is clicked on. The "google-maps-react" library provides an abstraction layer for working with the Google Maps API in a React application.

4) Form Handling: The component includes form handling logic for editing fuel station data. It includes event handlers for handling changes in the petrol and diesel prices, form submission, and validation of input values. When the form is submitted, it sends a PUT request to update the fuel station data on the server.

5) Error Handling: The component includes error handling logic for handling errors that may occur during data fetching and updating. It uses try-catch blocks to catch and handle errors, such as displaying error messages in the console or showing an alert to the user.

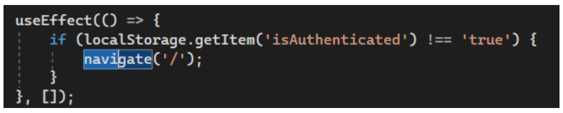


```javascript
const handleFormSubmit = async (event) => {
    event.preventDefault();

    if (
        selectedMarker.petrolPrice < 0.99 ||
        selectedMarker.petrolPrice > 2.2 ||
        selectedMarker.dieselPrice < 0.99 ||
        selectedMarker.dieselPrice > 2.2
    ) {
        alert("Prices must be between 1.0 and 2.1");
        return;
    }

    try {
        const response = await fetch(
            `http://localhost:3001/api/fuelstations/${selectedMarker._id}`,
            {
                method: "PUT",
                headers: {
                    "Content-Type": "application/json",
                },
                body: JSON.stringify({
                    petrolPrice: selectedMarker.petrolPrice,
                    dieselPrice: selectedMarker.dieselPrice,
                }),
            }
        );
        const data = await response.json();
        const updatedFuelStations = fuelStations.map((station) => {
            if (station._id === selectedMarker._id) {
                return {
                    ...station,
                    petrolPrice: data.petrolPrice || "N/A",
                    dieselPrice: data.dieselPrice || "N/A",
                };
            }
            return station;
        });
```

6) Navigation: The component uses the "react-router-dom" library to handle navigation. It uses the "useNavigate" hook to navigate to different pages or routes in the application based on user interactions, such as when a form is submitted or when a marker is clicked on.



```javascript
useEffect(() => {
    if (localStorage.getItem('isAuthenticated') !== 'true') {
        navigate('/');
    }
}, []);
```

Next, I'll Look at the design of the Server, written in Node.js

Client: A client sends requests to the server and receives responses. In this case, the client could be a web or mobile app that displays fuel station data.

Server: The server handles client requests and responds with appropriate data. It connects to the MongoDB database using Mongoose and uses Express to define API endpoints.

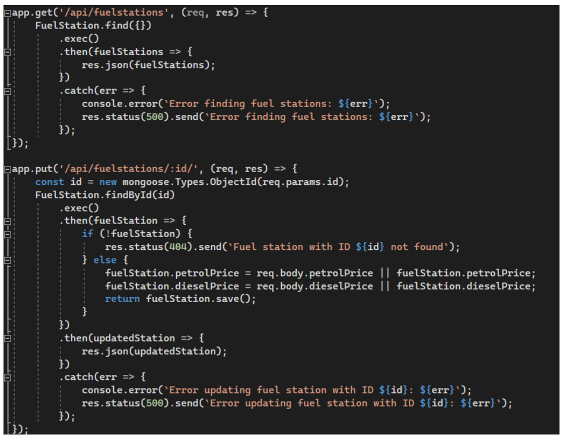
MongoDB: The MongoDB database stores fuel station data in a collection called "FuelStation". The data is represented using a Mongoose schema.

API endpoints: The server provides API endpoints for retrieving, updating, and adding fuel station data. The endpoints are:

GET /api/fuelstations: Retrieves all fuel station data from the MongoDB database. GET /api/fuelstations/:id: Retrieves a single fuel station by its ID. PUT /api/fuelstations/:id: Updates a single fuel station's petrolPrice and dieselPrice fields by its ID. GET /api/key: Retrieves the Google Maps API key.

Here's a more detailed breakdown of how the system works:

The client sends a request to the server. The server receives the request and



```
app.get('/api/fuelstations', (req, res) => {
    FuelStation.find({})
        .exec()
        .then(fuelStations => {
            res.json(fuelStations);
        })
        .catch(err => {
            console.error(`Error finding fuel stations: ${err}`);
            res.status(500).send(`Error finding fuel stations: ${err}`);
        });
});

app.put('/api/fuelstations/:id/', (req, res) => {
    const id = new mongoose.Types.ObjectId(req.params.id);
    FuelStation.findById(id)
        .exec()
        .then(fuelStation => {
            if (!fuelStation) {
                res.status(404).send(`Fuel station with ID ${id} not found`);
            } else {
                fuelStation.petrolPrice = req.body.petrolPrice || fuelStation.petrolPrice;
                fuelStation.dieselPrice = req.body.dieselPrice || fuelStation.dieselPrice;
                return fuelStation.save();
            }
        })
        .then(updatedStation => {
            res.json(updatedStation);
        })
        .catch(err => {
            console.error(`Error updating fuel station with ID ${id}: ${err}`);
            res.status(500).send(`Error updating fuel station with ID ${id}: ${err}`);
        });
});
```

determines which API endpoint to use. The server interacts with the MongoDB database using Mongoose to retrieve, update, or add data as necessary. The server responds to the client with the appropriate data. The client receives the response and updates its UI to display the data.

Now I'll take a look at the Home.js which was the implementation for a Firebase user Authentication.

The Home component initializes several state variables using the useState hook, which include loginEmail, loginPassword, signupEmail, signupPassword, signupPasswordError, errorMessage, and showLogin. These variables hold the values of input fields on the login and registration forms, error messages, and flags toggle between the two forms.

The component defines three functions, handleLogin, handleSignup, and handleToggle. The handleLogin function is called when the login form is submitted, it validates user input and calls the loginUser function to authenticate the user. If the user is authenticated, it sets a flag in local storage and navigates the user to the "/fuelinator" route. If there is an error during authentication, it sets an error message.

The handleSignup function is called when the registration form is submitted, it validates user input and calls the createUser and loginUser functions to register and authenticate the user respectively. If the user is authenticated, it sets a flag in local storage and navigates the user to the "/fuelinator" route. If there is an error during registration, it logs the error to the console.

The handleToggle function is called when the user clicks the "Register Here" or "Login Here" button to toggle between the login and registration forms.

The Home component renders the login and registration forms based on the value of showLogin variable. If showLogin is true, it renders the login form, otherwise, it renders the registration form. The rendered form includes input fields for email and password, a button to submit the form, and a button to toggle between the two forms. The component also renders an error message if there is an error during authentication or registration.

Chapter 5

System Evaluation

I feel as if I have done a relatively good job in my design of Fuelinator, however, I feel I left a lot to be desired also as I mentioned in the introduction, I could have really used more features in order to make this a much higher standard of a project, little things such as an AutoComplete search function to search by locations like towns in Ireland, I had actually attempted to implement this system but I was struggling to get it working properly with my code and ultimately decided It was better to have a fully working system than a broken system with a half working implementation. I also realized the user really needs a way to reset their password, this is just a major necessity in modern applications, why should the user have to re-register to use the application? I was able to find tutorials for Firebase, but for some reason, it didn't send the reset link correctly.

Another Issue I have is with the user updating the prices of the cheapest found stations in Ireland, when no county is selected, technically the county is null, so when attempting to update fuel prices, it throws an error, in relation to petrolPrice being null, which I was unable to fix, due to the fact the station is retrieved, with the correct data, but updating the prices through a null county breaks the code, similarly if I pick a county, update the price and head back to select county, it breaks the application unless I press cancel the edit, I would need this fixed before even thinking of adding new features.

I also feel when the user updates a price of a station, it is kind of an annoying feature that if it is no longer the cheapest price, for example, If I update the diesel price, using the find cheapest diesel price, once I save the new prices, the form refreshes back to cheapest petrol price, I should really have left that station's data on the screen, regardless of updating until the user either clicked any marker or selected a find cheapest price button again.

Going back to features to add; I wish I had used something like a timestamp extracted from the MongoDB data for each station object, highlighting when the prices were last updated, so the user can proceed with caution, and I'd want to inform the user, after 24 hours of no update, the fuel price is possibly incorrect, and offer a way to handle reviews, such as users voting submitted prices, helping other users see if the price was accurate when they visited, of course, this system isn't going to be flawless as a user can just down vote every station in Ireland if they so wished, and once the prices were updated the user reviews submitted back to 0 up votes and 0 down votes.

Lastly, I would have loved to use is location-awareness, where the user consents to sharing their location with the app, so when they click a station they can get directions from their location to the station, this would have been a fantastic feature, and really bring the app to life.

Chapter 6

Conclusion

The context of the project was for the development of an application called Fuelinator that allowed users to find the cheapest fuel prices in Ireland, even allowing them to select the fuel type. The main objective of the project was to develop a fully functional application that provided users with an opportunity to provide accurate and up-to-date fuel prices by constantly updating prices, as well as offering the users a user-friendly interface for them to interact, I feel I hit everything I set out to achieve and the context was met, even if it did have its flaws which I outlined in the System Evaluation.

The System Evaluation chapter discussed some of the limitations and issues with the application that I encountered during the development of the project. Some of my findings included the need for additional features such as a fully functioning AutoComplete search function to search for more specific locations, a password reset function to improve user authentication, and location awareness implementation in order to provide users with directions to the fuel station they wish to go to, with hopes of saving money based on the price shown for that station.

Additionally, I highlighted some of the big issues with the application's current functionality, such as the inability to correctly update fuel prices when no county is selected, and the visually annoying effect of the application refreshing to the cheapest petrol price after a user updates the price of a station, even if they updated the diesel price, instead of just leaving the current stations data on the screen, as in the station found when first searched by cheapest.

I suppose I learned that doing the work for a project like this, should be met head-on, and not when felt like it, or not in big chunks at a time, work like this is usually done in teams, and at a set pace, I found whilst I struggled with some of the features above, another developer on the team might have seen a solution or at the very least a workaround.

.1 Appendices/GitHub Repository

You can find the code on GitHub at `https://github.com/CianHession/Fuelinator`.

Bibliography

[1] React. https://legacy.reactjs.org/docs/getting-started.html.

[2] Google maps react package. https://www.npmjs.com/package/google-maps-react.

[3] Google maps react github repository. https://github.com/google-map-react/google-map-react.

[4] Node.js documentation. https://nodejs.org/en/docs.

[5] Express.js routing guide. https://expressjs.com/en/guide/routing.html.

[6] Firebase documentation. https://firebase.google.com/docs.

[7] React store locator package. https://www.npmjs.com/package/react-store-locator.

[8] Building a store locator in react using algolia, mapbox, and twilio: Part 1. https://www.algolia.com/blog/engineering/building-a-store-locator-in-react-using-algolia-mapbox-and-twilio-part-1/.

[9] Firebase authentication documentation. https://firebase.google.com/docs/auth.

[10] Google maps platform documentation. https://developers.google.com/maps/documentation.

[11] Mongodb documentation. https://www.mongodb.com/docs/.

[12] React store locator github repository. https://github.com/gocrisp/react-store-locator.

[13] Pumps.ie api. https://pumps.ie/api/.

[14] Twilio documentation. https://www.twilio.com/docs.

[15] Mapbox documentation. `https://docs.mapbox.com/`.

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15]