



Ponc

**Tomi Ilori
C16359881**

**Submitted in partial fulfillment of requirements for the degree of B.Sc. in Business Computing
Technological University Dublin
Year 4**

Supervised by - Thoa Pham

May 2020

Declaration

This work is the original all references and assistance are acknowledged

Signed : Tomi Ilori

Date : 1/05/2020

Table Of Contents

Acknowledgements	5
Abstract	7
Introduction	8
Project Introduction	8
Project Objective	8
Business Case	9
Business Rules	9
Requirements	10
Business Actors	10
User	10
Functional Requirements	11
Use Case Diagrams	11
User	11
User Requirements	12
System Requirements	13
Non-functional Requirements	15
Recoverability	15
Scalability	15
Performance	15
Interface Requirements	15
Response Time	15
Ease of Use	15
Dynamic Content Rendering	15
Design	16
System Architecture	16
Technology Stack	17
Server Side	17
Web Application	17
Technologies Used	18
Server	18
Frontend	18
Backend	19
Database	19
Other Technologies Used	20
IDE	20
Package Manager	20
JavaScript	21
Version Control	21

Server Side Technologies	21
Client Side Technologies	22
List Of Third Party Libraries Used	23
Data Model	24
Entity Relationship Diagram	24
Models	25
User	25
Entry	25
Event	26
Workout	26
FitnessData	26
EmailToken	27
Exercise	27
Software Design	28
Server	28
Routes	29
React	30
Component Structure	30
Navigation	31
Component Folders	31
Implementation	32
Non CRUD Functionality	32
Sending Emails	32
Email Verification	34
Calorie Calculation	35
Google Calendar Synchronization	36
SMS Reminders	37
Sentiment Analysis	38
Article Retrieval Using Google Search	39
Web Scraping	41
Machine Learning	42
General Implementation	43
Implementation Issues	44
User Manual	46
Registration	46
Login	47
Account Settings	47
Cancer Selection	48
Dashboard	49

Fitness Tracker	49
Diary	51
Emotional Awareness	53
Calendar	54
Understanding Your Cancer Type	57
Logout	58
Project Management	58
Test Plan	59
Conclusion	64
References	65

Acknowledgements

I would like to give a massive thanks to everyone who aided me during the process of completing this project and throughout the four years in college.

Firstly I would like to give thanks to Thoa Pham, I will admit that I was not the most focused on working on the project throughout the year and focused more on the modules and assignments that were due at closer dates, however he never gave up on me and continued to push me to work harder, her continuous advice and constant monitoring of my project is what allowed me to complete it in the time that I did. I would also like to thank Neil O'Connor who during my third checkpoint gave me the harsh reality of where I was at with my project and that shook me into making sure that I worked as hard as I possibly could.

Secondly I would like to thank Ross Nelson, one of my former work colleagues, he is currently studying in UCD and aims to cure cancer, it was him who set me on the idea to tackle cancer for my application. He provided me with valuable information and explained to me what was required in the world for cancer patients, I can not thank him enough for helping me make that tiny idea a full project.

Thirdly I would like to thank these three lecturers, Jenny Munnely, Farah Higgins and Catherine Higgins. These three women are amongst the most respectful, intelligent and devoted people I have ever met. They had always tested us respectfully to our abilities and for myself upon completing the work they had assigned us, gave me confidence that I was learning skills that I could keep for a lifetime.

I would also like to thank Don Ryan, his method of teaching, though harsh and very blunt made me realise how to really learn to program, if not for him I would have never devoted myself during my internship to self-learn the skills that allowed me to complete this project.

Finally I would like to thank my family and my girlfriend for keeping my head level for the final 2 months of the course and reminding me that I had to prevail no matter what. They were a strong foothold for me and allowed me to see reason when I felt there was nothing going right.

Abstract

Cancer is a life-changing disease that affects millions of people across the world, though it is heavily researched and documented, the ease of access and understandability to the general public and to those who suffer from cancer is not something that is provided with enough clarity to have a full grasp of it. Even though there are websites that have amazing content for people who suffer from this disease to read, there is also no interactivity. The aim of my project was to create a single website application that would provide cancer patients with both information and the interactivity to help fight their individual battles with cancer.

This project is available for use through web application and in this report I will demonstrate features and how designed and implemented each of these features. I will discuss the technologies used and the difficulties that I faced in the process of completing the project.

I also wanted to complete this application using mainly javascript. The possibility of using one language to go from database to website captivated me and was the main driver for me using the technologies that I did.

The name “Ponc” came abbreviated from the term “Precision Oncology”, this is the profiling of tumors by identifying the genes that cause them and providing specific, unique therapy for a user, which I modelled the user dashboard after, to be unique to the user. “Ponc” in gaeilge is a dot which could be used as a full stop, which is led to the name as a metaphor for stopping cancer. The crab is the astrological sign for cancer which is why I picked it as my mascot.

Introduction

Project Introduction

The aim of this project was to provide a single website which would allow individuals who suffer from cancer to have a website that would provide all the resources that they would need to aid them during the time that they suffer from the disease. I wanted to create an application that not only gave useful information but also gave people a chance to interact with it making it a dynamic web application.

The basis of the application was to create a dashboard that would give users the chance to be able to enact many of their everyday activities and give them better understanding of the things that could help them become healthier.

Originally I was very confident in this project and how I would be able to develop it, I learned very quickly that cancer is a very vast and still heavily misunderstood topic and there is very little concrete correlations that can be made between any individual type of cancer and how it came to be. The start was rough as i was shut down time after time as much of the data I wanted to use was either of no use, too complicated to understand or protected by government bodies to be used in my application so I settled to take what data I could access and moulded it into my project to fit accordingly.

Project Objective

My project turned into more of a wellness application for those who suffer from cancer. The application would revolve around a dashboard that would provide the following features for a user.

1. Register and login to the application
2. Monitor their physical activity
3. Keep a journal of their thoughts
4. Read articles based on their perceived mood
5. Track events in their life through a calendar
6. Provide understanding of the genes that have been mutated.

I wanted to create an application that would provide relief for cancer patients as it would be the one stop destination for everything that they need. While there are many websites that have valuable information they redirect individuals to other applications to maintain other aspects of their lives. Why separate everything when it can be all in one application?

Business Case

Today we are connected by the internet. This has made us more connected than we could ever imagine, there is a multitude of information on the internet about various topics. Cancer is one of these topics, cancer is so heavily researched that I got headaches trying to find relative and easily understandable information about it.

After conversing with a close friend of mine, I noticed that all these websites give out massive amounts of information about many different types of cancer, how they occur and what helps and what does not help people who suffer from cancer, the information is a lot trust me on that one.

I decided to tackle cancer in a way that said “practice what you preach”, I felt that there was no point in an individual reading multitudes of articles, reports and blogs that may give them information that they take in and forget the next day or have to navigate through the internet to find a piece of information that they found useful or somewhat interesting, but never act upon it’s like reading how to programming without ever writing code? Highly impractical.

This application would give users the chance to be able to act upon the information that they find out and track their journey as they continue to improve their health

Business Rules

1. Users must register and verify their account through email.
2. Users must be logged into the application to use it.
3. Users must select their cancer type after their initial login in order to access the dashboard
4. Users can only have one account associated with their email
5. Users must have access to the internet to use the application
6. Users must access the application through desktop as it is not responsively designed

Requirements

Business Actors

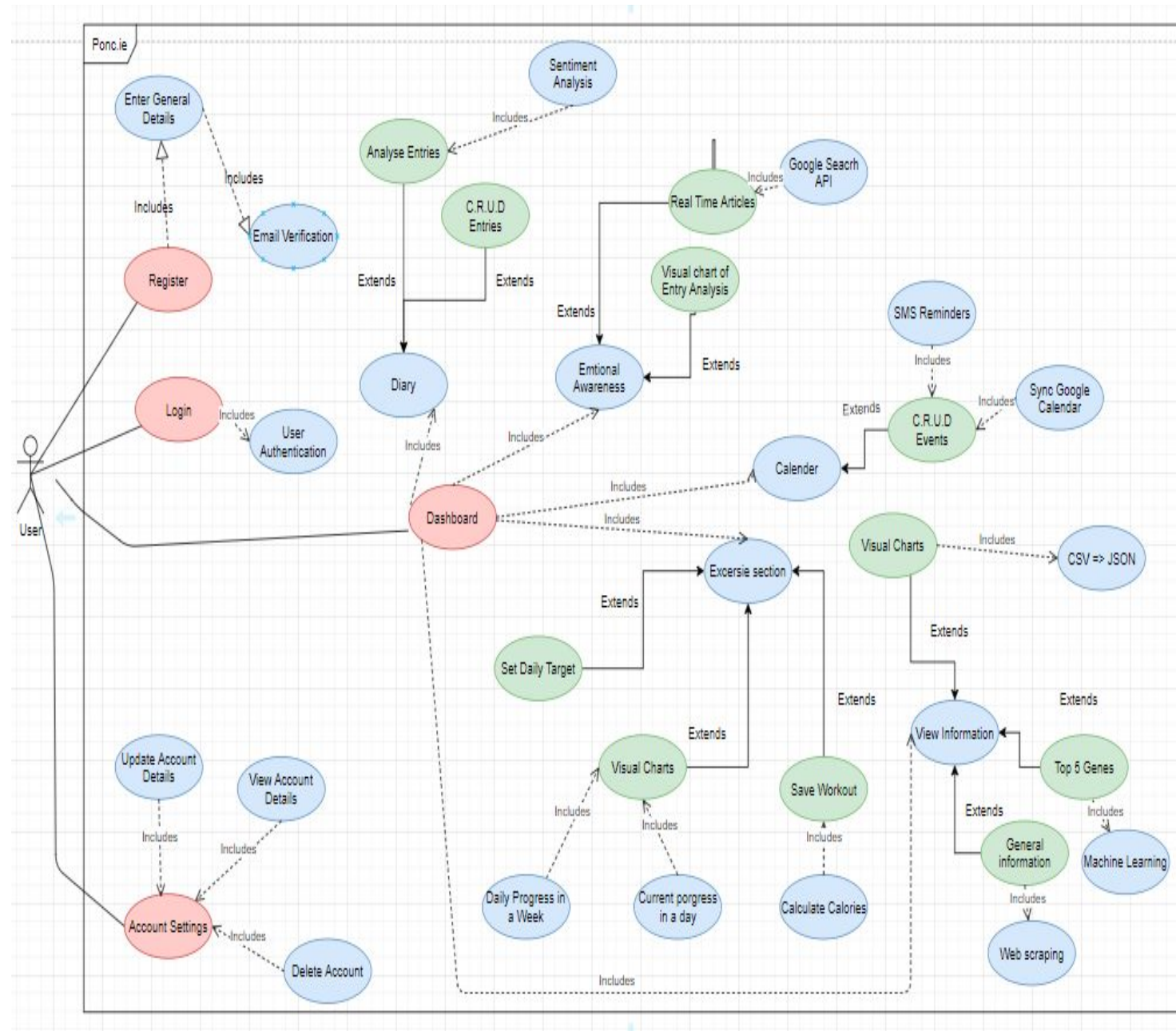
User

Everyone who registers to this application becomes a user. Every user can enact the same functionality. Once they have entered their personal details along with a password and an email their account is verified through a confirmation email sent to the user's personal email. Once registered they are allowed to view and edit their details in the profile section of the application , however before they can access the dashboard they must select a cancer type and once this is selected they have full access to the application.

Functional Requirements

Use Case Diagrams

User



User Requirements

Function	Requirement	Justification
User Register	User must create an account by inputting their details	Users need to have an account to be able to log into the application
Email Verification	User must verify their account by confirmation email	Commonly used a deterrent for people making spam accounts
Login/Logout	Users must login into the application with their email and password and be able to logout	Users must be able to log into the application in order to access its features and also log out when they want to leave it
Update Account Details	User can change personal details	If the user wants to change their personal details they should be able to do so
Delete Account	Users can delete their account	Allows user to delete their account if they do not want to use the application anymore
View Account Details	Users can view their current personal details	Users should be able to view their current details saved to the application
Pick Cancer Type	Users can select the type of cancer they are affected by	This allows the dashboard to be created.
Create Diary Entries	Users can store diary entries to the database	Used to analyse the overall mood of the user and allows the user to write their thoughts down
View Diary Entries	Users can access a list of their diary entries	Users should be able to view all the entries that they have made
Edit Diary Entries	User can edit the content their diary entries	Users should be able to add or remove or simply edit the contents of a diary entry
Delete Diary Entry	Users can delete their user entries	Users should be able to delete any diary entry they no longer wish to keep
Set Daily Calorie Target	Users can set a daily target for the amount of calories want to burn	Users should be able to set a target for themselves that they can track

Save Workout	Users can save a workout/activity they did	Users should be able to save the exercise that they did in a day.
Create Events	Users can create and record events in the calendar	Users should be able record important events in their lives
View Events	Users can view their events on the calendar	Users should be able to view their life events on the calendar
Edit Events	Users can edit the events already present in the calendar	Users should be able to make changes to these events
Delete Events	Users can delete the events on the calendar	Users should be able to delete events that they no longer require
Link Google Calendar	Users can sync their calendar in app with their google calendar to make things easier for their	If a user already has a google account then it would be beneficial to allow the user to sync the calendars

System Requirements

Function	Requirement	Justification
Establish User As Verified	System can determine if a user is verified	Users should be verified before they can login into the system
Send Verification Email	System can send email to verify user account	Users must receive the email that will allow them to verify their account
Graphically Display Weekly Progress of Calories Burned	System displays a chart of the users daily progress in terms if calories burned	Will help users visually see the results of what they are doing daily
Graphically Display Daily Progress of Calories Burned	System displays a chart of the users progress in terms if calories burned throughout the day	Will help users see their progress they are making in a day
Calculate Calories burned for a workout	System can calculate the amount of calories burned in an exercise	Users will ned to be able to record the amount of calories they are buring in a workout

Get Current Week Of The Year	System can obtain the current week of the year	Users can see what week in the year they are in
Analyse Diary Entries	System analyses diary entries and gets sentiment from them	This is used to generate an overall sentiment for users
Graphically Display Analysis Of Diary Entries	System displays a chart of the analysis of the diary entries the user has entered	Will visually help the users see the data that the system is recording from analysing their diary entries
Send SMS Reminders For User Events	When an event is created an hour before the event an SMS notification is sent	Will help remind the users of events that they recorded on the calendar
Use Google Search To Provide Articles Based off of Analysis of Diary Entries	System is able to determine a users overall sentiment and retrieves articles based on this sentiment	Will provide articles for users to inform or maintain their current mental health.
Scrape Web To Provide Information About Cancer Type	System can scrape web pages to gather information on a type of cancer	This is used to provide a general understanding of their cancer type
Provide 5 Most Common Driver Genes In A Type Of Cancer	System is trained to recognise a type of cancer and then the 5 most common driver genes affiliated with this cancer	Provides the 5 most common genes for a users cancer type
Graphically Display Top 10 Driver Genes For A Cancer Type	System obtains the top 10 driver genes for a cancer type and displays them in a chart	Allows the user to visually see how much each gene is present in cancer samples
List All Genes Present In Cancer Type	System can list all the genes mutated in a type of cancer	Provides a list of all cancer types for further information

Non-functional Requirements

Recoverability

All the data in this application is stored in a cloud database , if a user loses connection to the internet their data will still be available upon the next time they use the login and gain connection again. As the data the application uses is stored in the cloud the user does not need to worry about losing data on their device if it were to crash or break, the data is always accessible as long as there is internet connection

Scalability

This application has the potential to grow into one used by millions of individuals so i wanted to provide the ability to scale with increase of user usage. I use MongoDB Atlas which allows the database to scale. It is done by “automatic sharding” which distributes data across replica sets called shards this combined with automatic balancing ensures that data is equally distributed across shards as data volumes grow

Performance

In order for the app to be successful, it needs to work in real time. Customers need to have access to up to date information or else there is no point. Updates must be made to the app every time a change in the application is made. In order for the app to run efficiently, requests to the database and other API's must be immediate.

Interface Requirements

Response Time

This application provides users with real time information when they are accessing the database or any API's that require the application to function properly, The application response time is almost rapid with the longest recorded request being timed to 2.5 seconds.

Ease of Use

I wanted the overall experience of the application to be of a good standard to do this, I mixed multiple third party component libraries in order to improve the look and feel of the application. Finding the right one was tough but I came to use three which benefited me greatly.

Dynamic Content Rendering

I wanted the content in the application that is being rendered to be as dynamic as possible, I wanted it to be able to change with the user as they make changes in the application and also allow the system to react to these changes being made by the user and display them appropriately.

Design

System Architecture

This application was created using the “MERN” stack, this is a web development framework used by developers to create full stack web applications it consists of the following components

- **MongoDB Atlas:** A document-oriented, No-SQL database used to store the application data in the cloud.
- **NodeJS:** The JavaScript runtime environment. It is used to run JavaScript on a machine rather than in a browser, Node.js brings JavaScript to the server.
- **ExpressJS:** A framework layered on top of NodeJS, used to build the backend of a site using NodeJS functions and structures. Since NodeJS was not developed to make websites but rather run JavaScript on a machine, ExpressJS was developed.
- **ReactJS:** A library created by Facebook. It is used to build UI components that create the user interface of the single page web application.

Users interact with ReactJS at the applications front end residing in the browser. This front end is served by the application backend in a server that is created through ExpressJS running on top of NodeJS.

Any interaction that requires requests to be made that change the data in the application is sent to the NodeJS based Express server which makes calls onto the cloud database and returns the correct data which causes changes to the front end instantaneously.

I chose this framework as it allowed me to quickly develop this application and is very flexible, it also supports the MVC architecture which essentially created a smooth process for development that I could follow. Using this framework allowed me to develop the whole application in javascript which I felt made things a lot easier as I did not have to maneuver through different languages . There are plenty of third party plugins that can be used with this stack in order to create web applications , for example “ReactStrap” which make use of the “Bootstrap” CSS library to create reusable React Components. This framework allowed the application to be very minimal in memory size as it was only 594 megabytes on completion

Technology Stack



Server - Express/NodeJS

Frontend - React

Backend - NodeJS/MongooseJS

Database - MongoDB Atlas

The application has multiple layers that allowed it to function

Server Side

1. **Express Server** - Acts as a service layer where all the necessary routes that are required for the application are made accessible this is where connection is made to the database and the server is instantiated
2. **Routes** - As my application has multiple HTTP requests I split these requests into separate route files that would handle any request made from the user this acts as my controller layer.
3. **Models** - Similarly to the DAO layer in Spring boot applications each object in the application is given a model/schema using a technology called mongoose to give structure to a No-SQL database and allow them to be manipulated in the files that handle HTTP requests.

Web Application

React - React allowed me to make calls to both the backend of my application but also to any API that I needed to access, this is possible as React uses what is known as "Components", these are reusable independent pieces of a web page that can act alone or in collaboration to the rest of the application each with its own methods and internal state. This was useful as it allowed me to write my webpages almost like I was writing backend code.

Technologies Used

Server

ExpressJS



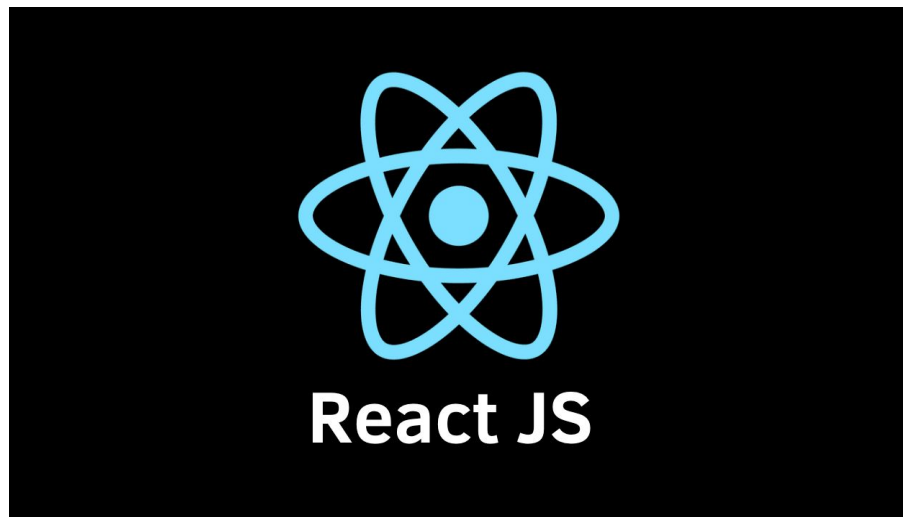
ExpressJs is a web application framework that allows you to create robust APIs and web servers to build websites or web applications with backend functionality. It is an open source lightweight package that does not conceal the main features of NodeJS. I used this

create my server and also handle HTTP requests made from the client

Frontend

ReactJS

React is a javascript library made by Facebook , it is used to create interactive UIs and design web pages for each state of the application. I chose to use react because of the rendering feature it possesses which will update and render just the right parts of a web page upon data change. It also uses encapsulated components that manage their own state which when combined make dynamic web pages..It also allowed me to use a host of third party libraries for my application



Backend

NodeJS



Node JS is an open source, cross-platform JavaScript run-time environment that executes JavaScript code server-side. Node.js lets developers use JavaScript for server-side scripting. I used NodeJS along with Express to create my server and handle HTTP requests. I also used Node in order to create models for each object in my application and interact with my database via third party plugins such as MongooseJS. I chose Node because it allowed me to run javascript on my local machine meaning that i could test functions on the backend and if they worked

successfully replicate the code on the client side of the application all in the one programming language. It also allowed me to use a host of third party libraries for my application

Database

MongoDB Atlas

A document-oriented, No-SQL database used to store the application data in the cloud. I chose to use this database as i wanted to explore what No-SQL really was, using this database allowed to me create objects that could be flexibly altered, meaning that i did not have to define a model in the first place or i could add

attributes to my database objects at any point of the application and have persist. This increased development time for the application has i did not need to define ever attribute that my users would need at the start of the application.



Other Technologies Used

IDE

Visual Studio Code



Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS. I chose to use Visual Studio Code because of its ability to instantly switch between programming languages, it also provides a host of features for users to use such as support for debugging, embedded Git control and GitHub, syntax highlighting, intelligent code completion, snippets, and code refactoring.

Package Manager

NPM



Probably one of the most important technologies used in the application. Node package manager is a package manager for JavaScript. It allows users to access an online database of public and private packages (also known as libraries) in their application. It also is used to manage the dependencies in an application. I used this to access multiple libraries for my application also to ensure that each were configured correctly and could work with the current versions of React and Node.

JavaScript



JavaScript is an object orientated computer language commonly used to create interactive effects within the web browser. I used javascript through my whole application from the back to the front. In the backend i use it to create my business logic for the application and in the front i use it along with jsx to dynamically render parts of my web pages and also to incorporate logic within the webpage



Version Control

Github

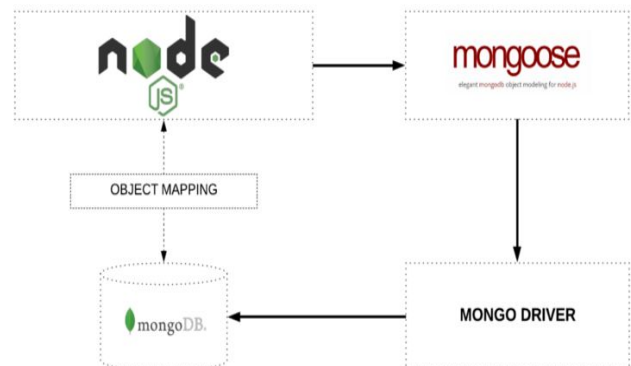
I used Github for version control of my project. Githubs version control allowed me to keep track of features I had completed and what changes were made between commits . This proved vital as my

college laptop of 4 years broke in the last month and without github I would not have been able to recover my project.

Server Side Technologies

MongooseJS

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB. I chose to use this technology as even though my database is No-SQL I am used to create databases with some sort of structure to it , this library allowed me to have this structure and also interact with the database through javascript code rather than using command prompts for the database.



Client Side Technologies

CSS

CSS (Cascading Style Sheets is a style sheet language used for styling HTML pages. I used CSS to apply styles to parts of my web page that were not accessed through React.I use many Css style sheets that come with third party libraries also to apply the default styles to components in my application.

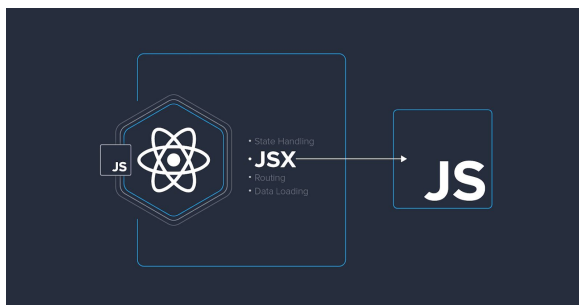


HTML



HTML is a markup language used for structuring content in a web browser. I used HTML but not in the same convention that many would, the basics of it were necessary to understand how to create web pages because I use React. It is not necessary to use HTML on its own but along with a mark-up language called jsx. I also used some parts of HTML in the project.

JSX



JSX stands for javascript xml , it allowed me to write HTML in React, it allowed m to write HTML elements in javascript and place them in the DOM without needed to use the “createElement()” method in javascript. JSX is an extension of Javascript language based in ES6 and is translated into regular javascript at runtime. One of the best features in jsx is “conditional rendering” which would allow to render elements on the web page according to the internal state of a component.It also

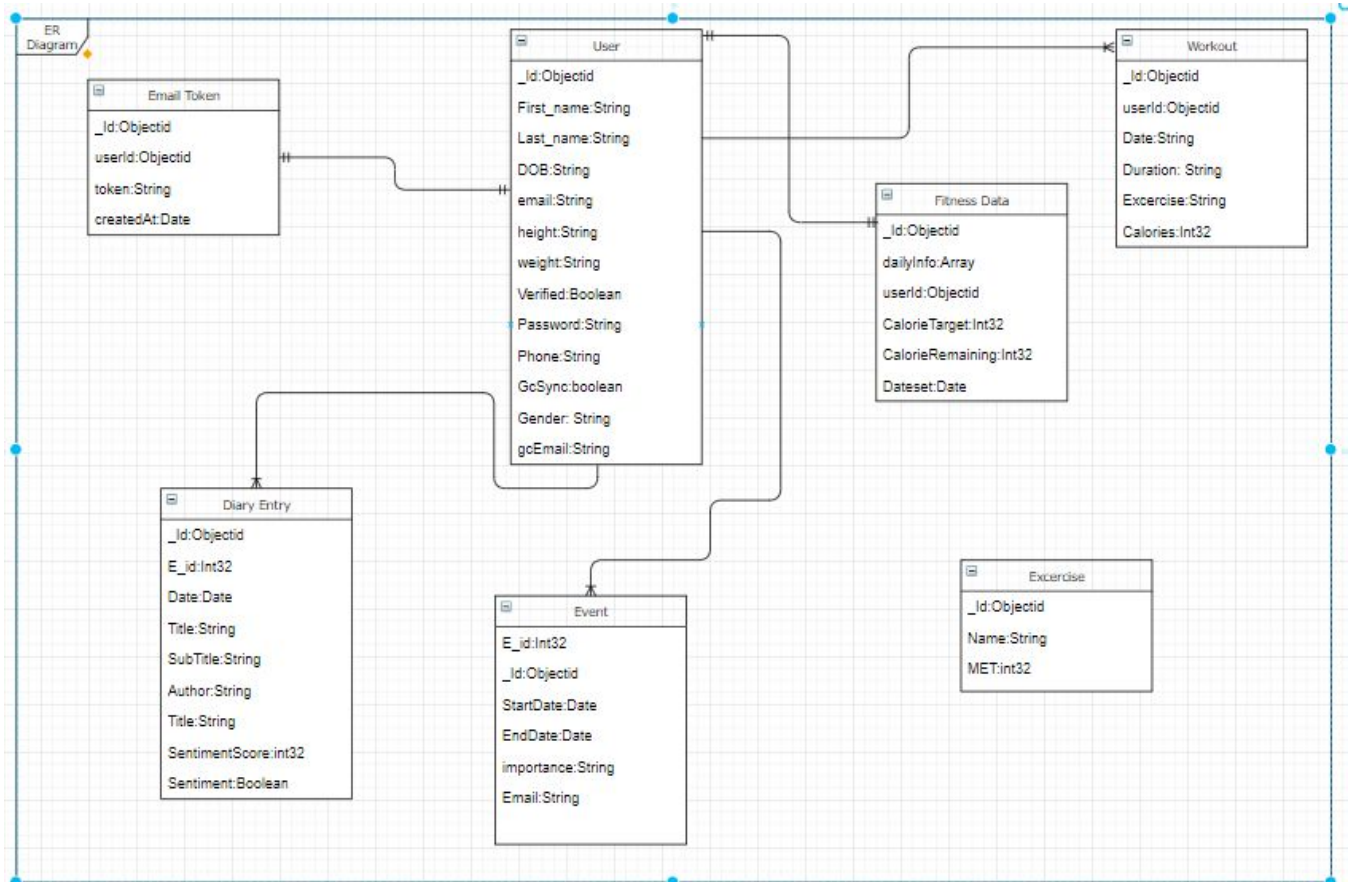
allowed me to write Javascript right beside HTML utilizing javascript methods in order to render HTML elements.

List Of Third Party Libraries Used

- Apos-to-lex-form - Used to convert contractions to lexicon e.g: I'm becomes i am
- Axios - Promise based HTTP client for the browser and node.js
- Crypto - Used to create email token
- Brain.js - Machine learning library
- Cheerio.js - implementation of core jQuery designed specifically for the server.
- Csvtojson - Turns csv to json
- Concurrently - Used to run client server and backend server at the same time
- Messagebird - Used to send sms messages
- Bootstrap - Css library for styling
- Reactstrap - React component library based off of bootstrap
- Chart.js - Library used to create charts
- Grommet -React component library
- PrimeReact - React component library
- React-d3-tree - Used for tree structure when user selects cancer type
- Framer Motion - Used for animations
- Full Calendar - Used to create calendar component in react
- Request - Used request certain web pages
- Create-react-app - Used to create React apps with no build configuration.
- Natural - General natural language facility for NodeJs
- Spelling corrector - Used to correct spelling errors
- Stop word - Used to filter out most common words in a language
- Nodemon - Used for hot reload i.e , restarting application when changes are made
- Moment - Used to parse validate and manipulate dates and times in javascript

Data Model

Entity Relationship Diagram



The user has a one to many relationship with events, diary entries & workouts.

The user has a one to one relationship with email tokens and fitness data

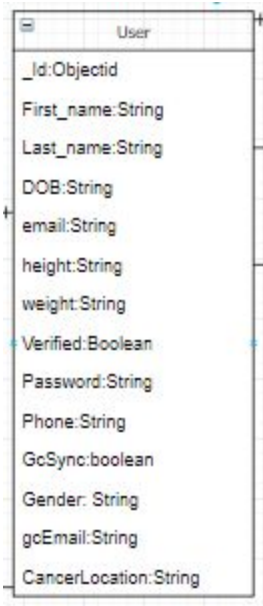
Exercise is a standalone object in the database.

Because Javascript type safety is not strict compared to Java, I was able to store most of my variables as strings even if they were numbers. When they are accessed the type is set at runtime depending on the operation being performed on the variable. The best example is the user's weight which is used to calculate the calories burned. This is stored as a string but when retrieved and put into an arithmetic operation it acts as a number.

MongoDB also allowed me to store arrays in my database object, this was used for the daily info value in the fitnessData object, i did not need to create a separate value to represent each day but could just store the values in an array with the object.

Models

User



_id :A unique identifier provided by MongoDB given to each user when they are added to the database

FirstName: String to represent the user's first name

LastName: String to represent the user's last name

Dob: String to represent the user's date of birth

email: String to represent the users email

height: String to represent the user's height

weight: String to represent the user's weight

Verified: Boolean to represent the if the user has verified their account

Password: String to represent the user's password

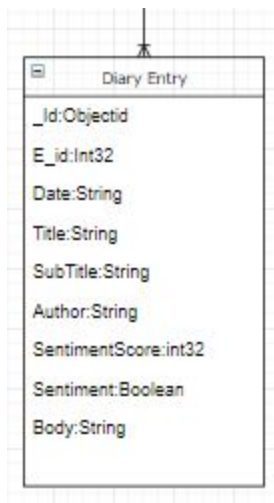
Phone: String to represent the user's phone number

Gcsync: Boolean to represent if the user has synced their google calendar

gcEmail: String to represent the user's google calendar ID

Cancer_Location: String to represent user's type of cancer

Entry



_id :A unique identifier provided by MongoDB given to each entry when they are added to the database

E_id: A number to represent the entry when being rendered

Date: String to represent when the entry was created

Title: String to represent the title of the entry

SubTitle: String to represent the subtitle of the entry

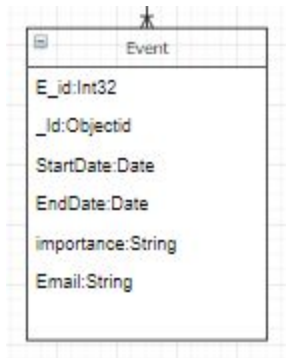
Author: String to represent the user who created the entry

SentimentScore: A number to represent the score of the entry

Sentiment: Boolean to represent how the sentiment of the entry

Body: String to represent the content of the diary entry

Event



_id :A unique identifier provided by MongoDB given to each event when they are added to the database

E_id: A number to represent the event when being rendered

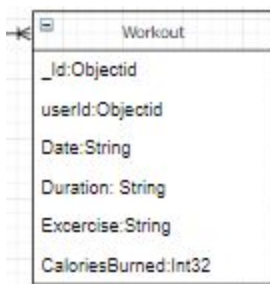
StartDate: Date to represent when the event starts

Enddate: Date to represent when the event ends

importance: String to represent the importance of the event

Email:String to represent the user who created the entry

Workout



_id :A unique identifier provided by MongoDB given to each workout when they are added to the database

userId :A unique identifier provided by MongoDB given to each user when they are added to the database , used to enforce relationship

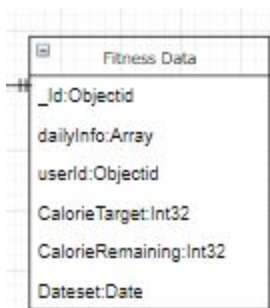
Date: String to represent when the workout was created

Duration: String to represent when the duration of the workout

Exercise: String to represent when the name of the exercise the user did in the workout.

CaloriesBurned: Number to represent the calories burned in the workout

FitnessData



_id :A unique identifier provided by MongoDB given to each fitness data object when they are added to the database

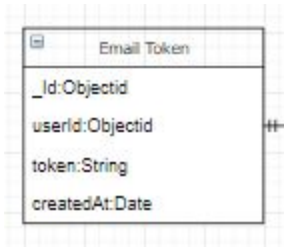
userId :A unique identifier provided by MongoDB given to each user when they are added to the database , used to enforce relationship

dailyInfo: An array of numbers to represent the users weekly progress

CaloriesTarget: Number to represent the daily calorie target

CaloriesRemaining: Number to represent the calories remaining according to how many calories have been burned from the daily target.

EmailToken



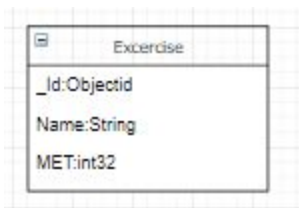
_id :A unique identifier provided by MongoDB given to each Email Token object when they are added to the database

userId :A unique identifier provided by MongoDB given to each user when they are added to the database , used to enforce relationship

Token: String used to represent the cryptic token associated with the verification link

CreatedAt: Date to represent when the token was created

Exercise



_id :A unique identifier provided by MongoDB given to each Exercise object when they are added to the database

Name: String to represent the name of the exercise

MET: A number to represent the metabolic equivalent of the exercise used to calculate the amount of calories burned in a workout

Software Design

Server

In the file "server.js" this is where I create the server , connect to the database and also establish access to the files that contain the business logic of my application

- **Creating the server**

To create the server I used the express module this module allows me to create a variable that represents an express application .I then created a a port variable and then called the ".listen()" method on the express object which returns a http.Server object on the port passed into this method

```
//Required modules
const express = require("express")
//Initialise express
const app = express()
//Body Parser Middleware
app.use(express.json())

const port = process.env.PORT || 5000

const mongoose = require("mongoose");

app.listen(port , ()=>
  console.log(`Server started on port ${port}`)
)
```

- **Connect To the database**

To connect to the database I needed to use the mongoose module. This module allowed me to create an object with all the relevant methods that i would need to connect to my database, Then i had to pass in the URL connection key obtained from MongoDB Atlas to connect my application to the cloud database.

```
const mongoose = require("mongoose");
//Database
const db = require("../config/Keys").MongoUrl;

//Connecting to database
mongoose
  .connect(db , {useNewUrlParser:true ,useCreateIndex:true , useUnifiedTopology: true})
  .then(()=> console.log("Connected To Database..."))
  .catch(err => console.log(err))
```

- **Establish business routes**

Each route file contains HTTP requests for the application to function. Each of these files were made exportable and then imported into the file where the server was being created. I then used the “.use()” method which allowed me to bind application level middleware to the Express object. Each of these files can be seen as middleware it is the middle point between the database and the front end.

```
//Routes
const Users = require("../Routes/API/users")
const Fitness = require("../Routes/API/fitness")
const Entries = require("../Routes/API/Entries")
const Events = require("../Routes/API/Events")

app.use("/Routes/API", Users);
app.use("/Routes/API", Fitness)
app.use("/Routes/API", Entries)
app.use("/Routes/API", Events)
```

The Server would then be started using nodemon which is a dependency used for hot reload , however seeing as i was going to use React as well which also uses a server i need to create my own script using concurrently that would call the scripts to run both servers at the same time. These scripts were written in my package.json file. In my terminal i would then call “npm run both” to start the application

```
"dev": "nodemon server.js",
"both": "concurrently \"npm run dev\" \"npm run client\" ",
```

Routes

There were four route files in my application , users , events , fitness and entries. Each one of these files followed the same structure:

1. Used express module
2. Used the express.Router object to handle HTTP requests from the client side of the application

```
const express = require("express")
const router = express.Router()
```

3. This object has access to each HTTP method , GET POST PUT DELETE , in my application there was usually a response sent back to the user so i stuck to just the get and post methods. This object allowed me to create methods like pieces of code that responded to a specific url from the client and executed business logic.

```
router.get("/Greeting" , (req ,res)=>
{
  res.json({msg:"it worked"})
})
```

Example of Router object in use

4. The router object was then made exportable using the module.exports function, this essentially returns an object containing all the methods that the file has.

```
module.exports = router
```

React

React uses what they call “Components” which lets you split the UI into independent, reusable pieces, conceptually they are like javascript functions. Instead of creating multiple small components for a single web page I treated each web page as its own component with its own methods and internal state which would allow me to make the pages render in distinctive ways based on the state of the component.

Component Structure

Each page of the application was treated as its own component, this meant that I could render pages according to the state that they have and also pass data into individual webpages , reducing the need to replicate code. I used ES6 class to define my components as class components. Each extended the Component class from react inside these class components was a mixture of javascript and jsx which allowed me to create the web pages.

```
import React, { Component } from 'react'

export class Personal extends Component {
```

By extending the Component class from react the Component/Web page can be made exportable. I then would export each component which would allow me to use them in other web pages accordingly

```
export default Personal
```

Navigation

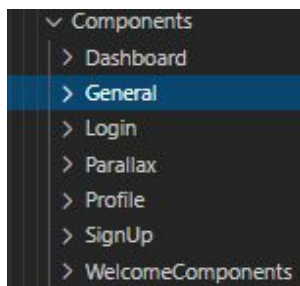
I used a react library called “react-router-dom” which allowed me to handle the routing of components. I centralised the access of these routes in one file called “index.js” this file imported every necessary component for the application to function and then by using the “Router , Route and path” tags I provided a URL path and the corresponding component that was to be rendered when this path is called. These paths were stored in a single variable that would render the appropriate component.

```
const routes = [
  <Router>
    <div>
      <Route exact path = "/" component = {Welcome}/>
      <Route path = "/why" component = {R1}/>
      <Route path = "/why/R2" component = {R2}/>
      <Route path = "/why/R3" component = {R3}/>
      <Route path = "/SignUp" component = {UserForm}/>
      <Route path = "/Login" component = {Login}/>
      <Route path = "/Profile" component = {Details}/>
      <Route path = "/Dashboard" component = {Dashboard}/>
      <Route path = "/UserDashboard" component = {UserDashboard}/>
      <Route path = "/Food" component = {FoodPage}/>
      <Route path = "/Fitness" component = {ExercisePage}/>
      <Route path = "/Diary" component = {DiaryPage}/>
      <Route path = "/Calender" component = {CalenderPage}/>
      <Route path = "/Emotion" component = {EmotionPage}/>
      <Route path = "/Research" component = {ResearchPage}/>
    </div>
  </Router>
]

ReactDOM.render(routes, document.getElementById('root'));
```

Component Folders

I created separate folders to house the components for each section of the application



Dashboard - This folder contained all the components used for the user dashboard

General - This folder contained general components that could be used throughout the application such as the navigation bar.

Login - This folder contained the components responsible for logging into the application

Parallax - This folder was meant to contain components for when the user first enters the application and is shown the reasons to join

Profile - This folder contained the components for the user's profile

SignUp - This folder contained components that handled the registration process for a user

WelcomeComponents - This folder contained the initial landing page component.

Implementation

Non CRUD Functionality

Sending Emails

I wanted this application to have the ability to verify user accounts through the use of emails, I had never had any experience sending emails in applications and this was the first non CRUD, functionality that i added to my project.

I used two libraries for this part of the project crypto for generating a unique string for my EmailToken object and nodemailer for the process of actually sending emails.

1. When a user registers their account details are saved, in the process of saving this account I make use of a javascript technique called chaining which allows me execute more then one method in quick succession after an initial method has been carried out.I then create the hex string token and the link that the user will use to verify their account.

```
newUser.save()
.then(user=>
{
  const crypt = crypto.randomBytes(16).toString('hex')
  const host = req.get('host')
  const link = `http://${host}/Routes/API/users/verify?token=${crypt}&&Email=${user.Email}`
  const fitnessdata = new FitnessData(
```

2. I then create an Email token object with the id of the new user and the hex string as parameters and then i save this object to the database

```
RegToken = new ET(
{
  userId:user.id,
  token: crypt
})
RegToken.save()
```

3. Again using chaining I begin to execute the operation of sending the email to do this I had to create a transporter object using nodemailer “.createTransport()” method , this would take in the email service sending the message , the account and the password of the sender email.


```
const transporter = mail.createTransport({
  service: "gmail",
  auth: {
    user: "tomiiloriponc@gmail.com",
    pass: "123."
  }
})
```

4. I then would create the actual email object here i would specify who the email is being sent by , the recipient , the subject matter , the actual html content which contains a button with a reference to the verification link and the template being email.

```
const mailOptions = {
  from: "tomiiloriponc@gmail.com",
  to: user.Email,
  subject: "Confirmation Email",
  html: "Hello,<br> Please Click on the link to verify your email.  
<br><a href="+link+">Click here to verify</a>",
  template: "email"
}
```

5. Finally I would use the nodemailer “.sendMail()” method passing in the email object and also a function for errors or success messages .

```
transporter.sendMail(mailOptions, function(error, info){
  if (error) {
    console.log(error);
  } else {
    console.log('Email sent: ' + info.response);
  }
});
```

tomiiloriponc@gmail.com

to me ▾

...

Hello,

Please Click on the link to verify your email.

[Click here to verify](#)

Users would then receive the following email

Email Verification

To verify the account was much easier then sending the actual email

1. The link that was sent has two parameters in the URL, token and email, these were the two values I would use to verify the user account to get these parameters. I had to extract them from the req.query object in the HTTP request.
2. I would then use the mongoose method ".findOne" on the Email Token model to find the token with the matching token string from the verification link.
3. If the token is found a message indicating that the token does not exist is shown, otherwise i then use the "findOne()" method on the User model to find the user with the corresponding email from the verification link
4. If the user is not found a message is sent indicating this else this found users "Verified" property is set to user and the user is updated a message indicating the account has been verified is returned to the user

```
//Search for Email token in V link
ET.findOne(
  {
    token:req.query.token
  }
)
.then(token=>
  {
    //Token not found
    if(!token)
    {
      return res.status(400).json({msg:`Token Not Found`})
    }
    else
    {
      //Search for User with email in V link
      User.findOne({
        Email:req.query.Email
      })
      .then(user=>
        {
          //User not found
          if(!user)
          {
            return res.status(400).json({msg:`User Not Found`})
          }
          else
          {
            //Updated user verified value to true
            user.Verified = true
            user.save()
            return res.status(400).json({msg:`Account Verfied Proceed to Login`})
          }
        }
      )
    }
  }
)
```

Calorie Calculation

When a user wants to save a workout the application is able to determine how many calories have been burned during the workout. This operation shows the use of React's class components and the state property

```
getExercices= () =>
{
  axios.get("/Routes/API//fitness/getExercices")
  .then(res =>{
    let exerciseAPI = res.data.map(exc=>
    {
      return {id:exc._id , name:exc.Name , MET:exc.MET}
    })
    this.setState({Exercices:exerciseAPI})
  })
}
```

```
router.get("/fitness/getExercices" , (req,res)=>
{
  Exercise.find()
  .then(exc =>res.json(exc));
})
```

1. The user is greeted by a dialog pane which allows them to select the duration and exercise that they completed. Each exercise is obtained from the database and loaded into the components state using the getExercices method in the front end which would make a HTTP request to the backend to retrieve all objects in the Exercise model . This would then return an array containing the exercises.

2. To calculate the calories burned I get the selected exercise from the select field. I then calculate the calories burned per hour, to do this i get the users weight that they input when

registering i then multiply this by the exercises "Metabolic equivalent" value i then get the percent of an hour that the user input, finally i multiply the calories burned per hour of doing this exercise by the percentage of an hour and round this value to the first decimal place

```
caloriesBurned =()=>
{
  //Selected Exercise
  const se = this.state.SelectedExercise
  //Get the object with the title that matches the selected exercise
  const found = this.state.Exercices.find(e => e.name === se)

  //Weight by MET gives clories per hour of doing this exercise
  const caloriesPerHour = this.state.weight * found.MET

  //Percentage of an hour
  const percentOfMet = this.state.Duration/60

  //Calories Burned calculated
  const cb = caloriesPerHour * percentOfMet

  //Rounded to first decimal place
  const num =cb.toFixed(1)
  //Workout calories burned vaalue is set to this calculated value
  this.setState({
    CaloriesBurnedWorkout:num
  })
}
```

Google Calendar Synchronization

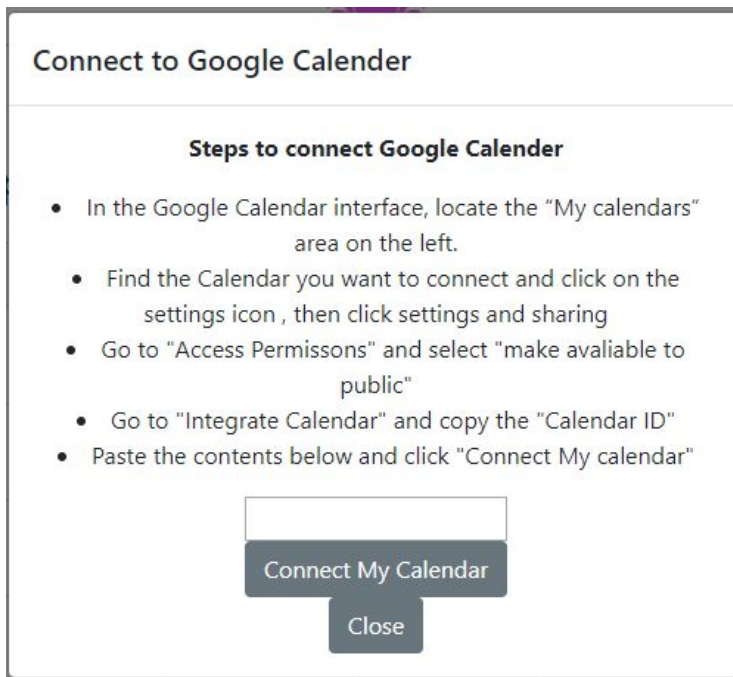
To synchronise the google calendar i made use the “fullcalendar” library for react and the Google Calendar API.

1. First I had to create a project in the Google Developers Console and obtain an API key for use with Google Calendar API

API Keys

<input type="checkbox"/>	Name	Creation date ↓	Restrictions	Key	
<input type="checkbox"/>	⚠ API key 1	Apr 20, 2020	None	AIZAaSyB44p...wVT2C_fdRw	

2. If the user wants to sync their google calendar to the calendar in the application I provide them with the steps to do so in a dialog pane.



3. Once the user pastes in the Calendar ID of the calendar they want to sync i make a HTTP request that finds the user in the database with corresponding email and then sets two new properties , one to indicate that the user has synced a google calendar and the ID of that Calendar

```
router.post("/Events/saveGC/:email" , (req,res)=>
{
  //find user
  User.findOne({Email:req.params.email})
  .then(user=>
  {
    console.log(user)
    //Add attributes to indicate user has a google account synced
    user.set("Gcsync" , true)
    user.set("gcEmail" , req.body.data.email)
    //Updated user
    user.save()
    console.log(user)
  })
})
```

4. Then using the fullcalendar library i use the Google calendar plugin, access the events in the users google calendar I pass in an array with both the events in the database and the google calendar into this calendar component in order to show both at the same time

```
//ARRAY OF EVENTS FROM DB AND GOOGLE
//Both are objetscs with an array of events
var events = [
  this.state.calendarEvents , {googleCalendarId:this.state.gcidconfirm}]

//API KEY AND PLUGINS
googleCalendarApiKey="XXXXXXXXXXXXXXXXXXXXXXXXXXXX"
plugins=[ dayGridPlugin , interaction , timeGridPlugin, googleCalendarPlugin ]
//Sources of events
eventSources={events}
```

SMS Reminders

When a user creates an event on the calendar to store in the database , I wanted to mimic the google calendar functionality of sending reminders to the user of the event that they have coming up. To do this i made use of the messagebird API/SDK for sending sms messages. I also used the momentJS library for formatting the times passed into the reminder.

This is mainly handled in the backend when an event is being saved.

1. First I find the user in the database that is currently logged in by passing the email address into the HTTP request and using this to query the database to find the user with the matching email.

```
User.findOne({Email:req.params.email})
.then(user =>
```

2. Once the user is found the details of the event are taken from the request body and the event is stored in the database

```
newEvent.Title = title
newEvent.E_id = id
newEvent.Body = content
newEvent.StartDate = start
newEvent.EndDate = end
newEvent.Importance = importance
newEvent.Email = req.params.email

newEvent.save()
```

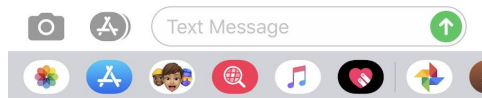
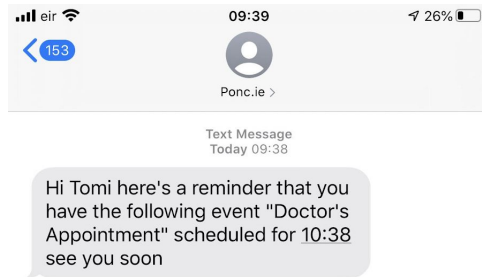
3. Using chaining after the event is stored in the database i begin to create the reminder
 - a. I noticed that the time being sent was an hour and two minutes behind because of the datetime element i was using so i had to use the moment.add() method to correct this mistake.
 - b. Secondly i set the reminder to happen an hour before the event by cloning the time and subtracting an hour off the time that event is due to start

```
var EventReminderStart = moment(req.body.StartDate).add({hours:1 ,
minutes:2})
var rem = EventReminderStart.clone().subtract({hours:1})
```


- c. Then using the messagebird library i create the message object , here i specify who is sending the message , the recipient being the user's phone number , the time that the SMS message should be sent and the body of the text.

```
messagebird.messages.create({
  originator:"Ponc.ie",
  recipients:[user.Phone],
  scheduledDatetime :rem.format(),
  body:`Hi ${user.FirstName} here's a reminder that you have the
  following event "${newEvent.Title}" scheduled for $
  {EventReminderStart.format("HH:mm")} see you soon`
})
```

The user would then receive a text reminder.



Sentiment Analysis

When a user creates a diary entry the application analyses the contents of the entry and produces a sentiment value based on this.

To do this I had to format the text using multiple libraries and then finally use a sentiment analyser to get the sentiment value of the text . These libraries included:

- AposToLexForm
- Natural
- SpellCorrecter
- Stopword

Process

1. A variable is created to represent the score of the content the user inputs , this is the result of a method called getSentiment

```
var score = getSentiment(body)
```

2. In this method I first convert any contractions to lexicons , this means text such as “I’m” becomes “I am” I then convert the text to lowercase and remove any non alphabetic characters using javascripts’ .replace method and regular expression.
3. I then imported a tokenizer what this does is that it creates an array of each word in the content from the user then i used this tokenizer on the lower case text
4. For each word I then corrected any spelling errors
5. I then removed any stop words in the text , these are commonly used words such as “ a , what ,but”
6. I then import a sentiment analyser a stemmer , a stemmer reduces any word to its root for example “giving” becomes “give”
7. I use the sentiment analyzer on the filtered text which has a dictionary of words that have a polarity score from positive to negative , this analyzer gathers the total score and returns it and then returns this value setting the sentiment score of the diary entry being saved to the database.

```
function getSentiment(body)
{
  //Convert text to lexicon
  const lexedBody = aposToLexForm(body);
  //Convert to lower case
  const casedBody = lexedBody.toLowerCase();
  //Remove non alphabetical characters
  const alphaOnlyBody = casedBody.replace(/^[a-zA-Z\s]+/g, '');

  //import tokenizer
  const { WordTokenizer } = natural;
  const tokenizer = new WordTokenizer();
  //Create a tokenized version of the content from user
  const tokenizedBody = tokenizer.tokenize(alphaOnlyBody);

  //for each word correct spelling
  tokenizedBody.forEach((word, index) => {
    tokenizedBody[index] = spellCorrector.correct(word);
  })

  //Remove stop words
  const filteredBody = SW.removeStopwords(tokenizedBody);

  //Import Sentiment analyser and Stemmer
  const { SentimentAnalyzer, PorterStemmer } = natural;
  //Create object to analyse text
  const analyzer = new SentimentAnalyzer('English', PorterStemmer, 'afinn');
  //Gets sentiment of filtered down text
  const analysis = analyzer.getSentiment(filteredBody);
  //returns value of sentiment
  return analysis
}
```

Article Retrieval Using Google Search

1. This process starts at the backend when the user enters this section of the application an object with the total number of positive , negative and neutral diary entries and also the type of sentiment with the most entries corresponding to it is retrieved by HTTP request. This is done by analysing each entry made by a certain user and then determining whether the sentiment was of the three possible values.

```
{positive: 1, negative: 0, neutral: 0, Sentiment: "positive"}
```

```

//Questions for google
const Queries=
{
  positive:"how positivity affects health recent",
  negative:"how negativity affects health recent",
  neutral:"how mental health affects health recent"
}

//question
var question =""

//Determines what question
if(this.state.ed.Sentiment === "positive")
{
  question=Queries.positive
  console.log(question)
}
if(this.state.ed.Sentiment === "negative")
{
  question=Queries.negative
  console.log(question)
} if(this.state.ed.Sentiment === "neutral")
{
  question=Queries.neutral
  console.log(question)
}

```

2. When the page renders an object with three potential questions for google is created then an empty question variable is made , depending on the sentiment retrieved from the database object the question is set to one of the three possible questions.

3. To actually use Google Search i used the “SerpStack” google search API, this was the only free option available that i could find. I first create an object with the access key provided by the website , the question set and the number of results I want.

```

//Object with access key, question and number of results
const params = {
  access_key: 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX',
  query: question,
  num:20
}

```

```

axios.get('http://api.serpstack.com/search', {params})
.then(response =>
{
  //filtered repsonse
  let apiResponse = response.data.organic_results.filter((article)=>{
    return article.snippet !== ""
  });
  //internal state set
  this.setState({articles:apiResponse})
  console.log(apiResponse)
})

```

4. I then use “Axios” which is a lightweight client for making HTTP requests in the browser, i pass in th object created and then i filter the results in order to only give me the articles with a preview so the user can have a general idea of what the article entails, finally i set the internal state to these results which are then rendered on the web page.

Web Scraping

I did not want to create an individual cancer object for each cancer type in my database, there was no API that would provide the information that I wanted to display in my application about a user's cancer type so I used web scraping to get a basic description about the cancer type a user has. To do this I used the request and cheerio library.

1. First I obtain the user's cancer type and this is sent from the webpage as a parameter in the request URL.
2. Then I use the request library to make a call to the "Irish Cancer Society" webpage affiliated to a certain cancer type which is passed in as a URL parameter
3. Cheerio is used to give developers the ability to use jQuery-like syntax in the backend; I use this to create a variable that represents the HTML of the web page found
4. Then using this variable I scrape the web page for the elements containing the appropriate information that I require.

```
router.get("/users/tumorDef/:type", (req, res) => {
  //User cancer type
  tumor = req.params.type.toLowerCase()
  //request made to Irish Cancer Society
  request(`https://www.cancer.ie/cancer-information-and-support/cancer-types/${tumor}-cancer`, (error, response, html) => {
    {
      //if there is no error
      if(!error && response.statusCode === 200)
      {
        //dollar sign variable acts as it would in JQuery allowing me to manipulate the dom
        const $ = cheerio.load(html)

        //Find h3 tags
        $("h3").each((i, el) => {
          //The first h3 element contained the information I need
          if(i === 0)
          {
            //Two pieces of information are taken from the next elements
            const def1 = $(el).next().text()
            const def2 = $(el).next().next().text()
            //content stored in an object
            const definitions = {
              def1: def1,
              def2: def2
            }
          }
          //object sent back to front end
          res.send(definitions)
        })
      }
    }
  })
})
```

5. I store these definitions in an object and send it back to the component that renders the cancer information, where they are rendered onto the webpage

Machine Learning

I wanted to include machine learning at some point in my application, however due to the complexity of cancer data and lack of clear correlation, it was very hard to make use of this technology, but i did manage to implement this feature to simulate what an expert may be able to further develop.

I used a machine learning library called brainJs to replicate what I wanted to do.

1. First I obtain the user's cancer type and this is sent from the webpage as a parameter in the request URL.
2. I then create a Long Short-Term Memory network which is a recurrent neural network that is capable of learning order dependence in sequence prediction problems. In this case I wanted it to recognise the type of cancer and return the corresponding 5 main genes that cause the type of cancer.
3. I trained the network to return a string containing the 5 most common genes in a type of cancer. Because of how direct the information is, I set it to only iterate over the results 200 times which was the shortest and most accurate value for iterations.
4. This string is then converted into an array of the 5 genes by using the tokenizer
5. This array is then sent to the React component

```
router.get("/users/getGenes/:type", (req , res)=>
{
    tumor = req.params.type

    const net = new brain.recurrent.LSTM()

    net.train([
        { input:"Breast" ,output:["TP53 " , "KMT2C " ,"PIK3CA " , "GATA3 " , "CDH1 " ]},
        { input:"Bladder" ,output:["TP53 " , "KDM6A " ,"PIK3CA " , "KMT2D " , "ARID1A " ]},
        { input:"Prostate" ,output:["TP53 " , "SPOP " ,"AR " , "FOXA1 " , "KMT2D " ]},
        { input:"Skin" ,output:["BRAF " , "NRAS " ,"FAT3 " , "CDKN2A " , "PTPRB " ]},
        { input:"Lung" ,output:["TP53 " , "KRAS " ,"KEAP1 " , "EGFR " , "STK11 " ]},
        { input:"Pancreas" ,output:["KRAS " , "TP53 " ,"SMAD4 " , "CDKN2A " , "ARID1A " ]},
    ],{iterations:200})

    const output = net.run(tumor)

    const { WordTokenizer } = natural;
    const tokenizer = new WordTokenizer();
    const genes = tokenizer.tokenize(output);

    console.log(genes)
    res.send(genes)
})
```

General Implementation

My project follows the same process for interacting with the database and the user , a simple example of this is loading the calendar events from the database. This is the same process for creating updating and deleting objects

1. Axios request is made from the React component , either usually GET or POST

```
axios.get(`/Routes/API//Events/getAll/${this.state.email}`)
```

2. This request is sent to the corresponding Node Js file which handles the request based on the URL route

```
Routes > API > JS Events.js > router.post("/Events/addEvent/:email")
```

3. The Node Js file redirects the route to the appropriate handler method
4. Business logic occurs
5. A response is sent back to React

```
router.get("/Events/getAll/:email" , (req , res)=>
{
  Event.find({Email:req.params.email})
  .then(events => res.json(events))
})
```

6. The React component gets the response and handles it by setting the data to corresponding values in its internal state

```
getAllEvents()
{
  axios.get(`/Routes/API//Events/getAll/${this.state.email}`)
  .then(res =>
  {
    let eventsdb = res.data.map((eve) =>
    {
      return this.formatEvent(eve)
    })
    console.log(eventsdb)
    this.setState({calendarEvents:eventsdb})
  })
}
```

Implementation Issues

Because my project used a technology stack that I had very little experience using I ran into a lot of problems while developing this application. Thankfully I was able to create solutions for the majority of my problems.

- **Styling and Layout** - I am generally just very bad at styling and making things look nice, I wanted my application to look good and I spent a lot of my initial time trying to use css to style my components, I was spending too much time trying to make my application look good rather than having it work, to conquer this issue i used React component libraries , these libraries have components like buttons and input fields automatically styled for you so i did not need to worry about spending time on making my components look good. I also had trouble developing the layout of my application and used CSS properties to divide up my page. This proved to be very bad in terms of responsiveness , so i used a grid layout system from one of the libraries i imported in order to divide my pages into blocks which made it easier to design.
- **Sentiment Analysis** - The original method that I used to get the sentiment score of a users diary entry was flawed , initially it referred to a dictionary of words and only gave a score if these words were present in this dictionary , it also was not able to identify negation, so if a user said "I am not happy" the method would return a positive score due to the presence of the word "happy". This is why i changed the method to use a library for the sentiment analysis part as it was more accurate and could handle negation
- **No Exercise API** - My application required a list of exercises in order for users to select when they want to log a workout, however there was no such API that was readily accessible. To combat this I created my exercise objects in my database. The only values i needed were the name of the exercise and the metabolic equivalent value which i retrieved from a website that had a store of these values.
- **Integrating Google Calendar** - Using the Google Calendar API itself was not very challenging as you just need the api key and the ID of the calendar you wish to use. The big issue for me was being able to render both the events from my database and the events from the google calendar at the same time on my calendar component. The library I used to create this calendar has a parameter called "eventSources" ' which takes in an array of different sources for events. This parameter was what I used to resolve this issue as I created a separate variable containing the array of events from my database and the array of events from google calendar.
- **Interactive Calendar** - I wanted the user to be able to drag and drop the events on the calendar and allow these changes to persist to the database, this fullcalendar library provided a plugin to make it interactive but it didn't explain how to use it. It took some time but i managed to track down what was known as the "EventAPI" object which is returned when an event is dragged and dropped on the calendar. This allowed me to access to necessary fields i needed to persist the data to the database
- **Schema Strictness** - At first i did not understand how to use a No-SQL database to create attributes for my database objects during runtime, however I researched the mongoose documentation and discovered that i had to pass in an object with the value "Strict" and set it to false

- **React Rendering** - React has a powerful feature that makes components re-render as soon as the internal state has been changed, this proved to be an issue when using animations as everytime i changed the state, charts would re-render which was not what i wanted in order to combat this i had to set the animation speed to be so fast that the human eye could not perceive that a change was being made.
- **SMS Reminders** - The data time element that i used to set the time of when events start was set to GMT+01:00, this meant that any time i input was an hour incorrect which meant that the sms messages were not being sent at the correct time. To solve this issue i used MomentJs in order to add an hour onto the time being recorded so the SMS reminder would be accurate.
- **Email Verification Email** - I was not able to style the email sent to users the way that i wanted to i tried to send separate files , but they did not allow me to pass in the verification link sent along with the email so i had to resolve to just make it as basic as possible i was still happy that it was functioning properly.
- **HTTP Calls From React** - I had huge trouble learning how to make HTTP requests from the frontend to my database initially. I believed that I could just make these calls using link elements from html, this did not work. After extensive research I began to use the Javascript Fetch API this made it easier to send requests from react to my database, halfway through my project i switched to an even more efficient way of making these requests by using the axios library.
- **Laptop Crashing** - My laptop crashed in the last month of my development of this application thankfully i had used github to track the versions of my project which allowed me to restore it to its initial state once i replaced the laptop
- **Machine Learning** - Cancer is a vast topic with a lot of information, a lot of this information does not exactly correlate and the ones that do were not the type of data i could make use of , originally i wanted to include a questionnaire that would take user information and decide determine the likely type of cancer they have, to do this would require huge amounts of data and work by experts. To combat this I used a small machine learning library and trained it to recognise the main genes that are in a type of cancer.
- **React Lifecycle & HTTP Requests** - React's lifecycle methods proved to be very useful in loading data into my application when a user went to a certain page, the initial method for when a components is created is called "componentDidMount" this allows you to perform actions as soon as the component renders. This proved to be an issue when fetching data that depended on data that also had to be fetched, the pages counted too fast meaning data from the database was not fetched before they were rendered, to combat this i used the "window.setTimeout()" method to slow some HTTP requests down in order for all the data that needed to be retrieved to be done in a synchronous manner

User Manual

Registration

To be able to use the application a user must first register to create an account. The registration form I created is a multi-step form with three parts account information , personal information and health information. The user is also able to go back to the previous step and make any changes to their details once they confirm their details an email is sent

Account Information

Email

demo@gmail.com

Password

.....

Phone Number

Format: Country Code + Number

35382864282



Personal Information

First name

Demo

Last Name

User

Gender

☒ Male

☐ Female

Date of Birth

12/04/1998



Health Information

Height

Meters

1.8

Weight

Kilograms

90



Confirm Details Entered

Name : Demo User

Email : demo@gmail.com

Phone : 35382864282

Date of Birth : 1998-04-12

Gender : Male

Height : 1.8 meters

Weight : 90 kg



Successful Registration

Thank you for Registering with us you will soon receive an email to verify your account once you have verified your account please click the button below to login

Login

Login

Log In

Email

Password

Login ✓

Dont have an account?

Register

The user must then log into the application providing their email and password once they log in they are given the option to go their dashboard or visit their profile



Successful Login

You have successfully logged click on the options below to proceed!!!

← Dashboard

View My Profile →

Account Settings

Account provides the user with three options. They can view their account details , update their account or delete their account



[Logout](#) | [My Dashboard](#) | [My Profile](#)

Account Information

This is the information regading your account details

Name : Tomi Illori

Email : tomiillori123@gmail.com

Phone Number : 353872368957

Date of Birth : 1999-04-25

Gender : Male

Height : 1.8 meters

Weight : 90 kg



Account
Details

Update
Details

Delete
Account

Account Details
Update Details
Delete Account

Update Details

First Name
Tomi

Last Name
Ilori

Email
tomilori123@gmail.com

Phone Number
353872368957

Password
...

Account Details
Update Details
Delete Account

Account Information

This is the information regarding your account details

Name : Tomi Ilori

Email : tomilori123@gmail.com

Phone Number : 353872368957


Date of Birth : 1999-04-25

Gender : Male

Height : 1.8 meters

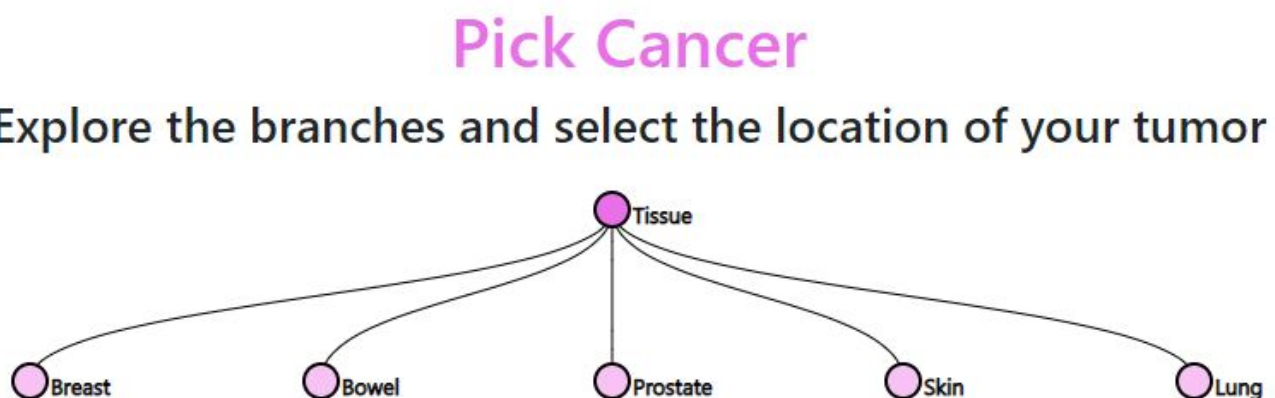
Weight : 90 kg

Delete Account



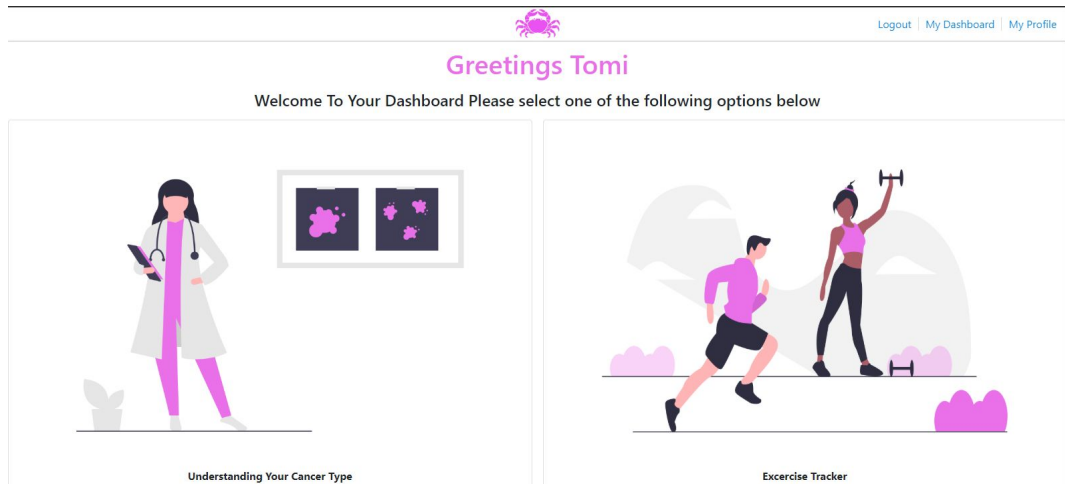
Cancer Selection

Users are given a tree with various types of cancer. Once the user selects the location of their tumor they are redirected to their dashboard



Dashboard

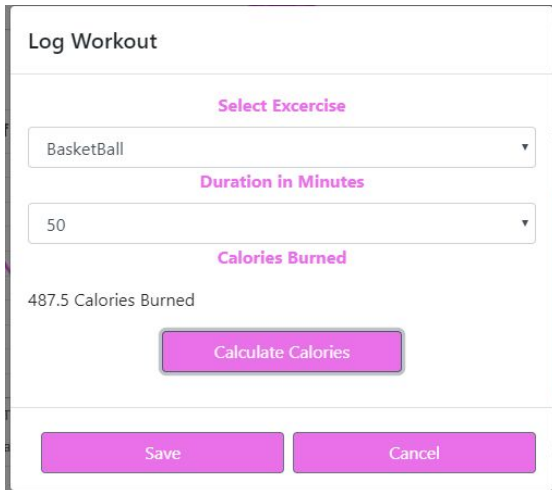
Once the cancer type has been selected users are brought to their dashboard which is made up of 5 parts



Fitness Tracker

This part of the dashboard is used to monitor the users physical activity there are two charts one to monitor the users daily target and how many calories they have burned and the other to monitor how many calories they burned each day of the week.





Log Workout

Select Exercise

BasketBall

Duration in Minutes

50

Calories Burned

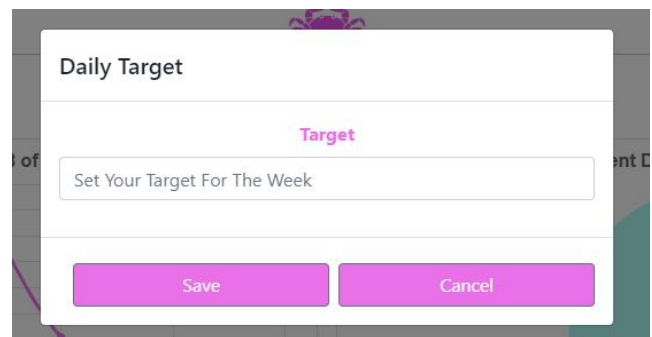
487.5 Calories Burned

Calculate Calories

Save Cancel

The user can also save a workout, through a dialog pane there is a selection of exercises and durations there is also a button that is used to calculate the calories burned for a specific workout the user can save this workout or cancel. When a workout is saved the charts adjust accordingly

The user can also change their daily target. When a target is saved the charts adjust accordingly

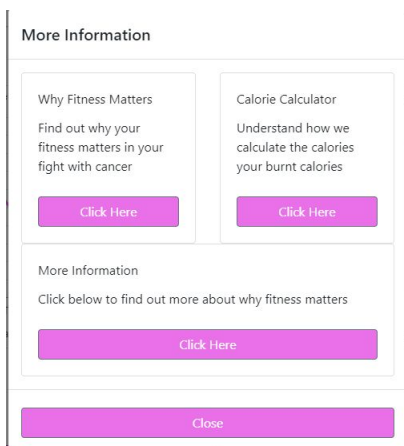


Daily Target

Target

Set Your Target For The Week

Save Cancel



More Information

Why Fitness Matters

Find out why your fitness matters in your fight with cancer

Click Here

Calorie Calculator

Understand how we calculate the calories your burnt calories

Click Here

More Information

Click below to find out more about why fitness matters

Click Here

Close

Finally there is a dialog pane with extra information for the user.

Diary

This part of the dashboard is used to act as journal for users consists of two sections the part where they can add diary entries and the part where they view the entries that they have

Write Something

Title

Sub-Title

Description

Submit

Most Recent Entries

Horrible Presentation
"Messed up"

ViewDelete

Best day ever
"Too good"

ViewDelete

Half done
"Nearly There"

Why Are the Colours Different?

The colour of each entry is determined based on the sentiment score when the application analyses the content of the diary entry. The user can view the entry and delete the entry.

When viewing the entry the user is given the option to edit the entry

Half done

Nearly There

well today i can say i am a little bit happy finally finished my final year project well most of it anyway i will probably improve it more in the coming week but i am happy with it right now

Date Created: 28/04/2020

Edit

Close

Edit Entry

Title

Half done

Sub-Title

Nearly There

Description

well today i can say i am a little bit happy finally finished my final year project well most of it anyway i will probably improve it more in the coming week but i am happy with it right now

Save

Cancel

Why The Colours Are Different?

Different Colours?

By now should notice that the colour of each entry is different, we designed this diary to actively read your entry and predict the mood that you may have felt at the time of entry , based on the content you wrote in your entry, your mood is placed on a score and this score is reflected in the colour of the entry

Here are the meanings of each color:

- Green = Very postive
- Blue = Slightly postive
- Grey = Neutral
- Yellow = Slightly negative
- Red = Very Negative

Close

There is also a dialog pane that explains to the user why the entries are coloured

Emotional Awareness

This part of the dashboard is used to show the user how their diary entries are analysed and also returns a list of articles based on the overall sentiment that the user has displayed from their diary entries.

Quote of Enlightenment

"Follow your instincts. That is where true wisdom manifests itself. - Oprah Winfrey"

Everytime the user enters this page a different quote is retrieved from an API contains quotes made by famous individuals in history

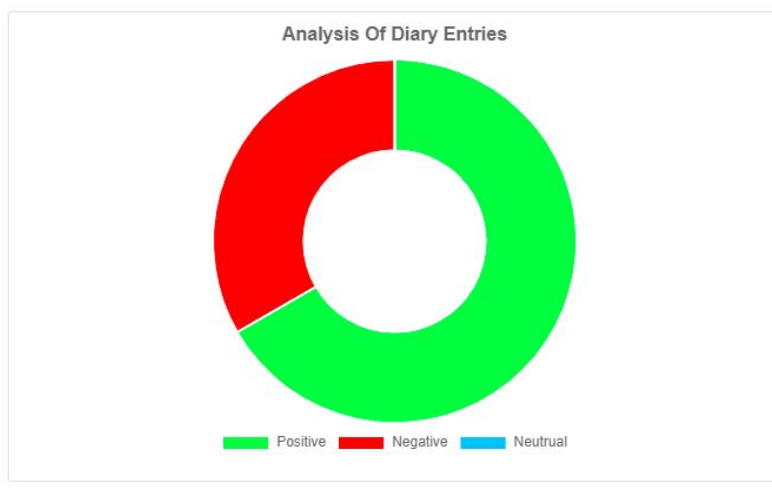
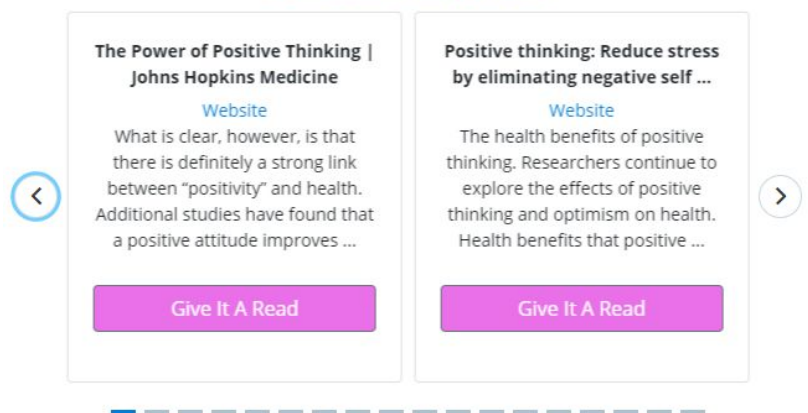


Chart displays the total number of entries that are either positive negative or neutral

Here is a carousel which retrieves 20 articles from a google search depending on the overall sentiment of the user each card has the article title , the website URL , a short description and button to redirect the user to the article page..

To most of your diary entries seem to be positive here's some articles to maintain that positivity



Calendar

This part of the dashboard is used by the user to maintain events in their lives users are able to add , view , edit directly or edit by drag and dropping and removing events, They can also connect to a calendar of their choice.

Users can view the Calendar by month by week or by day

<	>	today	April 2020				month	week	day
Sun	Mon	Tue	Wed	Thu	Fri	Sat			
29	30	31	1	2	3	4			
5	6	7	8	9	10	11			
12	13	14	15	16	17	18			
19	20	21	22	23	24	25			
26	27	28	29	30	1	2			
3	4	5	6	7	8	9			

	Sun 4/26	Mon 4/27	Tue 4/28	Wed 4/29	Thu 4/30	Fri 5/1	Sat 5/2
all-day							
6am							
7am							
8am							
9am							
10am							
11am					10:52 - 11:40 Meeting		
12pm							

<

>

today

April 30, 2020

month

week

day

	Thursday
all-day	
7am	
8am	
9am	
10am	
11am	10:52 - 11:40 Meeting
12pm	
1pm	

Connect to Google Calendar

Steps to connect Google Calendar

- In the Google Calendar interface, locate the "My calendars" area on the left.
- Find the Calendar you want to connect and click on the settings icon, then click settings and sharing
- Go to "Access Permissions" and select "make available to public"
- Go to "Integrate Calendar" and copy the "Calendar ID"
- Paste the contents below and click "Connect My calendar"

tomiilori123@gmail.com

Connect My Calendar

Close

Users can sync a google calendar of their choice once they input the ID of the calendar and click "connect my calendar" the page rerenders displaying all their events from google calendar

< >		today		April 2020			month week day	
Sun	Mon	Tue	Wed	Thu	Fri	Sat		
6a Any Assignment		9a Management 10a Distributed Sys 1p Strategic Manag		9a Android Lecture	12p Software Patter 3p Software Patter	4:30a Any Assignm 7:45a Chores 9a Gym		
19	20	21	22	23	24	25		
6a Any Assignment		9a Management 10a Distributed Sys 1p Strategic Manag		9a Android Lecture	11:30a Meeting 12p Software Patter 3p Software Patter	4:30a Any Assignm 7:45a Chores 9a Gym 2:52p Appointment		
26	27	28	29	30	1	2		
6a Any Assignment		9a Management 10a Distributed Sys 1p Strategic Manag		9a Android Lecture 10:52a Meeting	12p Software Patter 3p Software Patter	4:30a Any Assignm 7:45a Chores 9a Gym		
3	4	5	6	7	8	9		
6a Any Assignment		9a Management 10a Distributed Sys 1p Strategic Manag		9a Android Lecture	12p Software Patter 3p Software Patter	4:30a Any Assignm 7:45a Chores 9a Gym		

Create An Event

Start Date

End Date

Title

Importance

Users can create an event by clicking the “add event button” inputting the relevant information.

By clicking on an individual account users are presented with the options to view edit or remove the event selected

Event Details

Select one of the Following Options

Event Details

Title: Meeting

Start: 24/04/2020 @ 11:30

End: 24/04/2020 @ 13:0

Importance: Urgent

Edit Details

Change Event Details

Start Date

End Date

Title

Importance

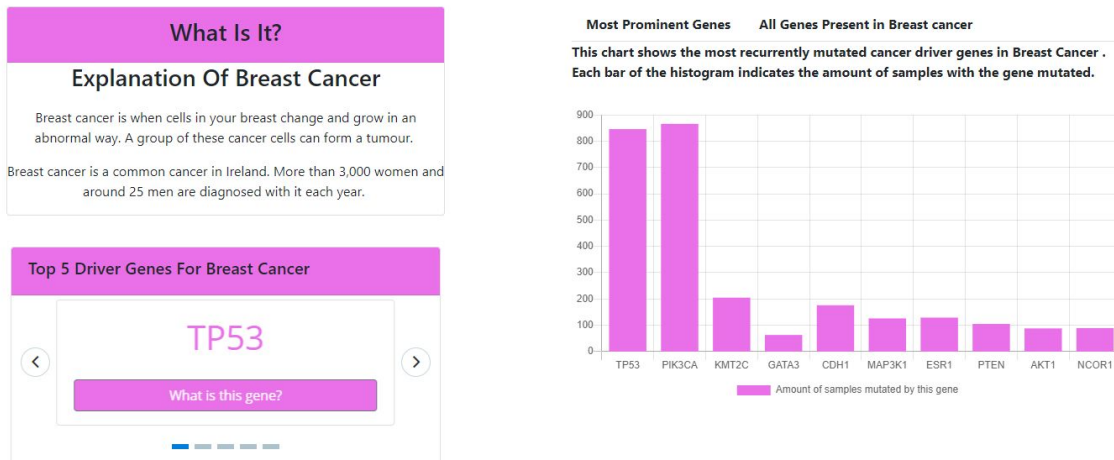
Understanding Your Cancer Type

This part of the application was mainly informational; it provides an explanation for what the cancer type is and shows the top 5 genes that are mutated in this type of cancer then there is a chart to display the top 10 genes by how many samples had this gene mutated.

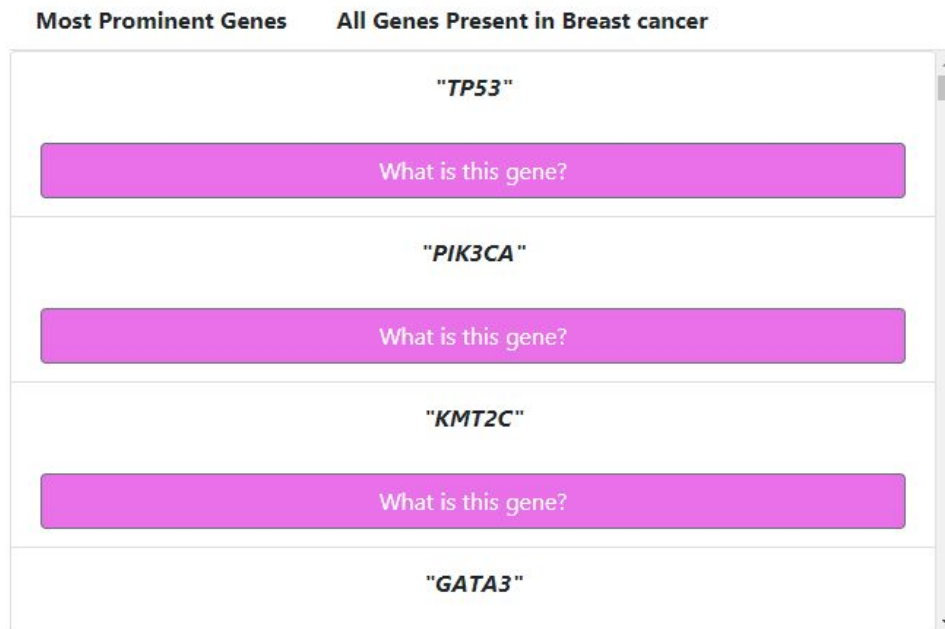


[Logout](#) | [My Dashboard](#) | [My Profile](#)

Cancer Type : Breast Cancer

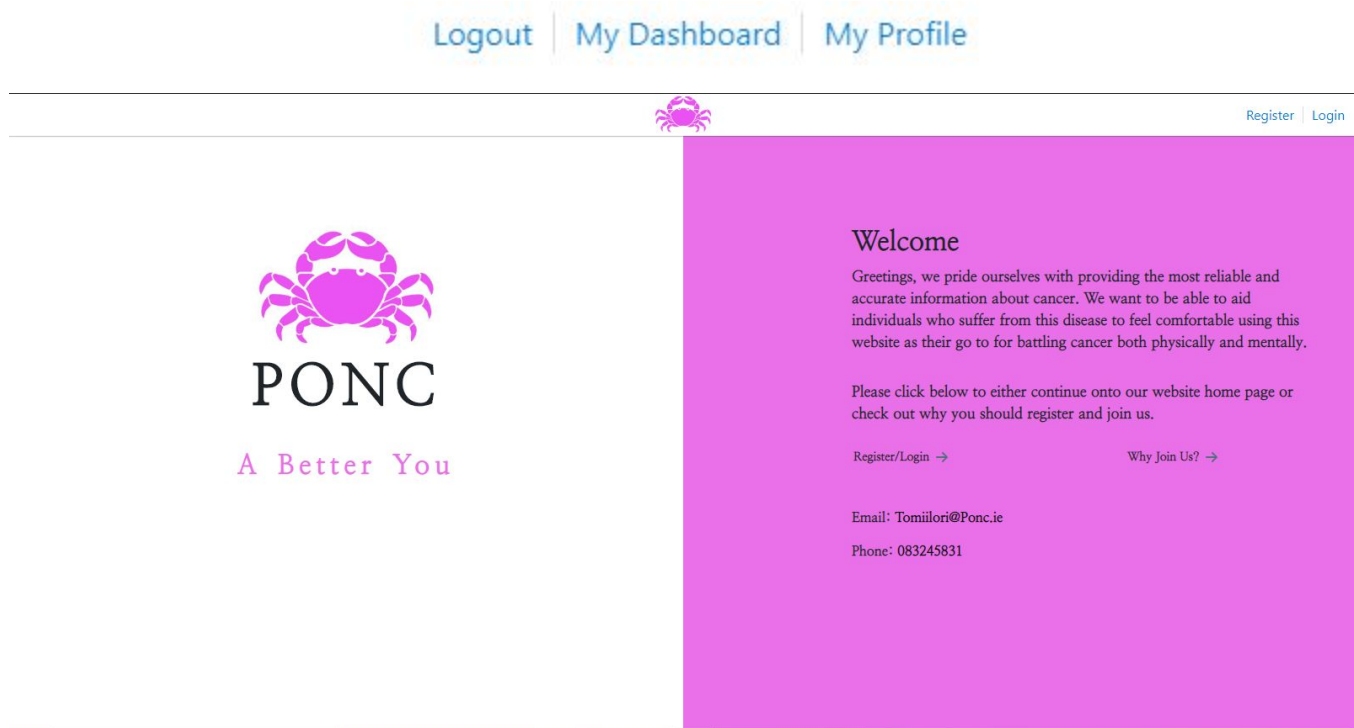


There is also a list of each gene that is associated with the type of cancer the user has each gene has a button that redirects the user to a web page with a trusted scientific description of what the gene is and what it does



Logout

To logout a user must click on the logout button in the navigation bar then they are redirected to the landing page



Project Management

Project management would have been my weakest attribute. I spent a lot of time focusing on module assignments rather than the final year project. This led me to properly starting the project at my third checkpoint which was a huge risk, it also led me to remove features or add more due to how little time I had to complete the project. When starting a new feature I would sketch out the layout of the page and then write out all the tasks I would need to complete the feature. I repeated this process and ended up reducing the time it took to complete features from a whole week to a few hours.

Test Plan

Action	Input	Expected Output	Actual Output	Test Results
Launch application	Npm run both	Landing page	Landing Page	PASS
Multi-Step form move forward	User clicks to move to next step in form	Following Page	Following Page	PASS
Multi-Step form move backward	User clicks to move to previous step in form	Preceding Page	Preceding page	PASS
User detail confirmation page	User inputs all data from fields	Information shown in a list	List of information	PASS
User created	After confirming details user object created in database	User object created in the database	User object with information from form present in the database	PASS
Email sent	After user is saved email is sent	Page indicating email has been sent and checking email inbox for the email	Page confirming email has been sent and the email is present in the inbox of the user	PASS
Email verification	User clicks link	Message indicating the user has been verified displayed and users verified property changed to true	Verified property set to true and message indicating this received	PASS
Login	User enters email and password to login	Confirmation that the user has been successfully logged in	Page that indicates successful login	PASS

View Profile	User clicks my profile in navigation bar	Brought to page with Account Details	Account Details Rendered	PASS
Update Account	User can edit information about themselves	Changes persist to database	Data persisted	PASS
Delete Account	User can delete their account	User object deleted from database	User deleted	PASS
Pick Cancer	User attempts to access dashboard after registration	Page with cancer selection displayed	Page displayed	PASS
Save Cancer type	User saves the type of cancer from the tree	Users cancer type is saved to the database	Cancer type saved and user redirected to dashboard	PASS
Save workout	User can save a workout	Workout persisted to the database and chart adjusts	Workout saved chart adjusted	PASS
Calories burned calculated	Selected exercise and duration	Value representing calories burned	Value calculated and rendered	PASS
Set Daily target	Daily target figure	Target persisted to the database and chart adjusts	Target persisted and chart adjusted	PASS
Add diary entry	Title subtitle and body of entry	Diary entry is stored in database	Entry stored	PASS
Edit diary entry	Changes to diary entry fields	Changes persisted to database	Changes stored in the database	PASS
View diary entry	User clicks "view" button	Modal displaying entry pops up	Modal appears with entry information	PASS
Entry colour is set	Depending on sentiment score colour is set	Colour should be either red , blue green , yellow or grey	Entry colour set	PASS
Sentiment Score	Body of diary	Value	Value calculated	PASS

is calculated	entry	representing the sentiment		
Sentiment is set	Sentiment Score	Value should be set to either true false or null	Value set to true false or null	PASS

Add event	Start time end time title and importance of the event	Event is stored in database	Event stored	PASS
Edit Event	Changes to event fields	Changes persisted to database	Changes stored in the database	PASS
View Events	On page render events fetched	Events displayed on calendar	Events Displayed	PASS

Remove Event	User clicks remove button to delete event	Event removed from the database	Event removed	PASS
Edit Event by Drag and drop	User drags event to different part of grid	Changes persisted to database	Changes persisted to database	PASS
View event	User clicks "view " button	Modal displaying event pops up	Modal appears with event information	PASS
Sync Google Calendar	User inputs Calendar ID of google calendar	Events obtained from google calendar and displayed on calendar	Events displayed on calendar	PASS
Quote pulled from API	User enters emotional awareness page	Quote generated and displayed at the top of the page	Quote displayed	PASS
Object containing analysis of diary entries	User enters emotional awareness page	Data should be retrieved and displayed on chart	Chart renders with correct data	PASS

Articles retrieved	Sentiment from analysis of entry data	Articles should be retrieved in correlation to uses overall sentiment	Articles displayed in correlation to user sentiment	PASS
Article link can be accessed	User clicks button to read article	User is redirected to article page	User is redirected to article page	PASS
Explanation of cancer type	User enters the understanding cancer page	Information should be displayed via web scraping	Information is displayed via web scraping	PASS
Top 5 driver genes	User enters the understanding cancer page	Carousel with genes should render	Carousel rendered	PASS
Chart with gene data of cancer type	User enters the understanding cancer page	Chart with gene data for cancer type should render	Chart rendered	PASS
List with gene data of cancer type	User enters the understanding cancer page	List with gene data for cancer type should render	List rendered	PASS
Gene link can be accessed	User clicks on link to read about gene	User is redirected to gene informational page	User is redirected to gene informational page	PASS
Return to dashboard	User clicks back button	User should be redirected to dashboard	User Redirected to dashboard	PASS

Logout	User clicks log out on navigation bar	User should be redirected to landing page	User is redirected to landing page	PASS
Dashboard via navigation bar	User clicks "My dashboard " on navigation bar	User should be redirected to dashboard	User is redirected to dashboard	PASS
5 genes returned	Users cancer type	The machine learning algorithm	Array of genes returned	PASS

		should return an array of genes corresponding to cancer type		
Gene data obtained from csv	User cancer type	Application should convert the corresponding csv file to json and return information as a json object	Gene data received as a json object	PASS
Calendar Views	User selects format they want to view calendar	Calendar should change to appropriate month , day or week view	Calendar view changes to selected view	PASS
Edit event duration by drag	User drags event to extend the duration	Change should persist to database	Change persisted to database	PASS
Return to landing	User clicks crab in the navigation bar	User should be redirected to landing page	User is redirected to landing page	PASS

Conclusion

Overall I am extremely happy with the fact that I was able to complete my project to this degree in less than a month. Even though i had to leave out features i was happy with what i was able to implement.

My career goal is to become a full stack developer which is someone who can work between the server side of applications and also the client side. This technology stack that I chose gave me great insight into what this role entails. I would have never expected myself to be able to put these technologies together myself. Because of how little the course focuses on javascript and client side technologies, to be able to develop this application is a testament to my determination to be a full stack developer as I was able to self learn and explore what was not taught in the course.

To extend this project in the future i would love to connect to an API related to food so users could monitor their calorie intake. I would also include games for people to play which would provide more information for users. Finally my biggest regret was not being able to properly implement machine learning the way i wanted to , in the future i would like to be able to create some sort of correlation that the application could teach itself to react differently to different cancer types and provide recommendations.

References

Serpstack.com. 2020. *API Documentation - Serpstack*. [online] Available at: <https://serpstack.com/documentation> [Accessed 30 April 2020].

V2.grommet.io. 2020. *Components - Grommet*. [online] Available at: <https://v2.grommet.io/components> [Accessed 30 April 2020].

Reactstrap.github.io. 2020. *Reactstrap - React Bootstrap 4 Components*. [online] Available at: <https://reactstrap.github.io> [Accessed 30 April 2020].

GitHub. 2020. *Bkrem/React-D3-Tree*. [online] Available at: <https://github.com/bkrem/react-d3-tree> [Accessed 30 April 2020].

freeCodeCamp.org. 2020. *How To Make Create-React-App Work With A Node Back-End API*. [online] Available at: <https://www.freecodecamp.org/news/how-to-make-create-react-app-work-with-a-node-backend-api-7c5c48acb1b0/> [Accessed 30 April 2020].

LogRocket Blog. 2020. *Building A Sentiment Analysis App With Node.Js - Logrocket Blog*. [online] Available at: <https://blog.logrocket.com/sentiment-analysis-node-js> [Accessed 30 April 2020].

npm. 2020. *Natural*. [online] Available at: <https://www.npmjs.com/package/natural> [Accessed 30 April 2020].

Irish Cancer Society. 2020. *Cancer Types*. [online] Available at: <https://www.cancer.ie/cancer-information-and-support/cancer-types> [Accessed 30 April 2020].

Intogen.org. 2020. *Intogen - Cancer Mutations Browser*. [online] Available at: <https://www.intogen.org/search> [Accessed 30 April 2020].

Reference, G., 2020. *Genes*. [online] Genetics Home Reference. Available at: <https://ghr.nlm.nih.gov/gene> [Accessed 30 April 2020].

GitHub. 2020. *Messagebird/Messagebird-Nodejs*. [online] Available at: <https://github.com/messagebird/messagebird-nodejs> [Accessed 30 April 2020].

Codemoto. 2020. *Email Verification In Node, Express, And Mongodb - Codemoto*. [online] Available at: <https://codemoto.io/coding/nodejs/email-verification-node-express-mongodb> [Accessed 30 April 2020].

Brain.js.org. 2020. *Brain.Js: Neural Networks In Javascript*. [online] Available at: <https://brain.js.org/#/> [Accessed 30 April 2020].

Business Insider. 2020. *How To Calculate The Number Of Calories You Burn Doing Anything, From Running To Sex*. [online] Available at: <https://www.businessinsider.com/how-to-calculate-calories-burned-exercise-met-value-2017-8?r=US&IR=T> [Accessed 30 April 2020].

Primereact, Available at: <https://primefaces.org/primereact/showcase/#/> (Accessed: 30th April 2020).

FullCalendar Documentation, Available at: <https://fullcalendar.io/docs> (Accessed: 30th April 2020).