An Intelligent Platform for Durability Monitoring of Reinforced Concrete Structures

Vagif R. Aliyev
H00236091
Submitted for the degree of BEng in Civil Engineering

Dissertation Supervisor: Dr. Benny Suryanto

School of Energy, Geoscience, Infrastructure and Society
Heriot-Watt University
April, 2019

Abstract

With ever increasing amounts of concrete structures, the monitoring of the performance of the surface zone of concrete has become more essential as it is a major factor in governing the rate of deterioration and predicting the in-service performance of the structure. Prof W.J McCarter has developed a remote interrogation system in Dornoch (Scotland) to study the performance of the cover-zone concrete. Previously, the data from the system had to be retrieved, combined, corrected and graphed manually through Excel for monitoring and analysis.

This piece of research focuses on developing a web application that can automate the processes previously performed manually and enhance the user experience for better monitoring and analysis of data. A database system was created to automatically add the readings from the system developed. The user can specify the exact combination of sensors and declare variables for data correction. The corrected values are then displayed as scatter charts with features such as interactive legends, hoover tooltip and range slider that provide more clarity to data compared with static charts in Excel. For a more accurate analysis, the activation energies for the specimens are calculated and presented to the user. To provide more insight into the site conditions, a separate database containing temperature and tidal data from the nearest stations was developed.

Website has been successfully launched on university servers and can be accessed at http://137.195.70.233/

Declaration Statement

I, Vagif Aliyev, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g. ideas, equations, figures, text, tables, programmes) are properly acknowledged at the point of their use. A full list of the references employed has been included.

Signed: V. ALIYEV

Date: 05/04/2019

Acknowledgments

Firstly, I would like to express my sincere gratitude to Dr. Benny Suryanto, my dissertation supervisor, for sharing his depth of knowledge and his consistent help during the entire process.

I would also like to express my thanks to John Spinks from the IT department at Heriot-Watt University for providing me with the documentation to learn how to launch a website successfully.

Table of Contents

Table of Figures

Table of Tables

Chapter 1 – Introduction

1.1 Background

A problem faced by many engineers worldwide is the premature deterioration of concrete highway structures. We have reached a time where the major part of developed countries infrastructures capital cost have decreased, however inspection and maintenance cost have grown to such a degree that they form majority of recurring cost of the infrastructure. The aim of the integrated monitoring system for new and existing reinforced structures is to reduce costs by assessing repair options and arranging inspection and maintenance programmes in accordance with the system, hence cutting down road closures which result in less traffic delays (Chrisp et al., 2010). Continual monitoring of cover-zone concrete (convercrete) also enables for a more detailed assessment of cover-zone performance. Integrated monitoring systems can also assist in correctly estimating both life-cycle cost and service-life calculations. Using the retrieved data alongside service-life models can yield in reduction of life-cycle costs, consequently making up for the installation costs of the monitoring systems.

The general line of action for structural monitoring, especially in arduous to access and remote sites, is to arrange site visits by engineers who can use handheld equipment for electrical measurements. Under such circumstances, the frequency of data is limited and provides only ‘snapshot’ view of covercrete condition, not accounting for the seasonal wetting and drying. Increasing the frequency of site visits would be financially not feasible for commercial practices due to the cost of travel and substance for the personal involved.

Previous research by Prof W.J McCarter about a test site in Dornoch, Scotland has shown that monitoring of covercrete as a function of time and depth, can be used to assess the current and future performance of the cover-zone (McCarter et al. 2012). Hence a sensor system to study this relationship was developed, however this only forms one significant element of an integrated monitoring system. Other elements were developed as well, to enable the data from the sensor to be retrieved without needing to visit the site through communicating with the system in Dornoch via a modem system. However, the information received from the site arrives in single files corresponding to the date recorded and has to be

combined together with other data manually. These data then needs to be corrected via mathematical formulae. Lastly, for analysis, graphs have to be created in excel which is time consuming and has limited options.

1.2 Aims and Objectives

This dissertation aims to develop an intelligent platform that would enable automatic monitoring of reinforced concrete structures without the presence of human interventions. This is a significant improvement to the current processes that are done manually and provide new tools to allow user to analyse the data in more depth.

The objectives of this study are:

(i) Web Application: design an interface that allows user to declare which sensors to read from. Correct these data according to the appropriate protocols. Display the various data calculated as scatter charts with interactive tools.

(ii) Automated Database: design an efficient programme that can automatically copy the contents of regularly incoming csv files into a single file in correct order of date and time.

(iii) Extra features: calculate the activation energies via the appropriate protocol in accordance with user's specification. Create a database for temperature and tidal readings from nearby stations.

Chapter 2 – Literature Review

2.1 Web Application Development

A web application is a client-server application that runs on a remote server where a browser acts as the client. Storage of data is primary performed on the server and is exchanged over the network via Hypertext Transfer Protocol (HTTP). Web applications are different from other applications as they do not require any installation. Hence, users are not limited to a specific operating system or hardware configuration, allowing the application to be a cross-platform service.

2.1.1 Front-End

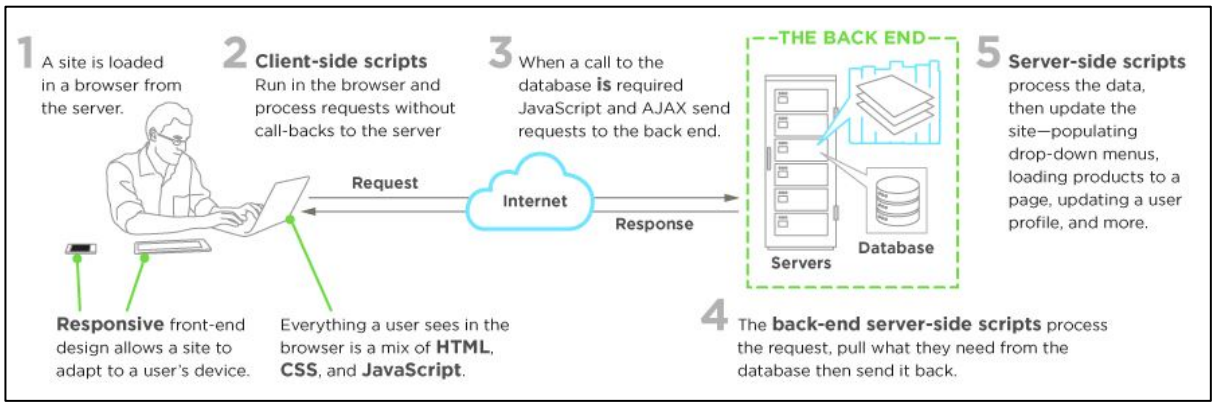


Figure 2.1 - Front-end development (Wodehouse, n.d.)

Front-end development, also referred as client-side development, involves developing the public part of a website, involving interaction with the user. Requests that the user creates with clicks are sent to the back-end for processing and return appropriate data to the front end (see Figure 2.1). One of the other purposes of front-end development is to make the user interface appealing. The information displayed should be in a format that is relevant and easy to read. There are variety of tools available to develop the front end of a website, and the core tools are: Hypertext Markup language (HTML), Cascading Style Sheets (CSS) and JavaScript (JS).

Hypertext Markup language (HTML) is the foundation of any webpage development, which is essential for a web page to exist. Hypertext refers to a text that has a link embedded in it termed hyperlink. Upon clicking on the word, the user will be brought to a new page. Markup language defines the overall structure of the site via images, tables, header and etc. HTML was developed

by physicist Tim Berners-Lee. Latest version of HTML called HTML5 was released as a stable W3C Recommendation on 28 October 2014(W3.org, 2014).

CSS is in charge of controlling the presentation of a site, providing the site with its own unique look. Every component defined in HTML can be styled and tuned by the developers. This is done by providing style sheets that are layered on top of style rules which are activated based on inputs such as device resolution and screen size.

JavaScript is an event-based imperative programming language that is the backbone for transforming static HTML pages into a dynamic and interactive interface. Document Object Model (DOM) provided by HTML standard can be used to change the structure of the site based on a user input such as mouse clicks. Using a feature called AJAX, stream of data can be sent to and received from a server without refreshing the website, thereby making the web page more dynamic.

2.1.2 Back-End

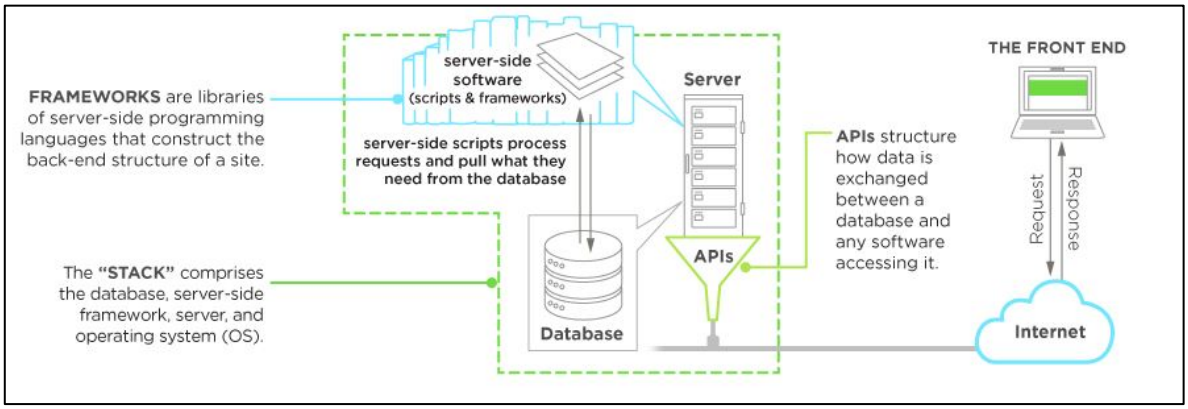


Figure 2.2 - Back-end development & frameworks (Wodehouse, n.d.)

Back-end development, also referred to as the server-side development, focuses on web application logic for the server and databases. This section powers the functionality of the application and is not seen by the user. The back end can be split into three sections: the server, the database and software for application logic (shown in Figure 2.2). In simple terms these can be described as: the server running the back-end software, the database storing applications data and software establishing communication between the two.

2.2 Technologies

2.2.1 Node.js

Node.js or Node is an open source, cross platform runtime environment licensed by MIT for executing JavaScript code outside of a browser. NodeJS was developed by Ryan Dahl in 2009 and its latest version is v0.10.36. NodeJS is built on Google Chrome's JavaScript Engine (V8 Engine) for easy development of fast and scalable network applications. The official website defines node,js as "using an event-driven, non-blocking I/O model that makes it lightweight and efficient, which is perfect for data-intensive real-time applications that run across distributed devices" (Node.js, n.d.). This would make it perfect for use in the work presented in this Dissertation. Node is often used to build back-end services, also called APIs or Application Programming Interfaces, which powers the client applications e.g a web app running on a web browser.

Node is ideal for building highly-scalable, data-intensive and real-time back-end services that power out client applications. There are other tools and frameworks out there available for building back-end services such as ASP.NET or Django. However, there are some aspects that make Node special than others:

(i) The most recent Octoverse infographic by Github ranks languages by number of pull requests received and JavaScript is placed at top by a big margin, shown below in Figure 2.3 (Github Inc., 2019).

Figure 2.3 - Languages ranked by pull (Voss, 2018)

(ii) It is relatively easy to setup and can be used for prototyping and agile development but can also be used for building superfast and highly scalable services.

(iii) It has been used by large companies such as PayPal, Uber and Neflix, amongst others. During rebuilding one of their Java based applications using Node, PayPal

found that it was built twice as fast with fewer people, 33% fewer lines of code, 40% fewer files and decreasing the average response time by 35% (Costa, 2019).

(iv) Another reason to choose Node is being able to use JavaScript, meaning that any front-end developer knowledge on JavaScript can be reused hence no new programming language will have to be learned. More importantly, being able to use JavaScript on both front-end and back-end will result in source code being cleaner and more consistent, using same name conventions, tools and best practices.

(v) Lastly, it has the largest ecosystem of open source libraries available (Docs.npmjs.com, 2019).

2.2.2 Node Packet Manager (npm)

NPM is the default package management and distribution system for NodeJS. It is also the world's largest software registry by a significant margin (shown in Figure 2.4) with over 700,000 open-source libraries available (Voss, 2018).

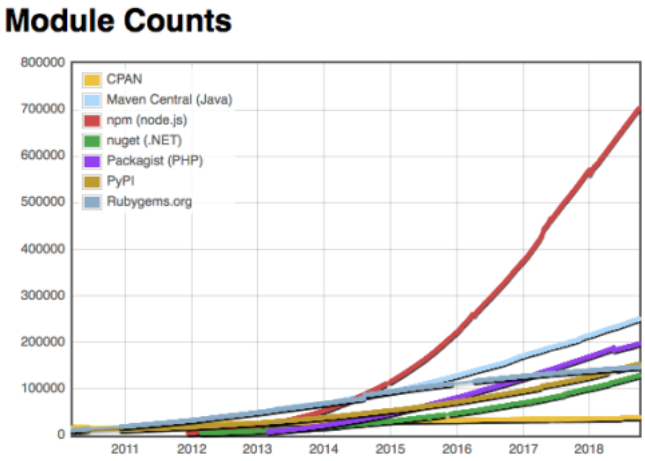


Figure 2.4 - Module counts of various registries (Voss, 2018)

Open source developers around the globe use npm to share packages. It enables Node.js modules to be published and distributed via simple command line interface. Instead of installing each library separately, one command "npm install" can be run and install everything in one go to save time. When the command is run, npm will look for the package.json file in the current folder. If found, it will install each library from the list.

2.2.3 Express.js

Express.js also referred as express is a minimalistic and flexible web framework for Node applications, providing large amounts of functions and features. Having access to extensive set of HTTP module services and middleware handlers, allows for easier and faster way of developing reliable and robust API compared to the core node modules (Mills, 2019). Express is the most popular Node web framework. Popularity of a web framework is important as it is a good measure of continual maintenance and available resources such as add-on libraries and technical support.

2.2.4 Plotly

Plotly, also referred as ploy.ly due its URL, is a company that develops data visualization and analytics tools. Their headquarters is located in Montreal, Quebec. Online graphing, analytics and statistic tools they provide is used by many individuals and corporations such as NASA and Apple.

2.2.5 Heroku

Heroku is one of the first cloud hosting platforms. It was first released in 2007 supporting Ruby programming language but it has evolved since then to support languages like Java, Python, Node.js. It's ease of use has made it top choice for developing projects by many. Heroku platform manages and maintains hardware and server, this allows for the user to focus on implementation of the idea, without having to set up the required supporting infrastructure. Processes like deployment, configuration, scaling and tuning are simplified further easing the development of the app for the user. Application deployment is managed through the use of Git. This is done through a single step of pushing the Git repository to Heroku servers.

2.2.6 Github

GitHub is a hosting service which allows users to host, share and collaborate on projects together. It can be used freely for open source projects. Projects hosted on GitHub are called repositories. Users can view their projects, pull down a version of this project to their device or add to the project.

A few simple git commands that are most commonly used when working on git projects would be:

- "git add" adds all files that are not already pushed to your GitHub repository.
- "git push" a simple 'push' to your repository on GitHub

2.2.7 Python

Python is considered a high-level general-purpose programming language. The first version was released in 1991 and create by Guido van Rossum. The current version, Python3, has been released in 2008. It has been used in variety of projects such as powering Instagram and testing Intel microchips. The process of manipulating files is an essential part of scripting in Python and has inbuilt functions to make creating and reading files less complex.

2.2.8 PostgreSQL

PostgreSQL is a database management system for general purposes. It is also referred as the most advanced open source database system. It has been in active development for 15 years and has earned strong reliability by the community. It has been developed by many volunteers across the world. Hence it is not owned or controlled by any corporation and is available for free. Companies like Apple and Fujitsu have built solutions via PostgreSQL.

2.2.9 OpenWeatherMap API

An application programming interface (API) is a set of subroutine definitions that enables software to software communication. Software companies make their APIs public so other developers can design products powered by their app's data.

OpenWeatherMap API is recognized as one of leading providers of digital weather information. It has been developed by a small IT company in 2014 and its headquarters is located in the UK and have offices around the globe. As of 4th of June of 2015, Google has closed down their Weather and Cloud solutions and has partnered with OpenWeatherMap and recommended its use as an alternative. Currently, this API has over a million users and acquire about 1500 new users daily (Openweathermap.org, n.d.). Basic features are free to use however paid plans for developers are available for more complicated levels of weather data and support.

2.2.10 Environment Agency Tide Gauge API

The UK National Tide Gauge Network acts as an access way for marine data sets held by the Environment Agency. The API is provided as open data under the Open Government Licence with no requirement for registration. The network has access to 44 locations around the United Kingdom coast for tidal elevations. The measurements are reported in near real time, every 15 minutes (Environment.data.gov.uk, 2019). The API provides information on the stations monitoring tidal data and their measurements.

2.3 Monitoring station: Dornoch Test Site

2.3.1 Test Site

In 1991, the construction of Dornoch Firth bridge was completed in north-east of Scotland (shown in Figure 2.5 (a)). The bridge was constructed using pre-cast concrete decks which were incrementally launched across reinforced concrete piers which were cast in-situ. The location of the site was chosen nearby, on the southern causeway that lead up to the road-bridge (show in Figure 2.5 (b)). Marine exposure site in Dornoch, provided the ideal environmental conditions for the research purposes. Site was successfully launched in 1998 and samples have been exposed to continuous splash, tidal cycling and airborne spray conditions ever since. Dornoch site is considered as part of the world-wide study (Holmes, et al, 2010). The concrete specimens on the site contain electrical sensors capable of measuring the water, ionic and moisture movement via an electrode array developed by Prof. John McCarter at Heriot-Watt University.

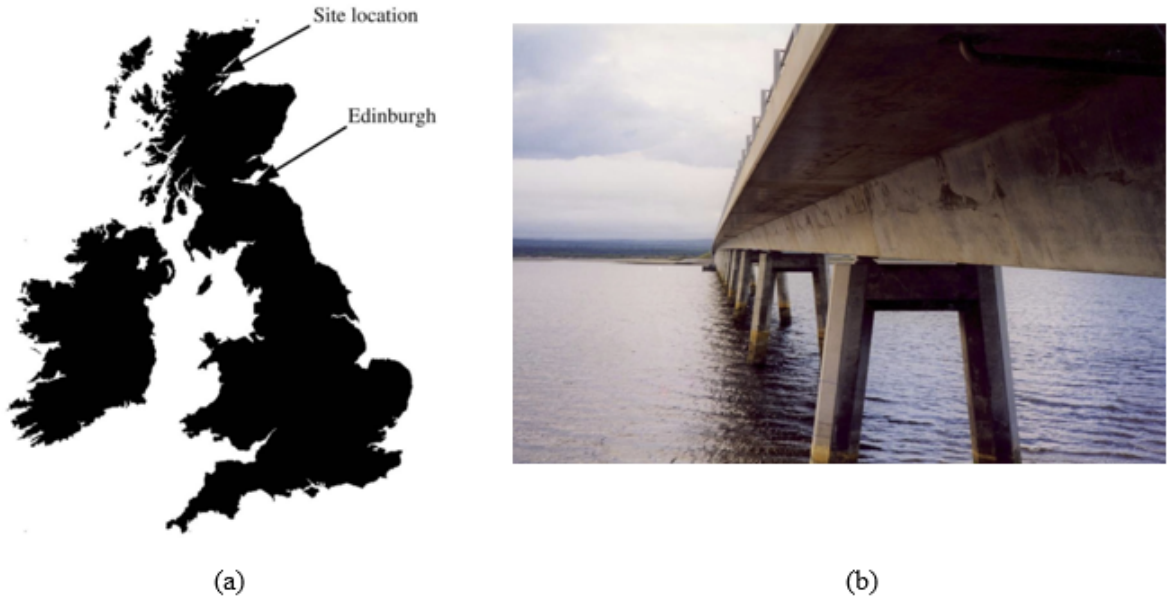


Figure 2.5 - (a) location of marine exposure site (b) bridge spanning the Dornoch Firth estuary (McCarter et al. 2012).

2.3.2 Sensors

Within the cover zone of concrete specimens, multi electrode array was embedded which enabled spatial distribution of electrical resistance to be monitored (McCarter et al. 1996). These measurements are essential to study movement of water, ion and moisture within the surface region (McCarter et al. 2012). Temperature distribution within the cover region can be monitored via the array as well. The array compromised of electrode pairs fitted on a polyvinyl chloride (PVC), in the shape of T which was secured using two stainless-steel bars (show in Figure 2.6 (a)). The length of the bars are designed in accordance to reinforcement detailing and are used to attach the array to the reinforcement. Additionally, at the points of contact the bars have been electrically isolated from reinforcement. A single electrode consists of a stainless-steel pin of 1.6 mm diameter that is sleeved to expose 5 mm tip. The pairs are set up to have 5 mm centre to centre spacing between the pins. Marine grade 316 was selected for all the stainless-steel sections. The electrode pairs were arranged at 5, 10, 15, 20, 30, 40, 50 mm from the concrete surface (shown in Figure 2.6 (b)) and four thermistors were positioned at 10, 20, 30, 40 mm from the concrete surface. To be able to convert measured resistance, R (ohms), to resistivity ρ (ohm-cm) the electrodes were calibrated in a solution of known resistivity before embedding them. This can be done through Equation (1):

$$\rho = kR \tag{1}$$

where *k* is the calibration factor for the array used in the concrete specimens in the test site, which was obtained as 1.25 cm ± 5%.

The resistance of concrete between each electrode pairs were measured using an auto ranging logger. To minimize electrode polarization effects, operating voltage of 1.0 V at a fixed frequency of 1 kHz which was indicated in previous studies (McCarter and Brousseau 1990). The same system was also used to retrieve the thermistor measurements.

Figure 2.6 - (a) rebar attachment and the array (b) placement of electrodes on the array (McCarter et al. 2012).

2.3.3 Specimens

The concrete specimens were exposed to a marine environment which included submerged, tidal and spray zones. Concrete mixes were determined in accordance with the requirements specified in BS-EN 206-1:2000 (BSI 2000b) for all exposure conditions and are displayed in Table 2.1.

Table 2.1 - Concrete mix design (Kim et al. 2018)

Mix designation	CEM I (kg/m³)	GGBS (kg/m³)	FA (kg/m³)	Coarse aggregate		Fine aggregate (kg/m³)	WR* (l/m³)	w/b	f_{28}** (MPa)
				20 mm (kg/m³)	10 mm (kg/m³)				
CEM I (PC)	460	+	+	700	350	700	1.84	0.4	70 (64.9)
CEM III/A (GGBS/40)	270	180	+	700	375	745	3.60	0.44	53 (53.9)
CEM II/B-V (FA/30)	370	+	160	695	345	635	2.65	0.39	58 (49.0)

* WR water reducer;
** f_{28} 28-day compressive strength. Figure in brackets represents compressive strength for migration test samples.

The slabs were 300 × 300 × 200 (thick) mm and plywood formwork was cast against the exposed working face. The array which was discussed in the previous section was installed at the centre of each slab. However, mild-steel ribbed reinforcing bars were used instead of the stainless-steel bars and positioned 50 mm from the surface. The schematic diagram is shown below in Figure 2.7.

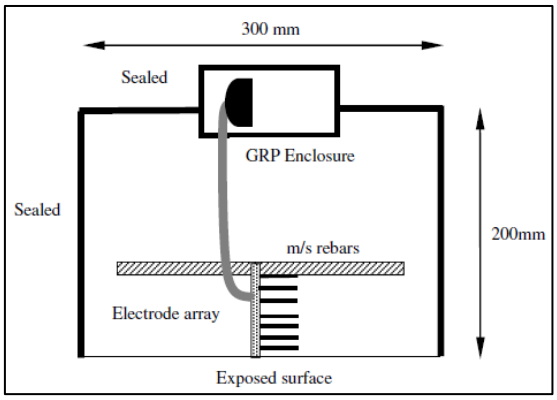


Figure 2.7 - Diagram of embedded array. (McCarter et al. 2012).

The ends of rebar were coated in epoxy with a cover of 50 mm. The bars were electrically connected to allow for half-cell potential to be measured. To limit the movement of ions and moisture into one dimension (1D), all surfaces except the exposed surface were applied an epoxy-based paint several times. Colour-coded cables were used to connect the electrodes and terminated to a 37 pin multi-pole female D connector which was sealed inside a watertight glass-reinforced plastic (GRP) enclosure (shown in Figure 2.8).

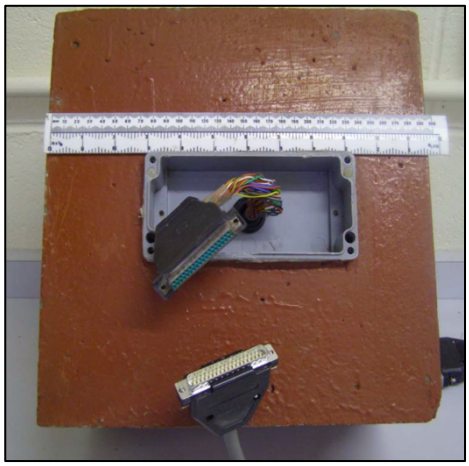


Figure 2.8 - GRB box enclosing a 37-pin D connector and lead (McCarter et al. 2012).

Transportation of the slabs to the marine site occurred approximately 5 weeks after demoulding and secured in galvanized steel frames. Six specimens per mix were positioned at three different exposure environments (shown in Figure 2.9) specified in BSI 2000b and 2006a:

(1) XS1 – Above the high-water level in the spray zone
(2) XS3 – Under the high-water level, referred as the tidal and splash zone
(3) XS2 – Below mid tide level, referred as submerged zone

A sample from each mix (total of 6 samples) in XS2 and XS3 exposure conditions were connected to the interrogation system to enable remote monitoring of the specimens.

Figure 2.9 - Specimens installed inside frames in XS1, XS2, XS3 positions (Kim et al. 2018)

2.3.4 The Remote Monitoring System

During the initial stages of the site, 1998, measurements on the slabs had to be taken manually during site visits. The remotes of the site made the frequent monitoring financially not feasible due to cost of travel and personnel involved, this resulted in erratic data sets. In order to gain a better understanding of cover-zone concrete performance in alternating environmental conditions, the rate of data retrieval had to be increased. Researches decided to develop a system that would eliminate the need of site visits via remote access and feed continuous stream of site data all year round. Developed system has been in place since November 2010.

As mentioned in the previous section, six samples were connected to the system for monitoring. Connection between the system and the female D connector of the sample was done via a connecting cable with 37-pin male D connectors at the ends. An epoxy based potting compound was used to flood the GRP box (show in in Figure 2.8) which was covering the male-female connections of the samples.

One of the concrete pier stems is being used to accommodate the system which comprises of two watertight enclosures (show in Figure 2.10 (a)). One of the enclosures is designed for the multiplexing unit (show in Figure 2.10 (b)) and the other one for the controller and the circuitry for resistance measurement (show in Figure 2.10 (c)). Connection between the controller and the multiplexing unit is permanent. Multiplexing unit was connected with the embedded arrays via cabling for all six samples.

The system draws its power from a solar panel (show in Figure 2.10 (d)) through a heavy-duty battery. In order to prevent overload, Steca charge controller is used to control the charge from the solar panel to the battery. Voltage/current is sourced to the system, for the logging and communication services, via the same controller (Chrisp et al. 2010).

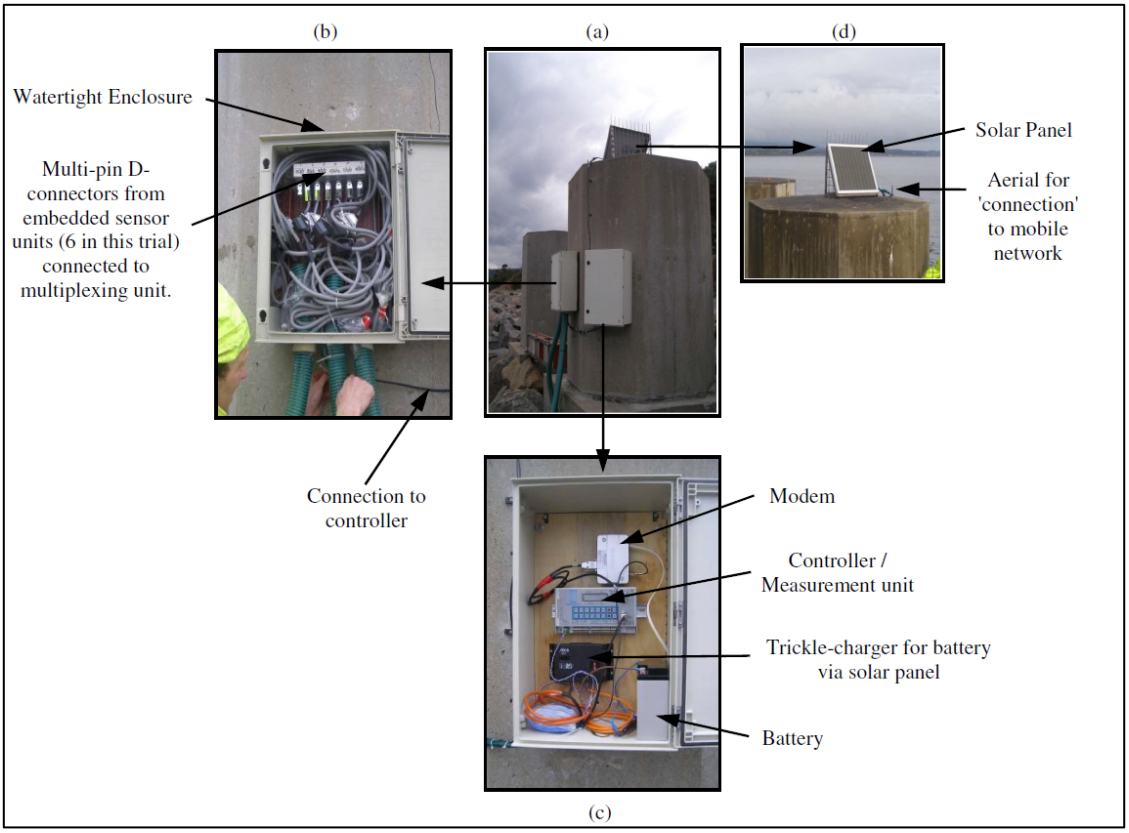


Figure 2.10 - (a) watertight enclosure for telecommunication and monitoring services; (b) cables from sensors terminating at multiplexing unit; (c) technologies involved in enabling remote access to data (d) Aerial for telecommunication and solar panel (McCarter et al. 2012).

The electrical data is retrieved and stored using the Instrument Solutions Resistance Data Logger which makes up the centre of the system. The logger can be interfaced to any desired sample sensors through a six-channel multiplexer unit. The telecommunication with the system is created by a GSM modem via 'dial-up' approach. Transparent link between the computer and the serial port on the controller is created via a computer mounted software that provides data connection to the modem. Recovery of data is executed the same method as a direct RS322 connection using standard ASCII commands via the computers modem. As shown in Figure 2.10 (d), a small antenna for modem to communicate is mounted by the solar panel and a protective conduit is used to connect both of them securely to the rest of the system. The GSM modem has a SIM card installed with a mobile airtime contract.

To reduce the overall power consumption, the logger uses the "wake / sleep" mode which draws minimal current between logging events. During non-active communication periods the modem is still continuously powered however just enough to answer calls (10% of full power). Upon detection of incoming data connection, the controller is woken up via the data carrier detect line on the modem serial port (the system is called up on from the laboratory). The data

stored by the system can be accessed and downloaded at any given time. For further power reductions, timer can be set on the power supply of modem to come on for a predetermined period for data recovery (e.g weekly). The measurements are returned as csv files.

2.3.5 Temperature monitoring and correction protocol

The resistance values measured by the thermistors can be converted into degree Celsius (°C) using the Steinhart-Hart equation:

$$T = [A + B\ lnR + C(\ln R)^3]^{-1} - 273.15 \quad (2)$$

Where,

T is the temperature (°C),

R is the resistance measured by the thermistor (ohms),

A, B and C are coefficients dependent on the type of thermistor and are 1.28×10^{-3} K^{-1}, 2.36×10^{-4} K^{-1}, and 9.31×10^{-8} K^{-1} respectively

ln stands for natural logarithm

To model the influence of temperature on resistivity, an Arrhenius relationship is used;

$$\rho = \rho_o e^{\left[\frac{Ea}{R_g T_k}\right]} \quad (3)$$

Where,

ρ is resistivity (kΩ-cm),

T_K is absolute temperature (K),

ρ_o is pre-exponential constant (kΩ-cm),

R_g is gas constant (8.3141×10^{-3} KJ/mole/K) and

E_a is activation energy for conduction processes in concrete (kJ/mole).

If ρx and ρy are the resistivity readings at temperatures $T_{k,x}$ and $T_{k,y}$, respectively, then, from Eq. (3):

$$\rho_x = \rho_y e^{\frac{Ea}{Rg}\left[\frac{1}{T_{k,x}} - \frac{1}{T_{k,y}}\right]} \quad (4)$$

From Eq. (4), a value of resistivity, ρ_y, recorded at a temperature, $T_{k,y}$, can be used to get an equivalent resistivity, ρ_x, of the material at temperature, $T_{k,x}$, with a known E_a/R_g ratio for the conduction stage. The process standardizes the measurements to a reference temperature, removing the effect of temperature on the electrical resistivity. The activation energy can be

entered either by the researcher or can be evaluated from the data for a more accurate value. The reference temperature ($T_{k,x}$) is provided by the user, usually around 25°C (298.15 K).

2.3.6 Evaluation of Activation Energy (E_a)

Through the experimental measurements the ratio of E_a/R_g can be determined and as the gas constant (Rg) is known, the activation energy (E_a) can be evaluated for the particular mix and electrode pair depth. Equation (3) can be rewritten as:

$$\ln \rho = \ln \rho_o + \frac{E_a}{R_g T_k} \tag{5}$$

Where,

ρ is resistivity (kΩ-cm),

T_K is the thermistor value (K),

ρ_o is pre-exponential constant (kΩ-cm),

R_g is gas constant (8.3141 × 10^{-3} KJ/mole/K) and

E_a is activation energy for conduction processes in concrete (kJ/mole).

Pearson product-moment correlation is applied to draw a line of best fit through the values of ln ρ and $T_k/1000$. The slope ($b = E_a/R_g$) of this line can be calculated from Equation (6):

$$b = r\frac{S_y}{S_x} \tag{6}$$

Where,

r is the Pearson correlation coefficient

Sy is the standard deviation of ln ρ values

Sx is the standard deviation of $T_k/1000$ values

Pearson correlation coefficient (r) can be calculated from Equation (7):

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2 \sum(y-\bar{y})^2}} \tag{7}$$

Where,

x is the ln ρ values

y is the $T_k/1000$ values

Chapter 3 – Work Programme

3.1 Work Package 1: Web Application

The main programming language for this project was chosen to be Node.js. There are several reasons for this and the most important one is being able to develop both the front-end and back-end in JavaScript using a runtime environment. Knowledge of JavaScript is a requirement for front-end web development and being able to use it for back-end as well will save valuable time. Deployment of web application is simplified as well since all web browsers support JavaScript. Data correction requires Steinhart-Hart equation and Arrhenius relationship to be applied to the dataset. This is a relatively easy task for any programming language to handle, and thus node.js is sufficient for this task. Node.js 10.15.3 LTS has been chosen for this project as it is the most recommended version by the company (available at https://nodejs.org/en/download/).

Node.js also has a large active community of developers that contribute continuously to further development. Libraries provided by the community in GitHub can be used to make certain developments easier and faster to implement. One of such libraries used in this project is Plot.ly to enable the ease integration of interactive graphs into the website. Plotly's API is relatively easy to learn compared with other options available. Examples provided are informative and straightforward to implement to our own specific design requirements. A variety of created charts by other users is available for inspection as well. Plot.ly comes with several built-in functionalities in which otherwise would need to be developed separately. It also comes with dynamic elements such as *hover tooltip* that would allow to display data values next to the plotted element. It is also possible to add buttons thereby enabling views of different layers of data to be displayed on the same chart. Furthermore, it is possible to add interactive sliders that can be used to adjust the view on the chart, allowing a user to focus on the response within a certain period of time. Interactive tooltips like these make determining specific value of points easier than with static Matplotlib charts (3leafnodes.com, 2017). Clarity provided by these tools would also allow for better navigation throughout a big dataset. Instillation of Plot.ly node module is very simple as npm is used, which can be done by a command "npm install plotly" – the only requirement for installation.

As this is the initial stage for the construction of the web platform, the dataset will be provided and automatic database will be implemented after this section is fully operational. For the development of the web application, csv files for the month of November 2017 have been provided by the Dissertation supervisor.

Using Heroku, the website will be launched on their platform and its performance will be assessed. The dataset fed to the system will be analysed on Microsoft Excel as well and compared with Web App results, in order to ensure that the calculation procedures have been applied correctly.

In this WP, the following procedure were to be followed:

1. User Interface

   The first action that needs to be taken was to determine which dataset the user wish to be displayed. Therefore, the user is given five fields to choose from a pulldown menu. Specimen number (range between 1 to 6), depth of the array (5, 10, 15, 20, 30, 40, 50 mm), thermistor number (range between 1 to 4), target temperature (ºC) and Activation energy (kJ/mol) are the fields that a user needs to populate at the start.
2. Data Correction

   Given that the specimens were subjected to the natural environments, the readings need to be corrected, using the correction protocol develop by Prof. W John McCarter. Using the specified dataset which has been retrieved, mathematical formulae were applied to convert the resistance values into resistivity and temperature readings.
3. Graphing

   These readings were then fed into Plot.ly to display two graphs; raw & corrected resistivity (ohm) against time and corrected thermistor readings (Cº) against time.

3.2 Work Package 2: Database & Automated data retrieval

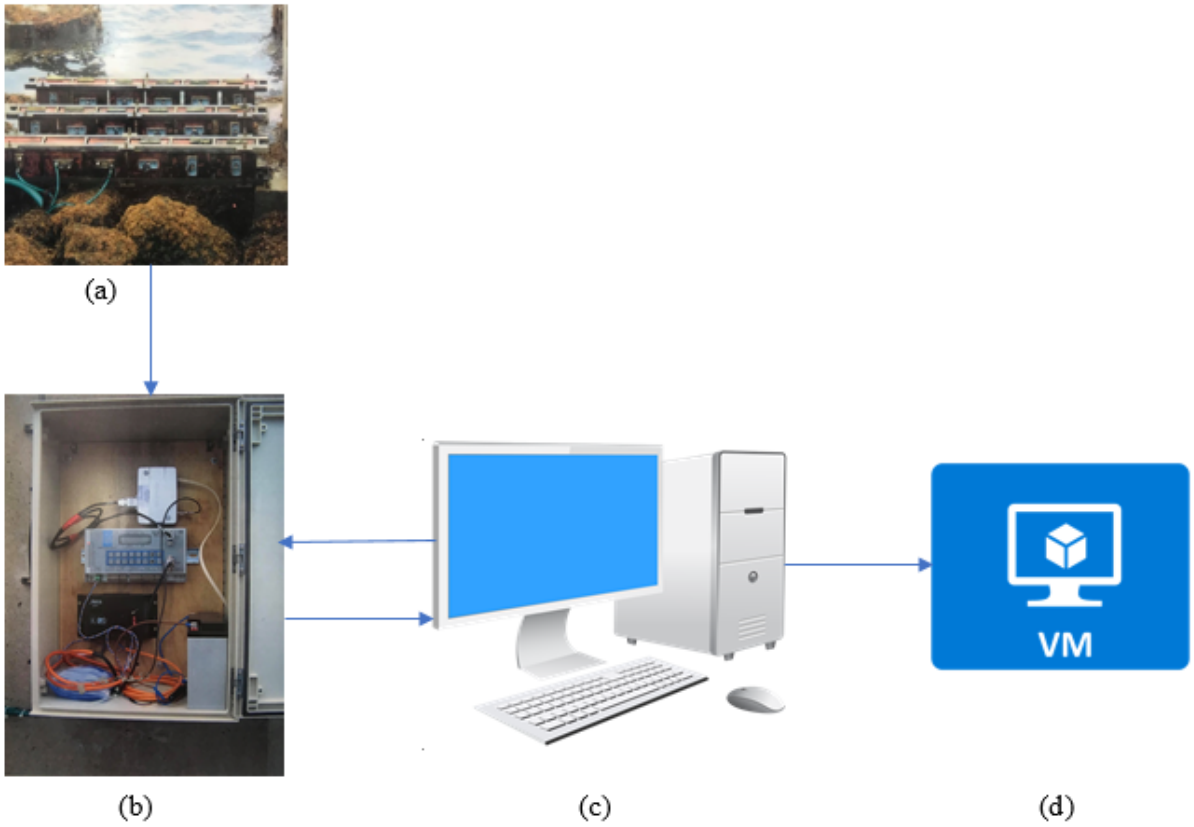


Figure 3.1 - (a) Specimens (b) Data Logger (c) HWU laboratory PC (d) Virtual Machine

Figure 3.1(a) shows specimens which have a water-proofed hard-wired connection to the remote monitoring system. The data logger (show in Figure 3.1 (b)) is in charge of communicating with both the PC and the electrode pairs. Data logger application on a Desktop PC can be used to declare how often the user wants electrode-pair readings to be recorded (i.e. 30 minutes which is the default at the moment) and when to send the dataset to the Desktop PC at Heriot-Watt University in Edinburgh (i.e. on a weekly basis). Automated database runs on the HW Desktop laboratory PC (shown in Figure 3.1 (c)), and the database is then transferred to the Virtual machine (shown in Figure 3.1 (d)) where the web application is run.

Individual data sets arrive to the Desktop PC at the HW laboratory from the Data logger in Dornoch. These files are in a csv (comma-separated values) format, which is a simplified spreadsheet stored as a plain text file. The layout of texts inside CSV files is as follows: each line indicates a different row and each column is separated by a comma. To get information from individual columns, loop and split methods have to be used. However, to make this task much easier, the CSV module will be used to handle this task. No installation is required for the module as it comes with Python and just has to be imported. The files are generated

according to the dates they correspond to. For example, a dataset containing readings from 09/11/2017 would be named as “17110901.CSV” with the year, month then days in respective order.

Database is created to decrease the work load and to increase the efficiency of Work Package 1 programme. This is done by combing all csv files into a singular file to be read, instead of having to read a big array of files separately.

The newly arrived CSV files will be automatically read and copied to the database. As the data being handling is sensitive, the copied CSV files will be moved to a different folder for backup purposes. The folder receiving the data from Data Logger will be empty, thereby allowing us to detect any new additions. The database contains all of the readings from the CSV files in ascending order of date and time.

In this WP, the following procedure were to be followed to create the database:

1. Access
   Folder containing the dataset coming from Dornoch Data Logger is to be opened and listed to ensure correct order of date.
2. Copy
   After opening the CSV file, each row is copied to a new CSV file named “all.csv” till rows are finished.
3. Backup
   The read file is transferred to a backup folder.
4. Repeat
   The next CSV file would need to go through the same process until the folder is empty. The folder is checked again for new uploaded files periodically according to user’s specification.

3.3 Work Package 3: Added extra features

This section is about extra features that will be integrated to the website application for better analysis of results.

3.3.1 Activation Energy

Value of activation energy can be inputted by a user; however, to aid this process, the exact activation energy values need to be calculated and presented to the user for better analysis of results. After all of the values have been calculated according to users' specifications, the activation energy for that particular specimen number and depth can be calculated via the method described in Section 2.3.6. It was deemed appropriate to present four activation energies, calculated using different thermistor values, such that a user can have a full information of this dataset.

3.3.2 Live weather forecast

OpenWeatherMap API provides current weather data and 5 day forecast for over 200,000 cities across the world free out of charge. Data is available in JSON format further decreasing our workload. JavaScript Object Notation (JSON) is a format used to store and transport data as JavaScript object notation syntax in plaintext files only. JSON is considered language independent meaning it can be read and generated in any programming language. However, as the syntax is the same as JavaScript JSON data can be easily converted into native JavaScript objects. Data provided include; temperature, weather conditions, wind and humidity.

3.3.3 Historical temperature readings

Unfortunately, OpenWeatherMap API provides historical data for premium users who pay. To overcome this obstacle, it was decided to create a database that will record the current weather temperature every time thermistor readings are recorded (30 minutes). This data is displayed in the same graph as the thermistor values. There are two popular types of databases used in industry today: Relational databases and Non-relational. The main difference between them is that non-relational databases are collections of files with different types and amounts of data stored in them. These files can be accessed by 'keys' when connecting to the database.

Relational databases are organised with tables where values and types of those values are strictly specified. As the type and volume of information to be stored is known in this case, relational SQL type database is more suitable.

3.3.4 Tidal Readings

Environment Agency Tide Gauge API provides tidal measurements across 44 stations in UK. However, there is no tidal station located in Dornoch. Hence, the measurements from the closest station Aberdeen will be used. These data are stored in an SQL database as well for graphing and can be displayed in the same graph as the thermistor values even though they have different units. This can be done as the range of data is within the same region.

Chapter 4 – Result & Discussion

4.1 Front-end Development

This section completely entails the section of web application that a user can see and interact. These codes responsible for this part are located in the “views” folder. Entirety of the front-end has been developed in CSS and EJS formats.

Two core technologies involved in building Web pages are HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets). HTML is used for describing the structure of a web page such as defining paragraphs and headlines, and embedding images. CSS allows the presentation of the web page to be described, such as colours, fonts and layout designs. It is independent of HTML and can be combined with any mark-up language. The separation between these languages makes it easier to share styles sheets between pages and tailor each page to its own environment. Using CSS instead of just HTML has many benefits. CSS style sheets once created can be reused in multiple HTML pages, this saves valuable time. Global changes to all HTML pages can be made automatically by altering the style sheet. Array of attributes in CSS is much larger than HTML attributes, giving the web page a superior look. This change in look can be observed for the data selector in Figure 4.3.

EJS (Embedded JavaScript) is an effective template engine utilized by node.js as part of the express framework. This approach allows HTML pages to be generated with JavaScript code. There are other templating systems available, however simplicity of logic and syntax makes EJS easy to set up and use. EJS also becomes very useful when html output involves a lot of JavaScript, such as generating dynamic contents (e.g. graphs) and working with real time updates (e.g. live weather forecast).

4.1.1 Head

The file for this section is called Header.ejs and is available with comments in Appendix 1. This part is executed prior to any other parts of the web page.

The head element acts as storage unit for metadata and data related to the HTML document. This section is not displayed for the user. Metadata can be used to define the title of the document, scripts and links.

Figure 4.1 - Browser tab

The “title” element is required by all HTML documents and defines the title for the browser tab which can be seen in Figure 4.1. The “link” element links external style sheets for use. The “script” element loads plot.ly functionality for use by the front-end.

4.1.2 Style Sheet

To determine the style, the script under Main.css was prepared and is available with comments in Appendix 2.

For the background of the website, the same colour pallet used in the Heriot-Watt website has been selected, which is defined as #014c8f in HEX format (shown in Figure 4.2).

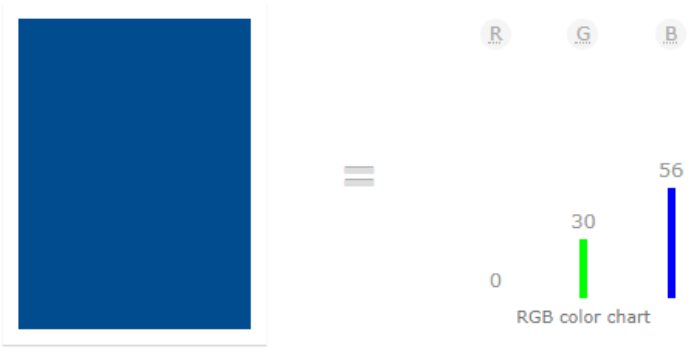


Figure 4.2 - Background colour

The text font has been selected from the Google Fonts catalogue as “Nunito” developed by Vernon Adams, available at https://fonts.google.com/specimen/Nunito.

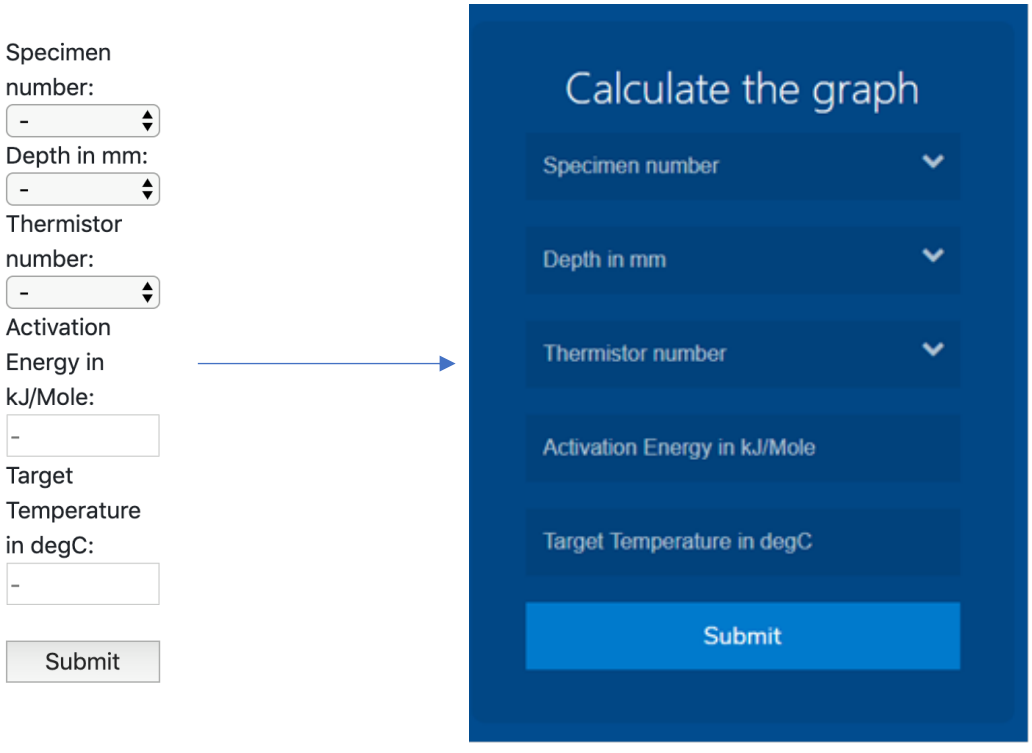
4.1.3 Data Selector

To enable data selection, the script under Calculator.ejs was developed and is available with comments in Appendix 3.

This tool allows a user to specify which data to be analysed. The first three entry fields are presented in the style of a dropdown menu in which the entries were made in accordance with the configuration of test specimens and embedded sensors in the previous research work (listed in Table 4.1). This Toggleable menu allows user to select one value at a time. The activation energy and target temperature are defined as "number" to ensure that text would not be entered to this field as it would not be able to be used in calculations. Once a user has pressed the submit button, these values are then forwarded onto the Back-end for data retrieval. In addition, by clicking the submit button, the web page will scroll down to graph section automatically for user's convenience.

Parameter	Values
Specimen number	1, 2, 3, 4, 5, 6
Depth	5, 10, 15, 20, 30, 40, 50
Thermistor number	1, 2, 3, 4
Activation Energy	Any number
Target Temperature	Any number

Table 4.1 - Permitted values for the parameters



(a) (b)

Figure 4.3 – Data selector (a) with HTML attributes (b) with CSS attributes

4.1.4 Graphing

The script for this section can be found under Home.ejs and is available with comments in Appendix 4.

In total, there are three graphs to be drawn by Plot.ly:

- "resChart" displays the raw and corrected resistivity values against time.
- "tempChart" primarily displays the corrected thermistor values against time. However, weather station temperature readings and tidal data from SQL database can also be displayed by interacting with the legends.
- "logChart" displays a scatter plot of ln ρ against $T_k/1000$ values for activation energy.

To ensure that all the Plot.ly functions run successfully, its' JavaScript functionality must be imported to the front end of the application and this is done by running the script function which calls the latest version of Plot.ly library.

The basic steps of using Plot.ly library in JavaScript were as follows:

- Create variables which hold x and y axis values for a set of points and some other extra meta-information such as title and location;
- Specify the layout and style of the graph: parameters such as name tags, margin size, etc;
- Trigger the plotting function which will render the data with a specified layout.

Arrays of data are received from the back-end and then parsed to the corresponding variables on front end. Due to the nature of the EJS syntax, different types of data had to be parsed in different ways. For example, while arrays of temperature contained decimal values, date and time arrays consisted of string values; however, the string contained commas that could be used as points for splitting the string values into an array.

There are two possible options for charts that can record information on both of the axes: line or scatter charts. Choosing between these two options can be a difficult task as they may look similar especially when scatter chart is drawn with lines. However, the horizontal axis in line charts usually show non-mathematical data, while scatter chart can display non-numeric and numerical values such as date and hours. Also scatter charts are thought to be more ideal for

comparison of numerical values and illustrating the pattern of large data. Hence scatter charts with lines created to connect each data points together was chosen for graphing of “resChart” & “tempChart”. Scatter chart with markers only was chosen for “logChart” to view the trend of the data which can be seen in Figure 4.5.

The sizes of the charts have been declared, so that “resChart” was larger than the other two as that was the primary graph used for analysis.

The scale determines the range of values that appear on the graph’s axes, affecting how data is presented. To scale the graphs efficiently, the data of activated legends were analysed, and the range for the y-axis was set as the maximum and minimum values. By disabling other legend(s), the range can be altered to view that data in more detail, demonstrated below in Figure 4.4.

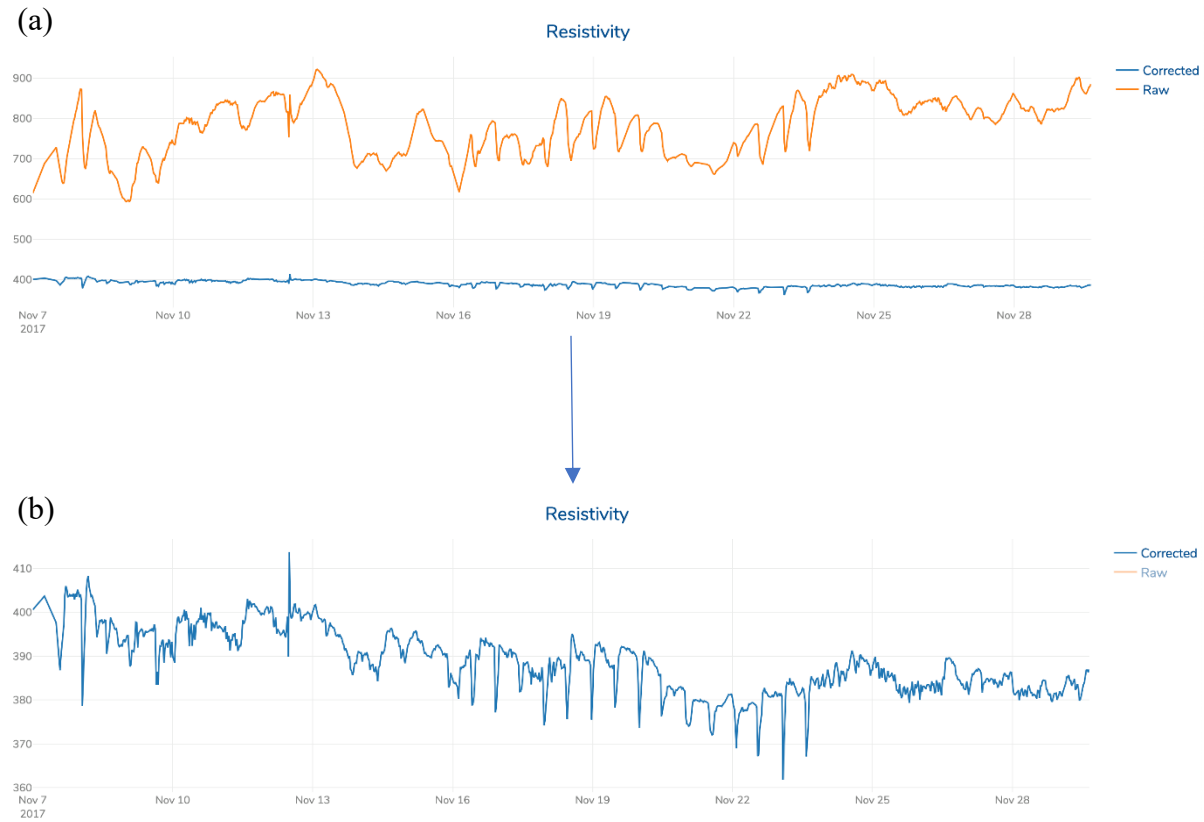


Figure 4.4 - (a) Both legends activated (b) Single legend activated

For the resistivity graph, both of the legends were activated by default to show the user the difference between the raw and correct resistivity. For the other graphs, only the selected thermistor graph, were made visible; however values for other thermistors could also be viewed on the same graph by interacting with the legends.

Time series with a range slider and selectors was added to allow a user to choose which section of the graph to display. At the moment, developing the synchronization of sliders was not possible as *Plotly.rangeSlider* function clashed with synchronization script. This bug had been reported to Plot.ly for fixing. Another feature included for the graphs was the capability to hover over data points on charts to view their actual values. Both of these features are demonstrated in Figure 4.6.

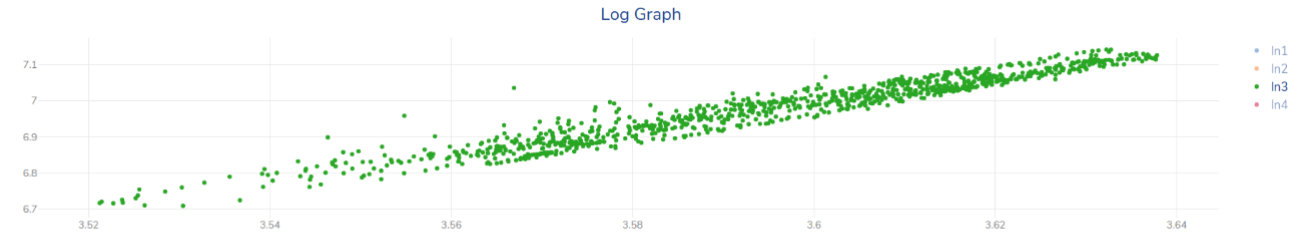


Figure 4.6 - Graph for Activation Energy

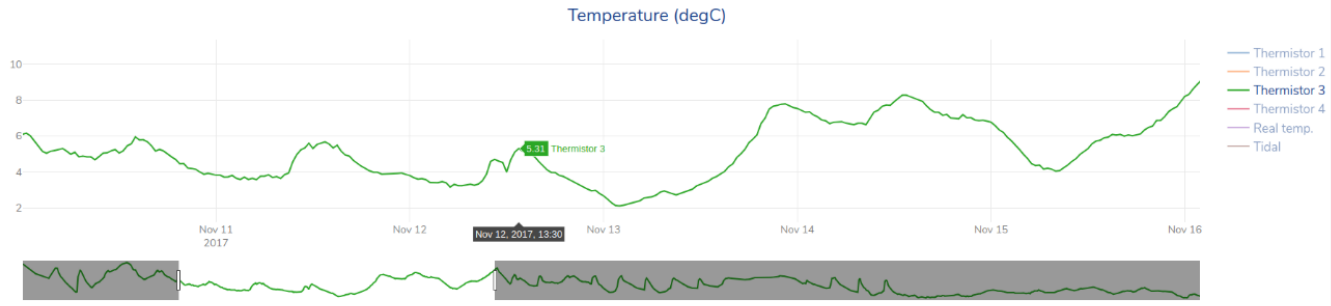


Figure 4.5 - Thermistor graph being interacted with

4.1.5 Weather Bar

Figure 4.7 - Weather bar

For this section, the Weather.ejs worked with Weather.js which sent a request to OpenWeatherMap and after a successful response, it parsed the data and outputted it to the screen. Both files are available in Appendix 5 & 6 with comments. The location of the site has been declared as Dornoch, GB with latitude of 57.88 and longitude of - 4.03.

API request is sent to retrieve the following live and forecast parameters from the weather station:

- Weather conditions
- Humidity
- Wind
- Current Date
- Temperature
- High/Low Temperature

The API provides 9 different weather conditions and indicates day/night cycle;

- Clear sky
- Few clouds
- Scattered clouds
- Broken clouds
- Shower rain
- Rain
- Thunderstorm
- Snow
- Mist

The weather conditions are indicated with icons designed by Erik Flowers and is available for free at http://erikflowers.github.io/weather-icons/.

The layout for the weather section is divided into 4 panels, shown in Figure 4.7:

- Left panel displays the location of the site and the current date.
- Centre panel displays the live weather condition in text and icon formats and the temperature.
- Right panel displays wind, humidity and low/high temperature values.
- Below these panels, forecast for the next six days are displayed by listing the name of the day, weather conditions via icons and low/high temperature predictions.

4.1.6 Activation Energy

The script for this section is available with comments in Appendix 7 and is called Activation-energy.ejs.

Figure 4.8 - Fields for displaying calculated Activation Energies

According to the user's selection of specimen number and depth, the activation energy for four thermistors were calculated in the backend and passed to the front end. These values were displayed under the weather bar (seen in Figure 4.8).

4.1.7 Navigation Bar

The file for this section is called Navigation.ejs and is available with comments in Appendix 8.

Figure 4.9 - Navigation bar

To make the navigation through the website quicker for the user, two elements have been added to the bar which can be seen in Figure 4.9:

- Home: this brings the user back to the main page
- About: this section will direct user to a new domain with "/about" link where About section is displayed

4.1.8 Footer

Figure 4.10 - Footer

The file for this section is called footer.ejs and is available with comments in Appendix 9.

This section was created for aesthetic purposes to enhance the user experience. This website has been developed for use by HWU staff, and hence university logo is displayed to indicate this. The development year and copyright symbol are displayed as well (shown in Figure 4.10). This acts as an easy method of protection against web plagiarism.

4.1.9 About

The file for this section is called about.ejs and is available with comments in Appendix 10.

The about section was created to eliminate the need to refer back to research papers to see the mathematical formulae used in correcting the resistance readings. The page is presented below in Figure 4.11:

Figure 4.11 - About page

4.2 Back-end Development

4.2.1 Automated Database

This section of the development is separate from the web application and is developed on Python3. The file for this section is called Script.py and is available with comments in Appendix 11.

To be able to execute the procedures mentioned in “Work Package 2”, it was necessary to import three libraries:

- “os” which allows the program to interact with the underlying operating system. The primary use in this case is to access the name of files and move data across directories. Used in “Access” and “Backup” section of the procedure.
- “csv” which allows Python to read and write the csv files. Used in the “Copy” section of the procedure.
- “time” which provides time-related functions for use, allowing a timer to be set to run the programme on a loop to check for new incoming files. Used in “Repeat” section of the procedure.

In addition to this, five variables were also needed to be declared:

- readDirectory corresponds to the folder that is receiving new data regularly from the Data Logger Application
- writeDirectory corresponds to where the database will be stored
- backupDirectory corresponds to where the files are moved and kept
- writeFileName is the name given to the database
- timeSleep is the time interval in seconds between the programme re-running.

Data Logger application allows the user to determine the time period between data retrievals. If the user wants to run the app every week then timeSleep can be mathematically defined as 7*24*60*60. Every time data retrieval period is changed in data logger app, the database script will have to be changed manually and ran again. However, computers have very strong computational power and this code can be run every second without slowing the computer down. Hence, defining data retrieval period once in Data Logger application is sufficient and programme requires no further interaction after it has been set up. This also allows users to

manually upload files such as historical data to readDirectory and have it uploaded to the database instantly.

The programme initially checks if the declared readDirectory file path is correct, if not error is displayed asking user to specify the path properly.

The programme consists of an infinite loop which checks the contents of the readDirectory. From the time module, time.sleep(sec) function is called which for the given number of seconds will suspend the execution of the programme when folder has been emptied out.

If the directory is not empty, all the files are taken and ordered in ascending order. In that order every file is iterated to copy the contents of the csv file row by row and moved to the backup directory after the last row has been copied.

To verify that the programme works, data is uploaded into the readDirectory in multiple instances checking in between if the database has been successfully updated and files transferred to backupDirectory. Result of the test was successful and is displayed below in Figure 4.12:

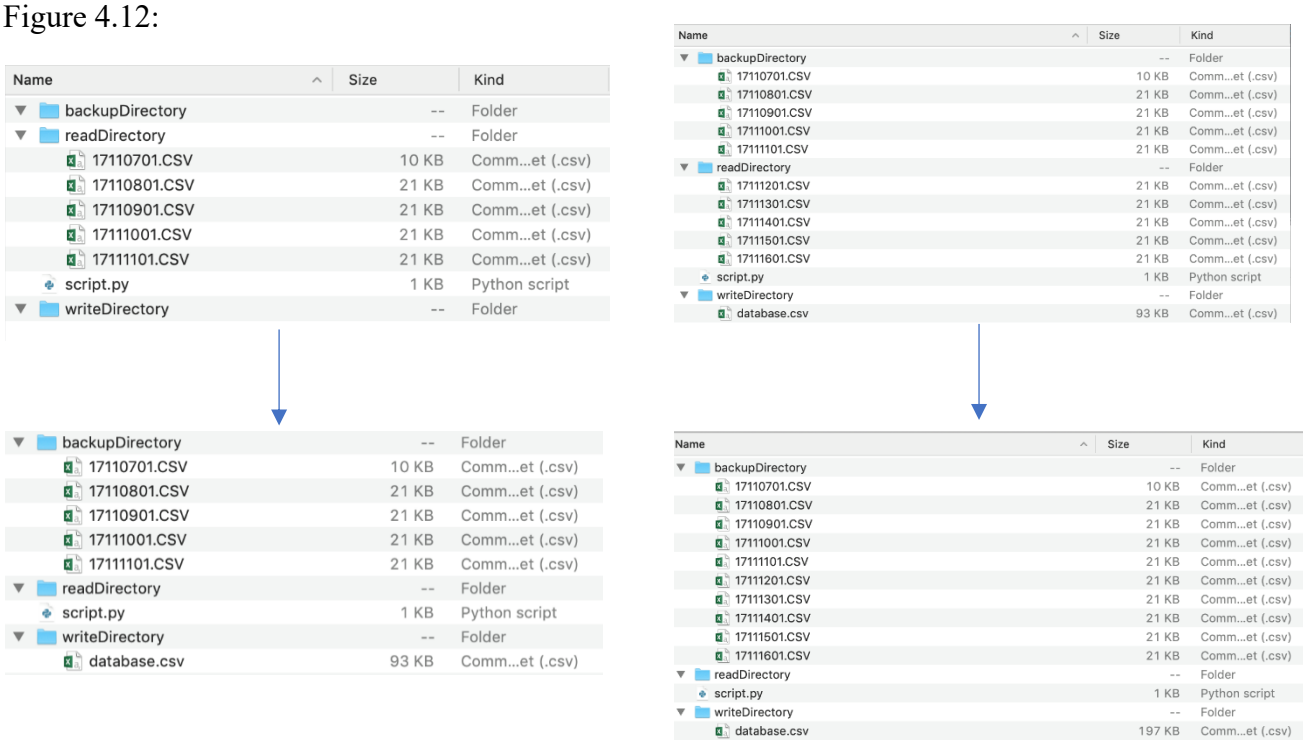


Figure 4.12 - CSV files fed in two different instances

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	07/11/2017	00:00:00	65641	83791	43570	29203	24319	29775	17893	9802	9793	9895	9571	101422	81345	93396	76083	75629	58100	68338
13	07/11/2017	18:30:00	78453	90447	50291	34340	28816	34914	20947	12079	11977	12038	11592	112631	96845	108727	88945	87830	67573	78076
23	07/11/2017	23:30:00	90305	102720	57210	39142	32809	39759	23811	13833	13757	13829	13358	130874	110977	124773	101819	100539	77424	89376
24	08/11/2017	00:00:00	92477	104118	58313	40030	33628	40746	24418	14176	14127	14221	13755	134764	113947	128175	104593	103172	79698	91884
40	08/11/2017	08:00:00	89770	99625	56958	39543	33564	40777	24559	13875	14094	14307	13914	131421	114038	127600	104555	103192	80163	92097
71	08/11/2017	23:30:00	61894	81188	42538	28650	23846	29218	17538	9492	9603	9696	9403	92307	78458	90838	74121	73701	56748	66835
72	09/11/2017	00:00:00	61741	81111	42422	28525	23722	29068	17429	9467	9549	9627	9333	92250	78170	90511	73821	73308	56391	66417
91	09/11/2017	09:30:00	72491	86803	47779	32589	27441	33476	20138	11114	11276	11419	11099	104553	90899	106288	86465	86160	66220	78467
119	09/11/2017	23:30:00	77781	80285	47657	33621	28897	34669	20992	12321	12324	12401	11932	98210	93856	104988	86718	85520	66287	75966
120	10/11/2017	00:00:00	76763	78795	46946	33242	28685	34427	20887	12183	12247	12359	11919	96214	92707	103631	85874	84731	65921	75556
141	10/11/2017	10:30:00	79747	82778	49262	34921	30163	36264	21967	12613	12814	12942	12523	103812	98126	109922	91192	89849	69901	80268
167	10/11/2017	23:30:00	86855	89823	53467	37938	32662	39186	23698	13942	14009	14103	13619	113772	107156	120377	99527	98058	76304	87203
168	11/11/2017	00:00:00	86942	89805	53452	37905	32624	39150	23693	13956	13971	14070	13605	113785	107017	120030	99306	97842	76125	87072
191	11/11/2017	11:30:00	74760	82371	48223	33705	28967	35177	21246	11839	12005	12180	11870	100739	94287	107457	89187	88299	69042	79775
215	11/11/2017	23:30:00	86304	98517	55871	38664	32633	39605	23725	13598	13656	13734	13309	119733	106441	122707	100597	99240	76794	89158
216	12/11/2017	00:00:00	86512	98816	56021	38761	32699	39683	23766	13632	13687	13755	13326	120349	106897	123139	101109	99773	77082	89472
242	12/11/2017	13:00:00	79411	90697	51803	35933	30494	37067	22251	12501	12664	12787	12401	115785	100021	115821	95267	94535	73161	85202
263	12/11/2017	23:30:00	92003	98989	57848	40606	34589	41683	25064	14743	14725	14782	14297	132800	113314	128986	106311	104737	81244	93050
264	13/11/2017	00:00:00	92548	99653	58209	40849	34789	41949	25215	14824	14814	14868	14386	133638	114081	129925	107016	105412	81743	93792
283	13/11/2017	09:30:00	89681	94204	56111	39769	34213	41232	24845	14443	14527	14648	14213	123113	109819	124508	103222	101800	79415	91026
311	13/11/2017	23:30:00	68454	73883	44556	31535	27028	32576	19695	11298	11371	11438	11021	91814	84123	95925	79641	78678	61207	70367
312	14/11/2017	00:00:00	68649	74259	44700	31635	27108	32688	19762	11323	11408	11477	11067	92038	84305	96085	79838	78914	61421	70663
339	14/11/2017	13:30:00	64869	73029	43242	30311	25853	31369	18894	10518	10663	10762	10409	89403	80983	93282	77502	76644	59641	69167
359	14/11/2017	23:30:00	70757	76609	45957	32448	27780	33540	20251	11560	11681	11762	11352	100424	87942	101085	83859	83031	64416	74372
360	15/11/2017	00:00:00	71086	76907	46143	32560	27878	33632	20307	11617	11722	11795	11381	100912	88287	101429	84153	83248	64600	74557
384	15/11/2017	12:00:00	77426	84994	50485	35633	30514	36970	22312	12632	12814	12953	12583	111435	96344	111295	92549	91666	71377	82710
407	15/11/2017	23:30:00	68983	81908	46730	32406	27344	33391	20087	11072	11245	11365	11035	103062	87657	102685	84914	84419	65272	76503
408	16/11/2017	00:00:00	68311	81269	46351	32134	27106	33107	19910	10972	11146	11254	10933	102356	86902	101809	84121	83771	64754	75877
432	16/11/2017	12:00:00	70832	81955	47364	32914	27915	34017	20462	11346	11528	11644	11298	103294	90108	105990	87365	86992	67315	78805
455	16/11/2017	23:30:00	80551	96916	53706	36818	30825	37756	22597	12607	12739	12831	12484	120106	102024	121512	99355	99005	75849	89881
456																				
457																				

Figure 4.13 - Database, opened in excel

The csv files were successfully combined into a single file and can be observed in Figure 4.13. Only 3 rows per day are displayed, this is done to show that all days were added in the database in the correct order.

4.2.2 Graphing

The file for this section is called app.js and is available with comments in Appendix 12.

To be able to execute the procedures mentioned in "Work Package 1", it was necessary to import the following libraries:

- "express" is the web framework for Node.js. Used for managing data transfer between the front-end and back-end.
- "body-parser" is used to parse the body of the incoming requests as they are not nicely formatted and easily readable by default in node.js. Supports JSON files.
- "path" module is used for working with file and directory paths.
- "fast-csv" provides utilities for parsing and formatting CSV files
- "node-schedule" allows for jobs to be scheduled for execution at specific intervals. This will be used when SQL database is created for the real temperature and tidal readings.
- "http" is a built-in module that enables node to transfer data over the Hyper Text Transfer Protocol (HTTP). This is used to call for the weather and tidal APIs.
- "fs" module enables node to access local files.

When the website is first launched, the backend will send empty values for all the variables to prevent Plot.ly failing.

The function fs.createReadStream() reads the database sequentially from the path file declared. The programme uses a pipe system meaning that the data is read row by row.

```
07/11/2017,00:00:00,65641,83791,43570,29203,24319,29775,17893,98
02,9793,9895,9571,101422,81345,93396,76083,75629,58100,68338,998
8,10149,10019,10060,14757,9962,6894,6045,4949,7558,5676,9905,985
7,10179,10182,51619,461935,68393,43083,28805,27566,21082,10122,1
0053,9856,9998,37874,44748,65700,49198,74314,61651,68636,9610,95
93,9721,9759,10181,11621,14442,12420,14379,11346,13789,9940,8709
,9843,9709,8.3,3858,-3,12.53,
```

Figure 4.14 - Single row of a CSV file

It is important to understand the layout of the CSV file as the programme will need instructions on which columns to extract readings from. The first column contains date and second contains time. There are a total of six specimens, and the first seven columns represent the resistance readings at different depth and the next four store the resistance readings from the thermistors. Therefore, each specimen consists of eleven columns. A single CSV row is displayed in Figure

4.14, *italics* represent resistance readings at different depth and underlines represent different thermistor readings.

To calculate the location of the thermistor readings requested by a user, the following structure was prepared. First, two columns were skipped as they represented the date and time values. Next, the programme had to determine how many columns to skip to reach the specified specimen number. This was done by subtracting 1 from the users declared specimen number and multiplying it by 11. To reach the thermistor readings, seven was added to skip over the depth values. Lastly, the specified thermistor number was added. The line for this process was displayed below in Figure 4.15. Programming languages count from 0 hence 1 was added to skip over the first two columns.

```
var tmp = data[1 + 10*(parseInt(req.body.specimen) - 1) + 7 + parseInt(req.body.thermistor)];
```

Figure 4.15 - Line for reading specified thermistor values

Following this, the raw thermistor readings could be acquired and corrected via the Steinhart-Hart equation (Eq. 2). These values were pushed to the front-end for the second graph to be displayed. However, the other thermistor readings were corrected thereby allowing user to view them by interacting with the legends.

Similar procedure was followed to calculate the location of the resistance readings at the requested specimen number and depth, displayed below in Figure 4.16:

```
var rd = data[1 + 11*(parseInt(req.body.specimen) - 1) + parseInt(req.body.depth)];
```

Figure 4.16 - line for reading specified resistance at different depth

Once the data were retrieved, Equation 1 was applied to convert the measured resistance to resistivity. Arrhenius relationship (Eq. 4) was then used to calculate the corrected resistivity values and these values were pushed to the front-end for the first graph to be displayed.

Lastly, the natural logarithm of the raw resistivity and 1000 divided by thermistor values were calculated for displaying the third graph.

After all the rows were read and values calculated, the activation energy could be then calculated using the Pearson product-moment correlation (Eq. 6 & 7) and pushed to front-end for display.

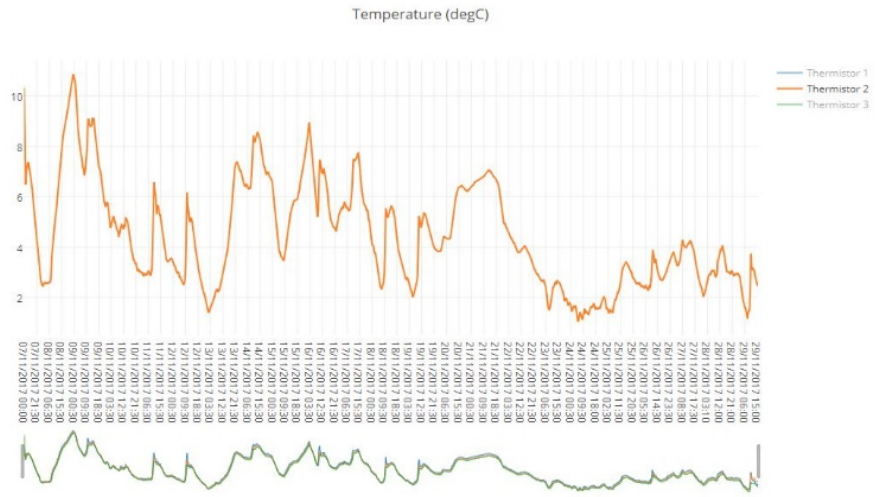


Figure 4.17 - Faulty time-series

The date values were presented in the CSV files as day/month/year. When presented in this format, the Plotl.ly plotting function did not adjust properly to a time-series plot with the range selector (shown in Figure 4.17). To fix this issue, the format was converted to year/month/day. The time value was in the correct format for Plot.ly, thus it could be pushed to the front-end without further process.

There are two instances where the weather data from station is used:
- The cron job where current temperature value is recorded in SQL database
- Current temperature sent to the front-end for display in the weather bar

Since relational database has been chosen to store the data, the Standard Query Language (SQL) will be used to communicate with the database. Overall two queries are used one to read the data from the database:

```
select = 'SELECT date, temp FROM temp ORDER BY date;';
```

Figure 4.18 - Select statement

SELECT statement was used to get data for columns date and temp from the temperature table.

Another step to write into the database:

```
var insert = 'INSERT INTO temp (date, temp) VALUES (\''+date+'\', \''+temp+'\');';
```

Figure 4.19 - Insert statement

INSERT statement was used to push data for columns date and temp into the temperature table.

The syntax of queries must follow SQL conventions. These queries were executed using pool object which maintained an open connection to the database. Same method was be applied to access and save Tidal data as well.

4.3 Launch

Throughout the development of the application, Heroku has been used to get the application quickly online for testing purposes without needing to deal with the infrastructure. Once the website was fully developed, it could be run on the platform provided by the IT department at Heriot-Watt University (HWU).

WinSCP (Windows Secure Copy) is an open-source file transfer tool for Windows. Its main functionality is to establish file transfer between a local and a remote computer. The data logger application is in the local windows system and script.py for automated database is in the remote computer. WinSCP can be scripted (shown in Figure 4.20) to automate the synchronization of remote SFTP directory with the local directory in a bi-directional upload and download meaning changes in any directory is applied to both.

```
option batch abort
option confirm off
open sftp://vagif:vz8zFv3U@137.195.70.233 -hostkey="ssh-rsa 2048
05:26:b3:69:be:63:a3:ed:d3:46:eb:5a:3c:d9:12:40"
synchronize both C:\Users\labuser\Desktop\DornochDatabase /home/vagif/DornochDatabase
put -delete C:\Users\labuser\Desktop\DornochDatabase\readDirectory\* /home/vagif/backup2/
exit
```

Figure 4.20 – SyncToBiDirectionalScript.txt

A second safety step has been implemented to copy the contents from the readDirectory and is stored in the remote server in addition to the back-up created on the local system.

Next, a windows batch file is created (shown in Figure 4.21) which will be used by Windows Task Scheduler to create a simple task to run the batch file on constant basis to ensure both directories are always up-to-date.

```
path C:\Program Files (x86)\WinSCP\
WinSCP /script=SyncToBiDirectionalScript.txt
pause
```

Figure 4.21 - SyncToBiDirectionalScript.bat

Now that the CSV files from the data logger can be accessed in the remote server, script.py is run in the background to ensure the database is updated constantly.

PuTTy is an open-source terminal emulator which is used to open an SSH tunnel to the bash command prompt on the VM. The bash command shell can be accessed by typing in the Server ID, username and password details. All of the elements of the platform can be retrieved through the GitHub repository created at the start of the development with a single command, shown below in Figure 4.22:

```
$git clone git clone https://github.com/vagifaliyev/websiteProject htdocs
```

Figure 4.22 - Pulling the latest version of the platform from GitHub

Now that all necessary components have been created for the platform, the application can be hosted by typing node App.js at the command prompt, however it will run only as long as there is a connected session. It will terminate as soon as the session is closed. To ensure that the application runs continuously, forever command will be used (show in Figure 4.23).

```
$forever start app.js
```

Figure 4.23 - Launch the website

The web page can be accessed on Heriot-Watt network at: http://137.195.70.233/

4.4 Test

This section is to prove that the system created for the platform works and all of the aims and objectives have been met. Full version of the website is available for view in Appendix 13.

Windows Task Scheduler is running the batch file for bi-synchronisation of local and remote folders successfully, shown below in Figure 4.24:

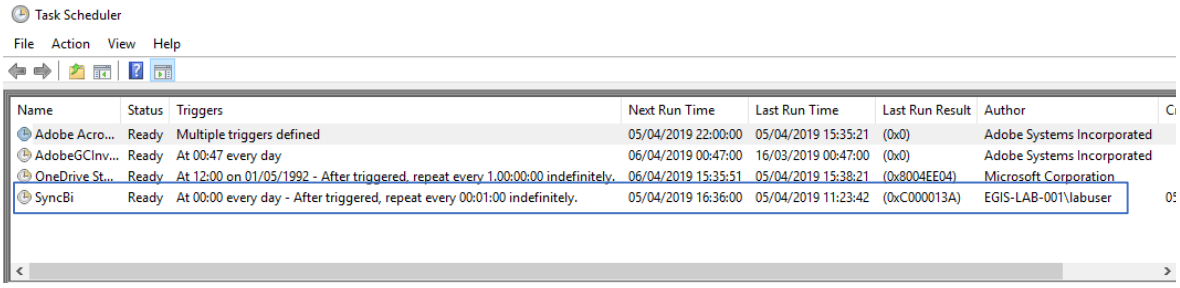


Figure 4.24 - Task Scheduler

Total of 23 CSV files were provided by the Dissertation supervisor, these data are uploaded to the *readDirectory* in 3 different instances to ensure that the platform displays the data without human intervention. The operation is a success and is displayed below in Figure 4.25. Hence, objectives described in WP2 for the automated database have been meet.

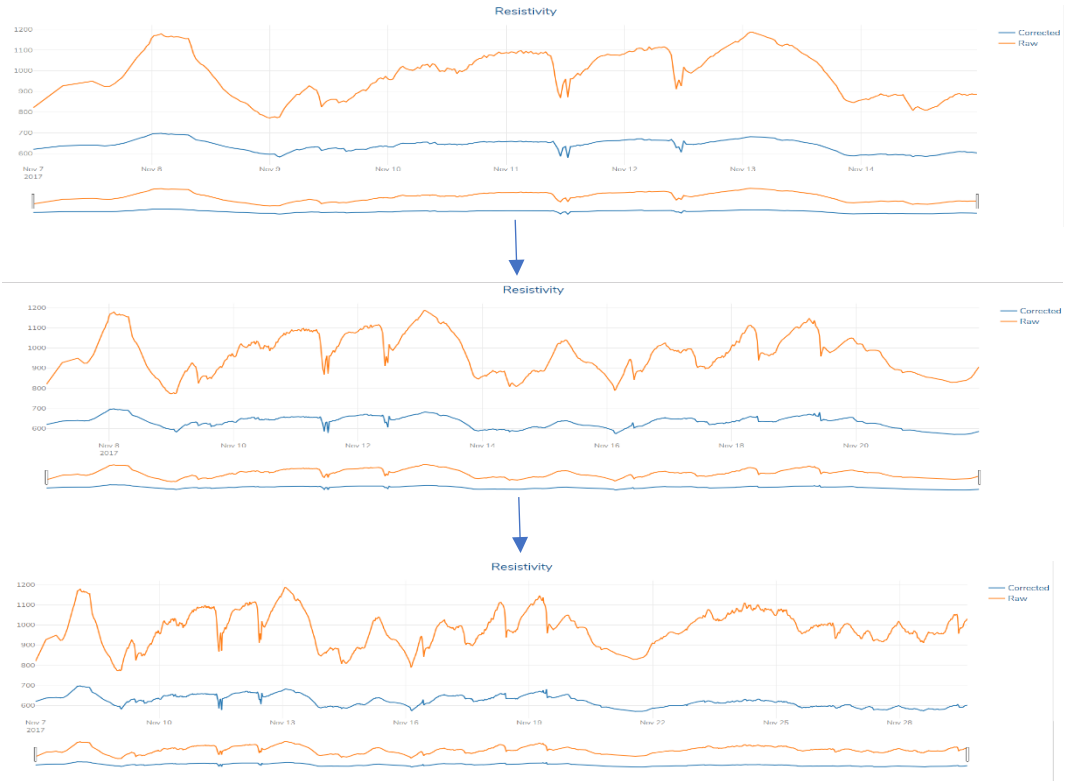


Figure 4.25 - Graphs automatically being updated with new data

To ensure that the programme is correcting and graphing the data correctly, the website graphs are compared with Excel graphs. Figure 4.26 shows the entries for the fields in the data selector.

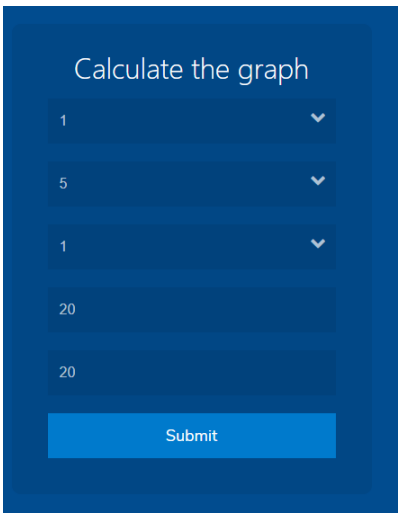


Figure 4.26 - Data selector

From the comparison with Excel graph and website graphs (shown in Figure 4.27 & 4.28), it can be concluded that the protocols for correction have been applied successfully.

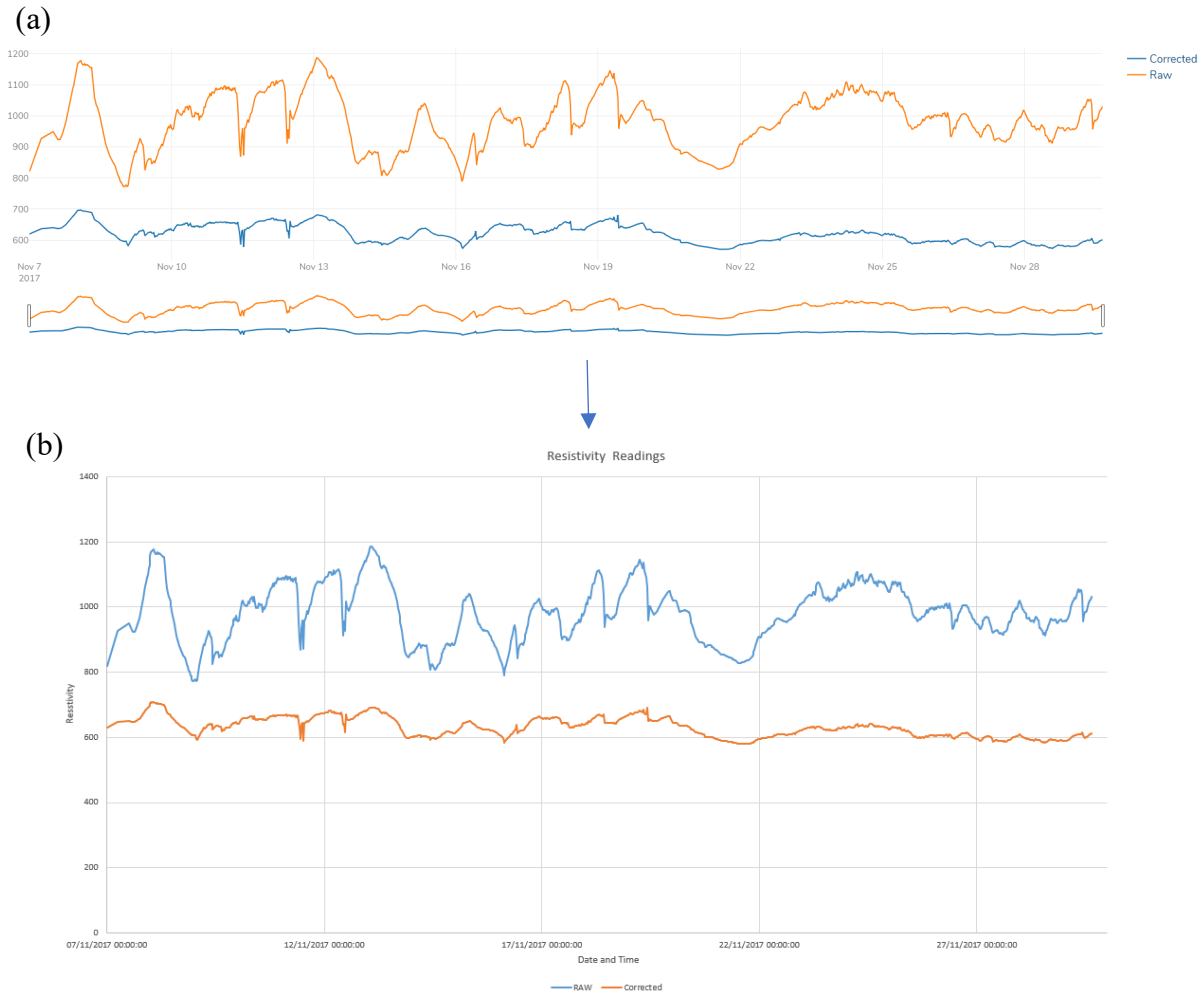


Figure 4.27 - (a) website resistivity graph (b) excel resistivity graph

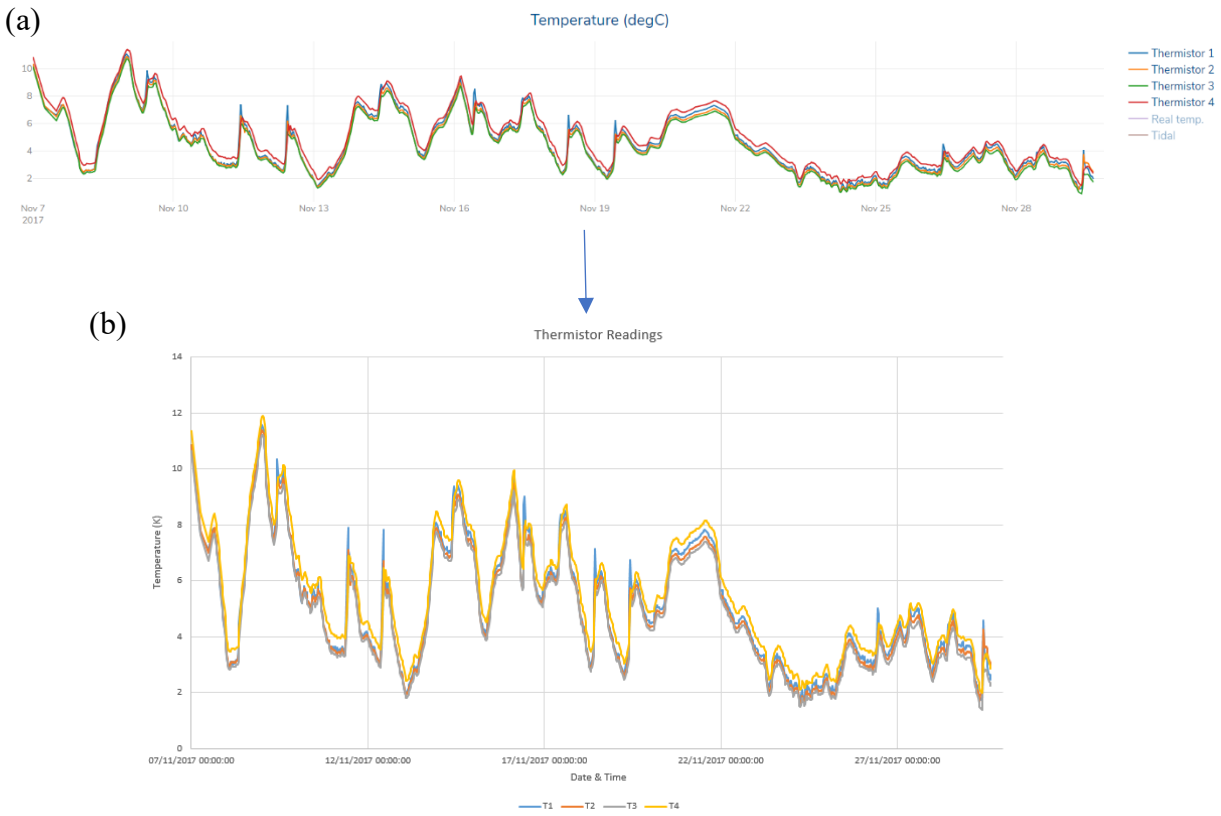


Figure 4.28 - (a) website thermistor graph (b) excel thermistor graph

Hence, the objectives mentioned in Work Package 1 have been met.

On the main page of the website, the current and six day forecast weather conditions and parameters are displayed (shown in Figure 4.29).

Figure 4.29 - weather bar

The activation energy is displayed below the weather bar, shown in Figure 4.30.

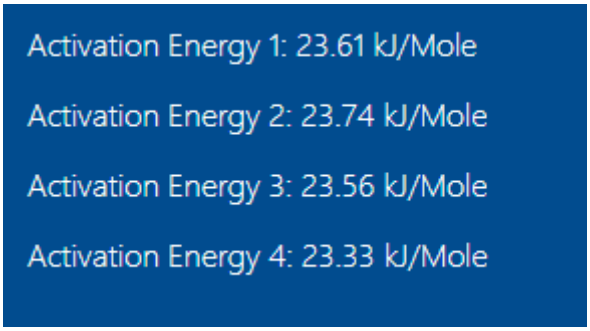


Figure 4.30 - Activation Energy

The historical and tidal data from the nearby stations are being stored in the SQL database and are displayed in Figure 4.31. Hence all the extra features described in Work Package 3 have been developed successfully.

(Note: the data starts from the time website was launched).

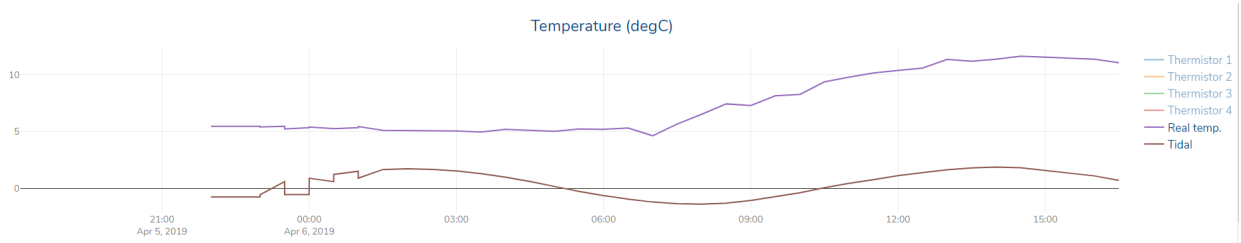


Figure 4.31 - Temperature and tidal readings displayed

Figure 4.32 - Temperature and tidal readings displayed

Chapter 5 – Conclusion

5.1 Development

All of the aims and objectives laid out in Section 1.2 have been met and achieved. After successfully developing an alpha version of the application that met all the objectives, to enhance the user experience Embedded JavaScript (EJS) and Cascading Style Sheets (CSS) were applied to the website.

Web application has been primary developed using Node. To determine the dataset the user wants displayed, a data selector bar was created. Appropriate protocols were applied to data in accordance to users entry and have been confirmed with manually created excel file. Data visualization was achieved with Plotly and various interactive elements were added to increase clarity for better analysis.

For the data arriving regularly from Dornoch site via the Data Logger, a script was developed in Python 3 to automatically combine all of the separate files into a single file to create a database.

A weather application displaying the current weather conditions and parameters with 6 day forecast was created using Node to give the user a better understanding of the on-site conditions. Current weather conditions from a nearby weather station were accessed via OpenWeatherMap API. The tidal data from the Environment Agency was accessed via Environment Agency Tide Gauge API. The incoming data from the APIs were inserted to a database created via PostgreSQL. The activation energies for the specimens were calculated using the protocol described in Section 2.3.6 and are displayed under weather application.

The web application has been launched on the Heriot-watt university server due to the sensitivity of data and can only be accessed on the campus network at: http://137.195.70.233/

5.2 Comparison

Excel requires a significant amount of manual effort to correct and graph all of the data. There are six specimens with seven different depths and four thermistors, this much data results in 168 different possibilities for corrected resistivity values for the resistivity graph. Excel also does not have a tool to enable the user to pull a specified graph for quick analysis.

With huge data volumes being represented in Excel, it becomes difficult to analyse it and may lead to misinterpretations. However, the web application has tools implemented such as range slider that allows user to easily choose dates to investigate and interactive legends to disable other elements for closer inspection of a particular element.

Another drawback of Excel is that it doesn't show data in real time, all the new data received from Data Logger has to be entered manually. With all that manual entry comes the risk of human error. As the amount of data in the spreadsheet grows, so does the chance of formula typos and errors as the fields were filled by copy and pasting.

Collaboration with others is more challenging with Excel, as it has to be continuously updated manually, since it is an installed application, not a cloud-based application like the platform developed.

References

McCarter, W. J., Emerson, M., and Ezirim, H. (1996). “Properties of concretein the cover zone: Water penetration, sorptivity and ionic ingress.”Mag. Concr. Res., 48(176), 149–156.

W. J. McCarter; T. M. Chrisp; G. Starrs; A. Adamson; E. Owens; P. A. M. Basheer; S. V. Nanukuttan; S. Srinivasan; and N. Holmes (2012). “Developments in Performance Monitoring of Concrete Exposed to Extreme Environments”

McCarter,W. J., and Brousseau, R. (1990). “The A.C. response of hardenedcement paste.” Cem. Concr. Res., 20(6), 891–900.

British Standards Institution (BSI). (2000b). “Concrete: Specification, performance,production and conformity.” BS EN 206-1, London.

Jaehwan Kim, W. John McCarter, Benny Suryanto. (2018). “Performance assessment of reinforced concrete after long-term exposure to a marine environment”

British Standards Institution (BSI). (2006a). “Concrete—Complementary British standard to BS EN 206-1,–Part 1: Method of specifying and guidance for the specifer.” BS 8500-1, London.

Chrisp, T., Starrs, G., McCarter, W., Owens, E., Nanukuttan, S., Holmes, N., Basheer, L.: Developments in Intelligent Monitoring ofConcrete Structures. Structural Faults & Repairs Conference, Edinburgh, Scotland, June, 2010.

Holmes, N., Basheer, L., Nanukuttan, S., Srinivasan, S., Basheer, P., McCarter, J., Chrisp, M.: Development of a New Marine Exposure Site on the Atlantic North-West Coast of Ireland. BCRI, Cork, Ireland, 2010.

3leafnodes.com. (2017). Plotly: Getting Started and First Impressions | 3LeafNodes.com. [online] Available at: http://www.3leafnodes.com/plotly-getting-started [Accessed 25 Mar. 2019].

Docs.npmjs.com. (2019). *npm Documentation*. [online] Available at: https://docs.npmjs.com/about-npm/ [Accessed 25 Mar. 2019].

Mills, C. (2019). *Express/Node introduction*. [online] MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction [Accessed 25 Mar. 2019].

Openweathermap.org. (n.d.). OpenWeatherMap API guide - OpenWeatherMap. [online] Available at: https://openweathermap.org/guide [Accessed 26 Mar. 2019].

W3.org. (2014). *Open Web Platform Milestone Achieved with HTML5 Recommendation*. [online] Available at: https://www.w3.org/2014/10/html5-rec.html.en [Accessed 11 Mar. 2019].

Wodehouse, C. (n.d.). [online] UPWORK. Available at: https://www.upwork.com/hiring/development/a-beginners-guide-to-back-end-development/ [Accessed 31 Mar. 2019].

Node.js, F. (n.d.). Node.js. [online] Node.js. Available at: https://nodejs.org/en/ [Accessed 2 Mar. 2019].

Costa, F. (2019). Node.js: the JavaScript platform used by Netflix and Uber - Xpand IT. [online] Xpand IT. Available at: https://www.xpand-it.com/2019/03/21/node-js-javascript-platform/ [Accessed 7 Mar. 2019].

Voss, L. (2018). This year in JavaScript: 2018 in review and npm's predictions for 2019. [online] The npm Blog. Available at: https://blog.npmjs.org/post/180868064080/this-year-in-javascript-2018-in-review-and-npms [Accessed 14 Mar. 2019].

Chrisp, T., Starrs, G., McCarter, W., Owens, E. and Nanukuttan, S. (2010). Developments in Intelligent Monitoring of Concrete Structures. [online] Available at: https://arrow.dit.ie/cgi/viewcontent.cgi?article=1012&context=engschcivcon [Accessed 15 Mar. 2019].

Environment.data.gov.uk. (2019). Real-time API reference. [online] Available at: https://environment.data.gov.uk/flood-monitoring/doc/reference [Accessed 7 Mar. 2019].

Appendix

1. Header.ejs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Vagif Aliyev</title>  <!-- Name for Browser Bar -->
    <script src="https://cdn.plot.ly/plotly-1.8.0.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.6.0/Chart.min.js"></script>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
      crossorigin="anonymous">
    <link rel="icon" href="https://png.icons8.com/ios/50/000000/combo-chart.png">
<!-- Icon for browser bar-->
    <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">
    <link rel="stylesheet" href="../css/main.css">
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-
awesome.min.css"
    rel="stylesheet">
    <link rel="stylesheet"
        href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0.0-
alpha.6/css/bootstrap.min.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-
datepicker/1.8.0/css/bootstrap-datepicker.min.css">

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/weather-
icons/2.0.9/css/weather-icons.css">
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-
datepicker/1.8.0/js/bootstrap-datepicker.min.js"></script>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script> <!-- call for
latest plotly library -->

  </head>

  <body>
```

2. Main.csss

```
body {
  background-size: cover;
  max-width: 100%;
  font-family: 'Nunito', sans-serif;
```

```
  font-weight: 300;
  background: #014c8f;
}

.wrapper {
  width: 100%;
  height: 100vh;
  background: #014c8f;
}

.about {
  width: 100%;
  height: 100%;
  background: #014c8f;
}


/* --------------header styles start------------------ */

.header {
  padding-top: 20px;
}

.header__logo {
  color: #fff;
  text-decoration: none;
  font-size: 24px;
  font-weight: 400;
  font-family: 'Nunito', sans-serif;
}

.header__logo:hover {
  color: #fff;
  text-decoration: none;
}

.header__nav {
  display: flex;
  align-items: center;
  margin-left: auto;
}

.header__nav-item {
  color: #fff;
  text-decoration: none;
  font-size: 20px;
  padding: 10px 15px;
  position: relative;
}

.header__nav-item::after {
```

```
  content: "";
  display: block;
  width: 0%;
  border-bottom: solid 1px #fff;
  transition: all .33s;
  left: 0;
  right: 0;
  margin: auto;
  position: absolute;
}

.header__nav-item:hover {
  color: #fff;
  text-decoration: none;
}

.header__nav-item:hover::after {
  width: 60%;
}

/* --------------header styles end------------------- */
/* --------------main styles start------------------- */

.main {
  padding-top: 80px;
}

/* --------------main styles end------------------- */
/* --------------calculator styles start------------------- */
.calculator-card {
  width: 400px;
  background: rgba(0, 0, 0, 0.07);
  border-radius: 10px;
  display: flex;
  flex-direction: column;
  /* justify-content: center; */
  align-items: center;
  padding-top: 30px;
  padding-bottom: 40px;
}


.calculator__title {
  color: #fff;
  font-weight: 100;
  padding-top: 0;
  margin-top: 0;
  padding-bottom: 10px;
}

form {
```

```
  width: 80%;

}

select {
  width: 100%;
  background: none;
  border-color: transparent;
  outline: none;
  /* remove focus ring from Webkit */
  background: rgba(0, 0, 0, 0.07);
  -webkit-appearance: none;
  /* remove the strong OSX influence from Webkit */
  border-radius: 0;
  padding: 12px;
  font-size: 16px;
  color: rgba(255, 255, 255, 0.678);
}

.select {
  position: relative;
  margin-bottom: 20px;
}

.select::after {
  content: '\f078';
  font: normal normal normal 17px/1 FontAwesome;
  color: rgba(255, 255, 255, 0.678);
  right: 11px;
  top: 0px;
  height: 40px;
  line-height: 40px;
  position: absolute;
  pointer-events: none;
}

.input {
  position: relative;
  margin-bottom: 20px;
  width: 100%;
}

input {
  width: 100%;
  background: rgba(0, 0, 0, 0.07);
  border: none;
  padding: 13px;
  color: rgba(255, 255, 255, 0.678);
  font-size: 16px;
}
```

```
input:-internal-autofill-selected {
  background-color: rgba(0, 0, 0, 0.07) !important;
}

input:-webkit-autofill,
input:-webkit-autofill:hover,
input:-webkit-autofill:focus,
input:-webkit-autofill:active {
  background-color: rgba(0, 0, 0, 0.07) !important;

}

input:-webkit-autofill {
  -webkit-box-shadow: inset 0 0 0px 9999px rgba(0, 0, 0, 0.07) !important;
}

input:-webkit-autofill {
  -webkit-box-shadow: inset 0 0 0px 9999px rgba(0, 0, 0, 0.07);
}

input:focus,
input:-webkit-autofill:focus {
  -webkit-box-shadow: inset 0 0 0px 9999px rgba(0, 0, 0, 0.07), 0 0 8px rgba(0, 0,
0, 0.07);
}

input::placeholder {
  color: rgba(255, 255, 255, 0.678);
  font-size: 16px;
}
input[type=submit] {
  color: #fff;
}
.calculator__btn {
  width: 100%;
  font-size: 18px;
  border: none;
  border-radius: none;
  background: #007acc;
  color: rgba(255, 255, 255, 0.678);
  font-weight: 100;
  cursor: pointer;
  font-family: 'Nunito', sans-serif;
  padding: 11.5px;

}

/* --------------calculator styles end------------------ */
/* --------------weather styles start------------------ */

#wrapper {
```

```
  flex-direction: column;
  padding: 0px;
  color: white;
}

.weather__container {
  width: 51%;
}

#current-weather {
  padding: 15px;
  margin-left: -15px;
  margin-right: -15px;
}

#mainTemperature {
  font-size: 3.5em;
  line-height: 0.7;
}

#tempDescription {
  margin-top: 10px;
  text-align: center;
}

.day-weather-box {
  /* background-color: #007acc; */
  border-radius: 5px;
  height: 5em;
}

.day-weather-box p {
  margin-bottom: 0;
}

.side-weather-info {
  padding: 0px 10px;
}

.day-weather-inner-box {
  display: flex;
  flex-direction: column;
  margin: 14px auto;
  padding: 0px 5px;
  align-items: center;
}

.forecast-main {
  padding: 0px 0px 0px 30px;
}
```

```
.forecast-icon {
  font-size: 25px;
  margin-left: 5px;
}

.modal-body p {
  color: #333;
}

.d-flex {
  width: 40px;
  justify-content: space-between;
}

.calculation-container {
  flex-direction: column;
  padding-bottom: 50px;
  color: #fff;
}

.calculation-container h2 {
  padding-bottom: 30px;
  text-align: center;
  font-family: 'Nunito', sans-serif;
  font-size: 40px;
  font-weight: 300;

}
.datepickers {
  padding-top: 75px;
}
.datepickers > form {
  display: flex;
  align-items: center;
  width: 100%;
}
.datepickers > form > div:first-child {
  margin-right: 20px;
}
.datepicker {
  position: relative;
  width: 33%;
}
.datepicker label {
  position: absolute;
  right: 19px;
  top: 20px;
}

.datepicker input {
  border: 1px solid #fff;
```

```
  padding: 12px;
  border-radius: 5px;
}
.datepicker-submit-btn {
  margin-left: auto;
}
.datepicker-submit-btn input {
  border: 1px solid #fff;
  padding: 12px 30px;
  border-radius: 5px;
  cursor: pointer;
}
.datepicker-submit-btn input:hover {
  color: #014077;
  background: #fff;
  transition: .3s all ease-in-out;
}
td.today.day {
  background: #007acc!important;
  color: #fff!important;
}
h6 {
  font-weight: 200!important;
}
.activation-energy {
  padding-top: 40px;
  display: flex;
  align-items: flex-start;
  flex-direction: column;
}
.dropdown-menu {
  min-width: 13rem!important;
  padding: 10px!important;
}
.activation-energy>div {
  display: flex;
  margin-bottom: 10px;
}

/* --------------weather styles end------------------ */
/* --------------graph styles start------------------ */
.graph-section {
  position: relative;
  width: 100%;
  height: auto;
  margin: 50px 0;
}

.graph-section>div {
  /* height: 300px; */
  position: relative;
```

```
  width: 100%;
  background: #014c8f;

}

/* --------------graph styles end------------------ */
/* --------------footer styles start------------------ */
footer {
  background: #014077;
}

.footer {
  display: flex;
  /* flex-direction: column; */
  padding-top: 40px;
  padding-bottom: 40px;
  justify-content: space-between;
  align-items: center;
  color: #fff;
}

.footer img {
  width: 90px;
}

.footer p {
  margin: 0;
  font-size: 15px;
  letter-spacing: 1.5px;
}

a {
  opacity: 1;
}

.footer-logo {
  /* margin-bottom: 30px; */
}

/* --------------footer styles end------------------ */

.container {
  width: 1300px !important;
  margin: auto;
  display: flex;
  justify-content: space-between;
}
```

3. Calculator.ejs

```
<div class="calculator-card">
    <h2 class="calculator__title">Calculate the graph</h2>
    <form class="calculator-form" method="POST" action="/#graph">
        <div class="select">
            <select name="specimen" required> <!-- Dropdown menu for Specimen
Number-->
                <option value="<%=specNumDefault%>"
selected><%=specNumDefault%></option>
                <option value=1>1</option>
                <option value=2>2</option>
                <option value=3>3</option>
                <option value=4>4</option>
                <option value=5>5</option>
                <option value=6>6</option>
            </select>
        </div>
        <div class="select">
            <select name="depth" required> <!-- Dropdown menu for Depth -->
                <option value="<%=depthDefault%>"
selected><%=depthDefault%></option>
                <option value=1>5</option>
                <option value=2>10</option>
                <option value=3>15</option>
                <option value=4>20</option>
                <option value=5>30</option>
                <option value=6>40</option>
                <option value=7>50</option>
              </select>
        </div>
        <div class="select">
            <select name="thermistor" required> <!-- Dropdown menu for Thermistor -
->
                <option value="<%=thNumDefault%>"
selected><%=thNumDefault%></option>
                <option value=1>1</option>
                <option value=2>2</option>
                <option value=3>3</option>
                <option value=4>4</option>
              </select>
        </div>
        <div class="input">
            <input type="number" name="ae" placeholder="<%=aaDefault%>" required>
<!-- Entry field for Activation Energy  -->
        </div>
        <div class="input"
            <input type="number" name="tt" placeholder="<%=ttDefault%>" required>
<!-- Entry field for Targer Temperature -->
        </div>
        <input type="submit" class="calculator__btn" value="Submit">
```

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

4. Home.ejs

```
<!-- Combine other ejs files in one page  -->
<% include partials/header %>
<div class="wrapper">
  <% include partials/navigation %>
  <main class="main">
    <div class="container">
      <% include partials/calculator %>
      <% include partials/weather %>
    </div>
  </main>

</div>
<div id="graph" class="graph-section"> <!-- Adjust graph sizes -->
  <div id="resChart" style="height: 500px;margin-bottom: 40px;"></div>
  <div id="tempChart" style="height: 400px;margin-bottom: 40px"></div>
  <div id="logChart" style="height: 400px;margin-bottom: 40px"></div>
</div>
<% include partials/footer %>
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
<script src="./js/weather.js"></script>
<script>
    //Declear variables to be recieved from back-end
  var res_cor_data = [];
  var res_raw_data = [];
  var temp_data = [];
  var time_labels = [];
  var day_labels = [];
  var labels = [];
  var t1 = !<%=t1%>;
  var t2 = !<%=t2%>;
  var t3 = !<%=t3%>;
  var t4 = !<%=t4%>;
  var temp1 = [];
  var temp2 = [];
  var temp3 = [];
  var temp4 = [];
  var yLn = [];
  var xKt1 = [];
  var xKt2 = [];
  var xKt3 = [];
  var xKt4 = [];
  var realtemp = [];
  var tidalheight = [];
  // Getting data array on integers from backend
  <% for (var d of res_cor_data) { %>
  res_cor_data.push(<%=d%>);
```

```
<% } %>
<% for (var d of res_raw_data) { %>
res_raw_data.push(<%=d%>);
<% } %>
<% for (var d of temp_data) { %>
temp_data.push(<%=d%>);
<% } %>
<% for (var d of temp1) { %>
temp1.push(<%=d%>);
<% } %>
<% for (var d of temp2) { %>
temp2.push(<%=d%>);
<% } %>
<% for (var d of temp3) { %>
temp3.push(<%=d%>);
<% } %>
<% for (var d of temp4) { %>
temp4.push(<%=d%>);
<% } %>
<% for (var d of yLn) { %>
yLn.push(<%=d%>);
<% } %>
<% for (var d of xKt1) { %>
xKt1.push(<%=d%>);
<% } %>
<% for (var d of xKt2) { %>
xKt2.push(<%=d%>);
<% } %>
<% for (var d of xKt3) { %>
xKt3.push(<%=d%>);
<% } %>
<% for (var d of xKt4) { %>
xKt4.push(<%=d%>);
<% } %>
<% for (var d of realtemp) { %>
realtemp.push(<%=d%>);
<% } %>
<% for (var d of tidalheight) { %>
tidalheight.push(<%=d%>);
<% } %>
var logRaw = [];
for (var d of res_raw_data) {
  logRaw.push(Math.log(d));
}
// Getting labels array of string from back-end
time_labels = '<%=time_labels%>';
time_labels = time_labels.split(",");
day_labels = '<%=day_labels%>';
day_labels = day_labels.split(",");
realdate = '<%=realdate%>';
realdate = realdate.split(",");
```

```
tidaldate = '<%=tidaldate%>';
tidaldate = tidaldate.split(",");
for (i = 0; i < day_labels.length; i++) {
  labels.push(day_labels[i] + " " + time_labels[i]);
}

// 1st chart
var corData = {
  x: labels,
  y: res_cor_data,
  name: "Corrected",
  type: "scatter"
};
var rawData = {
  x: labels,
  y: res_raw_data,
  name: "Raw",
  type: "scatter"
};
var data = [
  corData,
  rawData
];
var layout = {
  title: 'Resistivity',
  font: {
    family: "'Nunito', sans-serif",
    size: 17,
    color: '#004c8e'
  },
  xaxis: {
  'rangeslider': {
  },
    color: '#333',
    tickfont: {
      size: 14,
      color: '#7f7f7f'
    }
  },
  yaxis: {
    fixedrange: true,
    color: '#333',
    tickfont: {
      size: 14,
      color: '#7f7f7f'
    }
  },
  margin: {
    b: 0,
    t: 50,
    pad: 0
```

```
  }
};
Plotly.plot('resChart', data, layout, {
  displayModeBar: false
});

// 2nd chart
var t1Data = {
  x: labels,
  y: temp1,
  name: "Thermistor 1",
  visible: (t1) ? "legendonly" : true,
  type: "scatter"
}
var t2Data = {
  x: labels,
  y: temp2,
  name: "Thermistor 2",
  visible: (t2) ? "legendonly" : true,
  type: "scatter"
}
var t3Data = {
  x: labels,
  y: temp3,
  name: "Thermistor 3",
  visible: (t3) ? "legendonly" : true,
  type: "scatter"
}
var t4Data = {
  x: labels,
  y: temp4,
  name: "Thermistor 4",
  visible: (t4) ? "legendonly" : true,
  type: "scatter"
}
var realData = {
  x: realdate,
  y: realtemp,
  name: "Real temp.",
  visible: true,
  type: "scatter"
}
var tidalData = {
  x: tidaldate,
  y: tidalheight,
  name: "Tidal",
  visible: true,
  type: "scatter"
}
var data = [
  t1Data,
```

```
  t2Data,
  t3Data,
  t4Data,
  realData,
  tidalData,
];
var layout = {
  title: 'Temperature (degC)',
  font: {
    family: "'Nunito', sans-serif",
    size: 17,
    color: '#004c8e'
  },
  xaxis: {
    rangeslider: {
    },
    color: '#333',
    tickfont: {
      size: 14,
      color: '#7f7f7f'
    }

  },
  yaxis: {
    fixedrange: true,
    color: '#333',
    tickfont: {
      size: 14,
      color: '#7f7f7f'
    }

  },
  margin: {
    b: 0,
    t: 50,
    pad: 0
  }
};
Plotly.plot('tempChart', data, layout, {
  displayModeBar: false
});

// 3rd chart
var trace1 = {
  x: xKt1,
  y: yLn,
  name: "ln1",
  visible: (t1) ? "legendonly" : true,
  type: "scatter",
  mode: "markers"
}
```

```
var trace2 = {
  x: xKt2,
  y: yLn,
  name: "ln2",
  visible: (t2) ? "legendonly" : true,
  type: "scatter",
  mode: "markers"
}
var trace3 = {
  x: xKt3,
  y: yLn,
  name: "ln3",
  visible: (t3) ? "legendonly" : true,
  type: "scatter",
  mode: "markers"
}
var trace4 = {
  x: xKt4,
  y: yLn,
  name: "ln4",
  visible: (t4) ? "legendonly" : true,
  type: "scatter",
  mode: "markers"
}
var data = [
  trace1,
  trace2,
  trace3,
  trace4
];
var layout = {
  title: 'Log Graph',
  font: {
    family: "'Nunito', sans-serif",
    size: 17,
    color: '#004c8e'
  },
  yaxis: {
    fixedrange: true,
    color: '#333',
    tickfont: {
      size: 14,
      color: '#7f7f7f'
    }

  },
  xaxis: {
    color: '#333',
    tickfont: {
      size: 14,
      color: '#7f7f7f'
```

```
      }

    },
    margin: {
      b: 100,
      t: 50,
      pad: 0
    }
  };
  Plotly.newPlot('logChart', data, layout, {
    displayModeBar: false
  });

  // Display Activation energy

  document.getElementById("s1").innerHTML = "Activation Energy 1: " +
Math.round(<%=s1%> * 100) / 100 + " kJ/Mole";
  document.getElementById("s2").innerHTML = "Activation Energy 2: " +
Math.round(<%=s2%> * 100) / 100 + " kJ/Mole";
  document.getElementById("s3").innerHTML = "Activation Energy 3: " +
Math.round(<%=s3%> * 100) / 100 + " kJ/Mole";
  document.getElementById("s4").innerHTML = "Activation Energy 4: " +
Math.round(<%=s4%> * 100) / 100 + " kJ/Mole";
</script>
```

5. Weather.ejs

```
<div class="weather__container">
    <div class="container" id="wrapper">
        <div class="container-fluid" id="current-weather">
            <div class="row">
                <!-- Right panel -->
                <div class="col-md-5 col-sm-5">
                    <h5>
                        <spam id="cityName"></spam>, <spam id="cityCode"></spam>
                    </h5>
                    <h6 id="localDate"></h6>
                </div>
                <!-- Center panel -->
                <div class="col-md-3 col-sm-7" style="margin: 10px
auto;padding:0;">
                    <div class="row">
                        <i class="wi" id="main-icon" style="font-size: 60px;"></i>
                        <div style="display:flex; flex-direction: column; align-
items: center">
                            <spam id="mainTemperature"></spam>
                            <p id="tempDescription"></p>
                        </div>
                        <p style="font-size: 1.5rem;"><a style="color: #fff"
id="celcius">°C</a>
                    </div>
```

```
                </div>

                <!-- Left panel -->
                <div class="col-md-4 col-xs-12 col-sm-12 row" style="text-align:
right;">


                    <div class="col-md-12 col-sm-3 col-xs-3 side-weather-info">
                        <h6>Wind: <span id="wind"></span> m/s</h6>
                    </div>
                    <div class="col-md-12 col-sm-3 col-xs-3 side-weather-info">
                        <h6>Humidity: <span id="humidity"></span>%</h6>
                    </div>
                    <div class="col-md-12 col-sm-3 col-xs-3 side-weather-info">
                        <h6>High: <span id="mainTempHot"></span>°</h6>
                    </div>
                    <div class="col-md-12 col-sm-3 col-xs-3 side-weather-info">
                        <h6>Low: <span id="mainTempLow"></span>°</h6>
                    </div>

                </div>

            </div>

        </div>

        <!-- 6 days forecast -->
        <div class="row" style="width: 100%; padding: 2px;">

            <!-- Day 1 -->
            <div class="col-md-2 col-sm-6 day-weather-box">
                <div class="col-sm-12 day-weather-inner-box">
                    <p id="forecast-day-1-name"></p>
                    <div class="row my-2">
                        <i class="wi forecast-icon" id="forecast-day-1-icon"></i>
                    </div>
                    <div class="d-flex">
                        <span class="high-temperature" id="forecast-day-1-
ht"></span>
                        <span class="low-temperature" id="forecast-day-1-
lt"></span>
                    </div>

                </div>
            </div>
            <!-- Day 2 -->
            <div class="col-md-2 col-sm-6 day-weather-box">
                <div class="col-sm-12 day-weather-inner-box">
                    <p id="forecast-day-2-name"></p>
                    <div class="row my-2">
                        <i class="wi forecast-icon" id="forecast-day-2-icon"></i>
```

```
                    </div>
                    <div class="d-flex">
                        <span class="high-temperature" id="forecast-day-2-
ht"></span>
                        <span class="low-temperature" id="forecast-day-2-
lt"></span>
                    </div>
                </div>
            </div>
            <!-- Day 3 -->
            <div class="col-md-2 col-sm-6 day-weather-box">
                <div class="col-sm-12 day-weather-inner-box">
                    <p id="forecast-day-3-name"></p>
                    <div class="row my-2">
                        <i class="wi forecast-icon" id="forecast-day-3-icon"></i>
                    </div>
                    <div class="d-flex">
                        <span class="high-temperature" id="forecast-day-3-
ht"></span>
                        <span class="low-temperature" id="forecast-day-3-
lt"></span>
                    </div>
                </div>
            </div>
            <!-- Day 4 -->
            <div class="col-md-2 col-sm-6 day-weather-box">
                <div class="col-sm-12 day-weather-inner-box">
                    <p id="forecast-day-4-name"></p>
                    <div class="row my-2">
                        <i class="wi forecast-icon" id="forecast-day-4-icon"></i>
                    </div>
                    <div class="d-flex">
                        <span class="high-temperature" id="forecast-day-4-
ht"></span>
                        <span class="low-temperature" id="forecast-day-4-
lt"></span>
                    </div>
                </div>
            </div>
            <!-- Day 5 -->
            <div class="col-md-2 col-sm-6 day-weather-box">
                <div class="col-sm-12 day-weather-inner-box">
                    <p id="forecast-day-5-name"></p>
                    <div class="row my-2">
                        <i class="wi forecast-icon" id="forecast-day-5-icon"></i>
                    </div>
                    <div class="d-flex">
                        <span class="high-temperature" id="forecast-day-5-
ht"></span>
                        <span class="low-temperature" id="forecast-day-5-
lt"></span>
```

```
                    </div>
                </div>
            </div>
            <!-- Day 6 -->
            <div class="col-md-2 col-sm-6 day-weather-box">
                <div class="col-sm-12 day-weather-inner-box">
                    <p id="forecast-day-6-name"></p>
                    <div class="row my-2">
                        <i class="wi forecast-icon" id="forecast-day-6-icon"></i>
                    </div>
                    <div class="d-flex">
                        <span class="high-temperature" id="forecast-day-6-
ht"></span>
                        <span class="low-temperature" id="forecast-day-6-
lt"></span>
                    </div>
                </div>
            </div>
        </div>
        <% include activation-energy %>
    </div>

</div>
```

6. Weather.js

```
var unitIsCelcius = true;
var globalForecast = [];
// Maps the API's icons to the ones from https://erikflowers.github.io/weather-
icons/
var weatherIconsMap = {
  "01d": "wi-day-sunny",
  "01n": "wi-night-clear",
  "02d": "wi-day-cloudy",
  "02n": "wi-night-cloudy",
  "03d": "wi-cloud",
  "03n": "wi-cloud",
  "04d": "wi-cloudy",
  "04n": "wi-cloudy",
  "09d": "wi-showers",
  "09n": "wi-showers",
  "10d": "wi-day-hail",
  "10n": "wi-night-hail",
  "11d": "wi-thunderstorm",
  "11n": "wi-thunderstorm",
  "13d": "wi-snow",
  "13n": "wi-snow",
  "50d": "wi-fog",
  "50n": "wi-fog"
};
```

```
$(function(){
  getClientPosition();
  startClock();
});


function startClock(){
  setInterval(function(){
    $("#localTime").text(new Date().toLocaleTimeString());
  }, 1000);
}


function getClientPosition(){
    position = {
      city: "Dornoch",
      country: "GB",
      latitude: 57.88,
      longitude: -4.03
    }
    $("#cityName").text(position.city);
    $("#cityCode").text(position.country);

    getWeatherData(position.latitude, position.longitude);
}


function getWeatherData(latitude, longitude){
  $.ajax({
    type: "GET",
    url: "https://cors-
anywhere.herokuapp.com/http://api.openweathermap.org/data/2.5/forecast/daily?APPID=
9b4bbf30228eb8528d36e79d05da1fac&lat=" + latitude + "&lon=" + longitude +
"&units=metric&cnt=7",
    cache: true,
    headers: {
      "Access-Control-Allow-Headers": "x-requested-with"
    },
    success: function(forecast){
      globalForecast = forecast;
      updateForecast(forecast);

    },
    error: function(error){
      console.log("Error with ajax: "+ error);
    }
  });
}


// Update view values from passed forecast
function updateForecast(forecast){
```

```
  // Present day
  var today = forecast.list[0];
  $("#tempDescription").text(toCamelCase(today.weather[0].description));
  $("#humidity").text(today.humidity);
  $("#wind").text(today.speed);
  $("#localDate").text(getFormattedDate(today.dt));
  $("#main-icon").addClass(weatherIconsMap[today.weather[0].icon]);
  $("#mainTemperature").text(Math.round(today.temp.day));
  $("#mainTempHot").text(Math.round(today.temp.max));
  $("#mainTempLow").text(Math.round(today.temp.min));


  // Following days data
  for(var i = 1; i < (forecast.list).length; i++){
    var day = forecast.list[i];

    // Day short format e.g. Mon
    var dayName = getFormattedDate(day.dt).substring(0,3);

    // weather icon from map
    var weatherIcon = weatherIconsMap[day.weather[0].icon];

    $("#forecast-day-" + i + "-name").text(dayName);
    $("#forecast-day-" + i + "-icon").addClass(weatherIcon);
    $("#forecast-day-" + i + "-main").text(Math.round(day.temp.day));
    $("#forecast-day-" + i + "-ht").text(Math.round(day.temp.max));
    $("#forecast-day-" + i + "-lt").text(Math.round(day.temp.min));
  }
}


// Applies the following format to date: WeekDay, Month Day, Year
function getFormattedDate(date){
  var options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric'
};
  return new Date(date * 1000).toLocaleDateString("en-US",options);
}


// Formats the text to CamelCase
function toCamelCase(str) {
  var arr = str.split(" ").map(
    function(sentence){
      return sentence.charAt(0).toUpperCase() + sentence.substring(1);
    }
  );
  return arr.join(" ");
}

  $('#start-date').datepicker({
    todayHighlight: true,
```

```
     orientation: "bottom left",
     format: "mm/dd/yy",
    container: ".start-date"
  });
  $('#end-date').datepicker({
    todayHighlight: true,
     orientation: "bottom left",
     format: "mm/dd/yy",
    container: ".end-date"
  });
```

7. Activation-energy.ejs

```
<div class="datepickers">
</div>
<div class="row activation-energy"> // display activation energy
    <div class="col-md-12 col-sm-3 col-xs-3 side-weather-info">
        <h6> <span id="s1"></span></h6>
    </div>
    <div class="col-md-12 col-sm-3 col-xs-3 side-weather-info">
        <h6><span id="s2"></span></h6>
    </div>
    <div class="col-md-12 col-sm-3 col-xs-3 side-weather-info">
        <h6><span id="s3"></span></h6>
    </div>
    <div class="col-md-12 col-sm-3 col-xs-3 side-weather-info">
        <h6><span id="s4"></span></h6>
    </div>
</div>
```

8. Navigation.ejs

```
<!-- Declaring buttons for navigation bar -->
<header class="header">
    <div class="container" style="align-items: center">
        <a href="/" class="header__logo">Vagif Aliyev</a>
        <nav class="header__nav">
            <a href="/" class="header__nav-item">Home</a>
            <a class="header__nav-item" href="/about">About</a>
        </nav>
    </div>
</header>
```

9. Footer.ejs

```
<footer>
  <div class="container footer">
      <a href="https://www.hw.ac.uk/" target="_blank" class="footer-logo">
          <img src="pictures/logo.png" alt=""> <!-- insert HWU logo -->
      </a>
      <p>© Heriot-Watt University. Copyright 2019. All Rights Reserved</p>
  </div>
</footer>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
crossorigin="anonymous"></script>
  <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
crossorigin="anonymous"></script>
</body>
</html>
```

10.About.ejs

```
<% include partials/header %>
<div class="about">
  <% include partials/navigation %>
  <main class="main">
    <div class="container">
      <div class="calculation">
        <div class="container calculation-container">
            <h2 class="mx-auto">Calculations</h2>
            <p>
                To be able to convert measured resistance, R (ohms), to resistivity
ρ (ohm-cm) the electrodes were
                calibrated in a solution of known resistivity before embedding
them. This can be done through
                Equation (1):
            </p>
            <p class="mx-auto">
                <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}\rho&space;=&space;kR&space;\hspace{1cm}&space;(1)&space;}"
target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}\
rho&space;=&space;kR&space;\hspace{1cm}&space;(1)&space;}" title="\large
{\color{White}\rho = kR \hspace{1cm} (1) }" /></a>
            </p>
            <p>
```

```
                where k = calibration factor for the array, was obtained as 1.25 cm
± 5%

            </p>
            <p>
                The resistance values measured by the thermistors can be converted
into degree Celsius (°C) using
                the Steinhart-Hart equation:

            </p>
            <a class="mx-auto my-4"
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;{\color{White}&space;
T&space;=&space;[A&plus;B\ln&space;R&space;&plus;&space;C(\ln&space;R)^{3}]^{-
1}&space;-&space;273.15&space;\hspace{1cm}&space;(2)}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;{\color{White}&space;T&space
;=&space;[A&plus;B\ln&space;R&space;&plus;&space;C(\ln&space;R)^{3}]^{-1}&space;-
&space;273.15&space;\hspace{1cm}&space;(2)}" title="{\color{White} T = [A+B\ln R +
C(\ln R)^{3}]^{-1} - 273.15 \hspace{1cm} (2)}" /></a>
            <p>Where,</p>
            <p>T is the temperature (°C),</p>
            <p>R is the resistance measured by the thermistor (ohms),</p>
            <p>A, B and C are coefficients dependent on the type of thermistor and
are
                <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;{\color{White}&space;
1.28\times&space;10^{-3}K^{-1},&space;2.36\times&space;10^{-4}K^{-
1},&space;and&space;\hspace{0.2cm}&space;9.31\times&space;10^{-8}&space;K^{-1}}"
target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;{\color{White}&space;1.28\ti
mes&space;10^{-3}K^{-1},&space;2.36\times&space;10^{-4}K^{-
1},&space;and&space;\hspace{0.2cm}&space;9.31\times&space;10^{-8}&space;K^{-1}}"
title="{\color{White} 1.28\times 10^{-3}K^{-1}, 2.36\times 10^{-4}K^{-1}, and
\hspace{0.2cm} 9.31\times 10^{-8} K^{-1}}" /></a>
            </p>
            <p>
                Ln stands for natural logarithm
            </p>
            <p>
                To model the influence of temperature on resistivity, an Arrhenius
relationship is used:
            </p>
            <a  class="mx-auto my-3"
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;\rho&space;=&space;\rho&space;_{0}e^{[\frac{E_{a}}{R_{g}T_{k}}]}&space
;\hspace{1cm}&space;(3)}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;\rho&space;=&space;\rho&space;_{0}e^{[\frac{E_{a}}{R_{g}T_{k}}]}&space;\hspac
e{1cm}&space;(3)}" title="\large {\color{White} \rho = \rho
_{0}e^{[\frac{E_{a}}{R_{g}T_{k}}]} \hspace{1cm} (3)}" /></a>
            <p>Where,</p>
            <p>ρ is resistivity (kΩ-cm),</p>
```

```
            <p><a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;T_{k}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;T_{k}}" title="\large {\color{White} T_{k}}" /></a> is
                absolute temperature (K), </p>
            <p>
              <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;\rho&space;_{0}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;\rho&space;_{0}}" title="\large {\color{White} \rho _{0}}" /></a> is pre-
exponential constant (kΩ-cm),
            </p>
            <p>
              <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;R&space;_{g}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;R&space;_{g}}" title="\large {\color{White} R _{g}}" /></a>
                is gas constant
                <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;{\color{White}&space;
(8.3141&space;\times&space;10^{-3}&space;KJ/mole/K)}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;{\color{White}&space;(8.3141
&space;\times&space;10^{-3}&space;KJ/mole/K)}" title="{\color{White} (8.3141 \times
10^{-3} KJ/mole/K)}" /></a> and
            </p>
            <p>
              <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;E_{a}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;E_{a}}" title="\large {\color{White} E_{a}}" /></a> is
                activation energy for conduction processes in concrete (kJ/mole).
            </p>
            <p>
                If
                <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;\rho&space;_{x}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;\rho&space;_{x}}" title="\large {\color{White} \rho _{x}}" /></a>
                and
                <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;\rho&space;_{y}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;\rho&space;_{y}}" title="\large {\color{White} \rho _{y}}" /></a>
                are the resistivity readings at temperatures
```


and

respectively, then, from Eq. (3):

From Eq. (4), a value of resistivity,

, recorded at a temperature,

can be used to get an
equivalent resistivity,

,of the material at temperature,

, with a known
<a href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{

```
White}&space;E_{a}&space;/&space;R_{g}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;E_{a}&space;/&space;R_{g}}" title="\large {\color{White} E_{a} / R_{g}}"
/></a>
                ratio for the
                conduction stage. The process standardizes the measurements to a
reference temperature, removing the
                effect of temperature on the electrical resistivity. The activation
energy can be entered either by
                the researcher or can be evaluated from the data for a more
accurate value. The reference
                temperature
                (<a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;T&space;_{k,y}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;T&space;_{k,y}}" title="\large {\color{White} T _{k,y}}" /></a>)
                is provided by the user, usually around 25°C (298.15 K).
            </p>
            <p>
                Through the experimental measurements the ratio of
                <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;E_{a}&space;/&space;R_{g}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;E_{a}&space;/&space;R_{g}}" title="\large {\color{White} E_{a} / R_{g}}"
/></a>
                can be determined and as the gas constant
                (<a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;R&space;_{g}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;R&space;_{g}}" title="\large {\color{White} R _{g}}" /></a>)
                is known the activation energy
                (<a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;E_{a}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;E_{a}}" title="\large {\color{White} E_{a}}" /></a>)
                can be evaluated for the particular mix and electrode pair
                depth. Equation (3) can be rewritten as
            </p>
            <a class="mx-auto mt-2 mb-4"
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;\ln&space;\rho&space;=&space;\ln\rho_{0}&plus;\frac{E_{a}}{R_{g}T_{k}}
&space;\hspace{1cm}&space;(5)}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;\ln&space;\rho&space;=&space;\ln\rho_{0}&plus;\frac{E_{a}}{R_{g}T_{k}}&space;
\hspace{1cm}&space;(5)}" title="\large {\color{White} \ln \rho =
\ln\rho_{0}+\frac{E_{a}}{R_{g}T_{k}} \hspace{1cm} (5)}" /></a>
                <p>
```

```
                        The plot of ln ρ against
                        <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;T_{k}/1000}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;T_{k}/1000}" title="\large {\color{White} T_{k}/1000}" /></a>
                         , will be a straight line with a slope of
                         <a
href="https://www.codecogs.com/eqnedit.php?latex=\fn_cm&space;\large&space;{\color{
White}&space;E_{a}&space;/&space;R_{g}}" target="_blank"><img
src="https://latex.codecogs.com/svg.latex?\fn_cm&space;\large&space;{\color{White}&
space;E_{a}&space;/&space;R_{g}}" title="\large {\color{White} E_{a} / R_{g}}"
/></a>
                          . For this process the temperature values must be in
units of Kelvin.
                    </p>
        </div>
      </div>
    </div>
  </main>
</div>

<% include partials/footer %>
```

11.Script.py

```
import csv
import time
import os

readDirectory = 'readDirectory'
writeDirectory = '../apps/websiteProject/htdocs/csv'
backupDirectory = 'backupDirectory'
writeFileName = 'database.csv'
timeSleep = 1

# Infinite loop which checks for newly arrived data files
while True:
    # If the specified path exists
    if os.path.exists(readDirectory):
        # Get all items in the directory
        files = os.listdir(readDirectory)
        # If there are any files
        if len(files) > 0:
            # Open the file we read from and open the file we write to
            with open(readDirectory + '/' + (sorted(files))[0],'r',
encoding='latin-1') as csvRead, open(writeDirectory + '/' + writeFileName, 'a') as
csvWrite:
                readerCSV = csv.reader(csvRead, delimiter=',')
                writerCSV = csv.writer(csvWrite, delimiter=',')
```

```
                # Copy data from the read file into write file
                print("Copying the files...")
                for row in readerCSV:
                    writerCSV.writerow(row)
                # Move copied file to the backup folder
                os.rename(readDirectory + '/' + sorted(files)[0], backupDirectory +
'/' + sorted(files)[0])
        else:
            print("Folder is empty.")
    else:
        # If the read path is not specified properly we exit with Error
        print("Path to the read folder is wrong!:   " + readDirectory)
        raise Exception

    print('Waiting' + str(timeSleep) + 'seconds...')
    time.sleep(timeSleep)
```

12.App.js

```
// Request Libraries
var express = require('express');
var bodyParser = require('body-parser');
var path = require('path');
var fs = require('fs');
var csv = require('fast-csv');
var schedule = require('node-schedule');
var http = require('http');
var https = require('https');

var app = express();

// Establish SQL Database connection
const { Pool, Client } = require('pg')
const connectionString =
'postgres://uyuapysi:pYlez3muTAJvLGLdigo_rxHv0r1n0a_5@manny.db.elephantsql.com:5432
/uyuapysi'

// Creating a connection pool
const pool = new Pool({
  connectionString: connectionString,
})
// Check that DB connection was established
if (!pool){
  console.log("Error while connecting to the DB");
}

// Cron job that runs every 30 minutes
var temp;
var j = schedule.scheduleJob('*/30 * * * *', function(){
  // Temperature from weather station
```

```
http.get('http://api.openweathermap.org/data/2.5/weather?q=Dornoch&APPID=f7eb12f351
a6cc84894f63865583ae7e', (resp) => {
  let data = '';
    // A chunk of data has been recieved.
    resp.on('data', (chunk) => {
      data += chunk;
    });
    // The whole set of data has been received
    resp.on('end', () => {
      temp = Math.round((JSON.parse(data).main.temp - 273.15)*100)/100;
      var d = new Date();
      var month = (d.getMonth()+1 < 10) ? '0'+(d.getMonth()+1) : (d.getMonth()+1);
      var day = (d.getDate() < 10) ? '0'+(d.getDate()+1) : (d.getDate()+1);
      var hour = (d.getHours() < 10) ? '0'+(d.getHours()) : (d.getHours());
      var minutes = (d.getMinutes() < 10) ? '0'+(d.getMinutes()) :
(d.getMinutes());
      var seconds = (d.getSeconds() < 10) ? '0'+(d.getSeconds()) :
(d.getSeconds());
      var date = '' + d.getFullYear() + '-' + month + '-' + day + ' ' + hour + ':'
+ minutes + ':' + seconds;
      // Constructing SQL query
      var insert = 'INSERT INTO temp (date, temp) VALUES (\''+date+'\',
\''+temp+'\');';

      pool.query(insert, (err, res) => {
        if (err){
          console.log("Error while inserting into the DB");
        } else {
          console.log("Data inserted");
        }
      })

    });
  }).on("error", (err) => {
    console.log("Error: " + err.message);
  });

  // Tidal data
  https.get('https://environment.data.gov.uk/flood-
monitoring/id/stations/E70739/measures', (resp) => {
  let data = '';
    // A chunk of data has been recieved.
    resp.on('data', (chunk) => {
      data += chunk;
    });
    // The whole response has been received.
    resp.on('end', () => {
      tidal = JSON.parse(data).items[0].latestReading.value;
      var d = new Date();
      var month = (d.getMonth()+1 < 10) ? '0'+(d.getMonth()+1) : (d.getMonth()+1);
```

```
      var day = (d.getDate() < 10) ? '0'+(d.getDate()+1) : (d.getDate()+1);
      var hour = (d.getHours() < 10) ? '0'+(d.getHours()) : (d.getHours());
      var minutes = (d.getMinutes() < 10) ? '0'+(d.getMinutes()) :
(d.getMinutes());
      var seconds = (d.getSeconds() < 10) ? '0'+(d.getSeconds()) :
(d.getSeconds());
      var date = '' + d.getFullYear() + '-' + month + '-' + day + ' ' + hour + ':'
+ minutes + ':' + seconds;
      // Constructing SQL query

      var insert = 'INSERT INTO tidal (date, height) VALUES (\''+date+'\',
\''+tidal+'\');';

      pool.query(insert, (err, res) => {
        if (err){
          console.log("Error while inserting into the DB");
        } else {
          console.log("Data inserted");
        }
      })
    });
  }).on("error", (err) => {
    console.log("Error: " + err.message);
  });

  console.log('Cron job run successfully!');
});

// View Engine (declaring ejs will be used for front-end)
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Body Parser Middleware
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: false}));

// Set Static Path (path to directory for ejs files)
app.use(express.static(path.join(__dirname, 'views')));

// Passing empty values to the page when initially loaded
app.get('/', function(req, res){
   res.render('home',{
     res_cor_data: [],
     res_raw_data: [],
     temp_data: [],
     time_labels: [],
     day_labels: [],
     temp1: [],
     temp2: [],
     temp3: [],
     temp4: [],
```

```
      realtemp: [],
      realdate: [],
      tidaldate: [],
      tidalheight: [],
      t1: true,
      t2: false,
      t3: false,
      t4: false,
      yLn: [],
      xKt1: [],
      xKt2: [],
      xKt3: [],
      xKt4: [],
      s1: 0,
      s2: 0,
      s3: 0,
      s4: 0,
      specNumDefault: 'Specimen number',
      depthDefault: 'Depth in mm',
      thNumDefault: 'Thermistor number',
      aaDefault: 'Activation Energy in KJ/Mole',
      ttDefault: 'Target Temprature in degC'
   });
});

// About page setup
app.get('/about', function(req, res){
   res.render('about');
});

// Main graph page setup
// CSV Data Transfer
app.post('/', function(req, res){
// Variables declared
  var res_cor_data = [];
  var res_raw_data = [];
  var temp_data = [];
  var temp1 = [];
  var temp2 = [];
  var temp3 = [];
  var temp4 = [];
  var time_labels = [];
  var day_labels = [];
  var t1 = parseInt(req.body.thermistor) == 1;
  var t2 = parseInt(req.body.thermistor) == 2;
  var t3 = parseInt(req.body.thermistor) == 3;
  var t4 = parseInt(req.body.thermistor) == 4;
  var yLn = [];
  var xKt1 = [];
  var xKt2 = [];
  var xKt3 = [];
```

```
  var xKt4 = [];

  fs.createReadStream('./csv/database.csv') // reading the CSV database
    .pipe(csv())
    .on('data', function(data){
      // Correcting Temperature
      var tmp = data[1 + 11*(parseInt(req.body.specimen) - 1) + 7 +
parseInt(req.body.thermistor)];
      var cor_tmp =
1.287600011/1000+Math.log(tmp)*2.357183092/10000+Math.pow(Math.log(tmp),
3)*9.509464377/100000000;
      cor_tmp = 1/cor_tmp - 273.15;
      temp_data.push(Math.round(cor_tmp*100)/100);
      var tmp1 = data[1 + 11*(parseInt(req.body.specimen) - 1) + 7 + 1];
      var tmp2 = data[1 + 11*(parseInt(req.body.specimen) - 1) + 7 + 2];
      var tmp3 = data[1 + 11*(parseInt(req.body.specimen) - 1) + 7 + 3];
      var tmp4 = data[1 + 11*(parseInt(req.body.specimen) - 1) + 7 + 4];
      var cor_tmp1 =
1.287600011/1000+Math.log(tmp1)*2.357183092/10000+Math.pow(Math.log(tmp1),
3)*9.509464377/100000000;
      var cor_tmp2 =
1.287600011/1000+Math.log(tmp2)*2.357183092/10000+Math.pow(Math.log(tmp2),
3)*9.509464377/100000000;
      var cor_tmp3 =
1.287600011/1000+Math.log(tmp3)*2.357183092/10000+Math.pow(Math.log(tmp3),
3)*9.509464377/100000000;
      var cor_tmp4 =
1.287600011/1000+Math.log(tmp4)*2.357183092/10000+Math.pow(Math.log(tmp4),
3)*9.509464377/100000000;
      cor_tmp1 = 1/cor_tmp1 - 273.15;
      cor_tmp2 = 1/cor_tmp2 - 273.15;
      cor_tmp3 = 1/cor_tmp3 - 273.15;
      cor_tmp4 = 1/cor_tmp4 - 273.15;
      temp1.push(Math.round(cor_tmp1*100)/100);
      temp2.push(Math.round(cor_tmp2*100)/100);
      temp3.push(Math.round(cor_tmp3*100)/100);
      temp4.push(Math.round(cor_tmp4*100)/100);
      // Raw Data
      var rd = data[1 + 11*(parseInt(req.body.specimen) - 1) +
parseInt(req.body.depth)];
      // Corrected Data
      var ae = parseFloat(req.body.ae);
      var tt = parseFloat(req.body.tt);
      var exp = (1/(tt+273.15) - 1/(cor_tmp+273.15))*ae*1000/8.3141;
      var cd = Math.round(rd*Math.pow(Math.E, exp));
      res_raw_data.push(rd*0.0125);
      res_cor_data.push(cd*0.0125);
      // Time on X axis, date and time
      time_labels.push(data[1]);
      var day = data[0].substring(0,2);
      var month = data[0].substring(3,5);
```

```
  var year = data[0].substring(6,10);
  day_labels.push(year + "-" + month + "-" + day);
  // Values for 3rd graph
  yLn.push((Math.log(0.0125*rd)*100)/100);
  xKt1.push(((1000/(cor_tmp1+273.15))*100)/100);
  xKt2.push(((1000/(cor_tmp2+273.15))*100)/100);
  xKt3.push(((1000/(cor_tmp3+273.15))*100)/100);
  xKt4.push(((1000/(cor_tmp4+273.15))*100)/100);
})
// Calculating Activation Energies
.on('end', function(data){
  // Both _labels and _data are String Arrays
  console.log('Read Finished, calculating slope...');
  var s1;
  var s2;
  var s3;
  var s4;
  var sum = 0;
  for (i = 0; i < yLn.length; i++) {
    sum += yLn[i];
  }
  var yMean = sum/yLn.length;
  // AE 1
  sum = 0;
  for (i = 0; i < xKt1.length; i++) {
    sum += xKt1[i];
  }
  var xMean = sum/xKt1.length;
  var top = 0;
  var bX = 0;
  var bY = 0;
  for (i = 0; i < xKt1.length; i++) {
    top += (xKt1[i]-xMean)*(yLn[i]-yMean);
    bX += (xKt1[i]-xMean)*(xKt1[i]-xMean);
    bY += (yLn[i]-yMean)*(yLn[i]-yMean);
  }
  var r = top/Math.sqrt(bX*bY);
  var sy = Math.sqrt(bY/(xKt1.length-1));
  var sx = Math.sqrt(bX/(xKt1.length-1));
  s1 = r*sy/sx*8.3141;
  // AE 2
  sum = 0;
  for (i = 0; i < xKt2.length; i++) {
    sum += xKt2[i];
  }
  xMean = sum/xKt1.length;
  top = 0;
  bX = 0;
  bY = 0;
  for (i = 0; i < xKt1.length; i++) {
    top += (xKt2[i]-xMean)*(yLn[i]-yMean);
```

```
  bX += (xKt2[i]-xMean)*(xKt2[i]-xMean);
  bY += (yLn[i]-yMean)*(yLn[i]-yMean);
}
r = top/Math.sqrt(bX*bY);
sy = Math.sqrt(bY/(xKt2.length-1));
sx = Math.sqrt(bX/(xKt2.length-1));
s2 = r*sy/sx*8.3141;
// AE 3
sum = 0;
for (i = 0; i < xKt3.length; i++) {
  sum += xKt3[i];
}
xMean = sum/xKt1.length;
top = 0;
bX = 0;
bY = 0;
for (i = 0; i < xKt1.length; i++) {
  top += (xKt3[i]-xMean)*(yLn[i]-yMean);
  bX += (xKt3[i]-xMean)*(xKt3[i]-xMean);
  bY += (yLn[i]-yMean)*(yLn[i]-yMean);
}
r = top/Math.sqrt(bX*bY);
sy = Math.sqrt(bY/(xKt3.length-1));
sx = Math.sqrt(bX/(xKt3.length-1));
s3 = r*sy/sx*8.3141;
// AE 4
sum = 0;
for (i = 0; i < xKt4.length; i++) {
  sum += xKt4[i];
}
xMean = sum/xKt4.length;
top = 0;
bX = 0;
bY = 0;
for (i = 0; i < xKt4.length; i++) {
  top += (xKt4[i]-xMean)*(yLn[i]-yMean);
  bX += (xKt4[i]-xMean)*(xKt4[i]-xMean);
  bY += (yLn[i]-yMean)*(yLn[i]-yMean);
}
r = top/Math.sqrt(bX*bY);
sy = Math.sqrt(bY/(xKt4.length-1));
sx = Math.sqrt(bX/(xKt4.length-1));
s4 = r*sy/sx*8.3141;
// END
// Access data from SQL database
var realtemp = [];
var realdate = [];
var tidaldate = [];
var tidalheight = [];
select = 'SELECT date, temp FROM temp ORDER BY date;';
pool.query(select, (errDB, resDB) => {
```

```
if (errDB){
  console.log("Error while selecting Temp Data");
} else {
  select = 'SELECT date, height FROM tidal ORDER BY date;';
  pool.query(select, (errTi, resTi) => {
    if (errTi){
      console.log("Error while selecting Tidal Data");
    } else {
      for (var row in resDB.rows) {
        realtemp.push(parseFloat(resDB.rows[row].temp));
        realdate.push(resDB.rows[row].date);
      }
      for (var row in resTi.rows) {
        tidalheight.push(parseFloat(resTi.rows[row].height));
        tidaldate.push(resTi.rows[row].date);
      }
      console.log("Rendering...");
      var depthArray = [0, 5, 10, 15, 20, 30, 40, 50]
      // Send all calculated variables to front-end for graphing
      res.render('home',{
        res_cor_data: res_cor_data,
        res_raw_data: res_raw_data,
        temp_data: temp_data,
        time_labels: time_labels,
        day_labels: day_labels,
        temp1: temp1,
        temp2: temp2,
        temp3: temp3,
        temp4: temp4,
        realtemp: realtemp,
        realdate: realdate,
        tidaldate: tidaldate,
        tidalheight: tidalheight,
        t1: t1,
        t2: t2,
        t3: t3,
        t4: t4,
        yLn: yLn,
        xKt1: xKt1,
        xKt2: xKt2,
        xKt3: xKt3,
        xKt4: xKt4,
        s1: s1,
        s2: s2,
        s3: s3,
        s4: s4,
        specNumDefault: req.body.specimen,
        depthDefault: depthArray[req.body.depth],
        thNumDefault: req.body.thermistor,
        aaDefault: req.body.ae,
        ttDefault: req.body.tt,
```

```
              });
            }
          })
        }
      })
    });
});

// Setting application to run
app.listen(process.env.PORT || 3000);
```

13. Full Web page