PROJECT — DIGITAL SOLUTION (IA2)

by

CALEB MURPHY

DIGITAL SOLUTIONS

(Year 12)

HIGHFIELDS STATE SECONDARY COLLEGE

(Highfields)

May 2020

© CALEB MURPHY, 2020

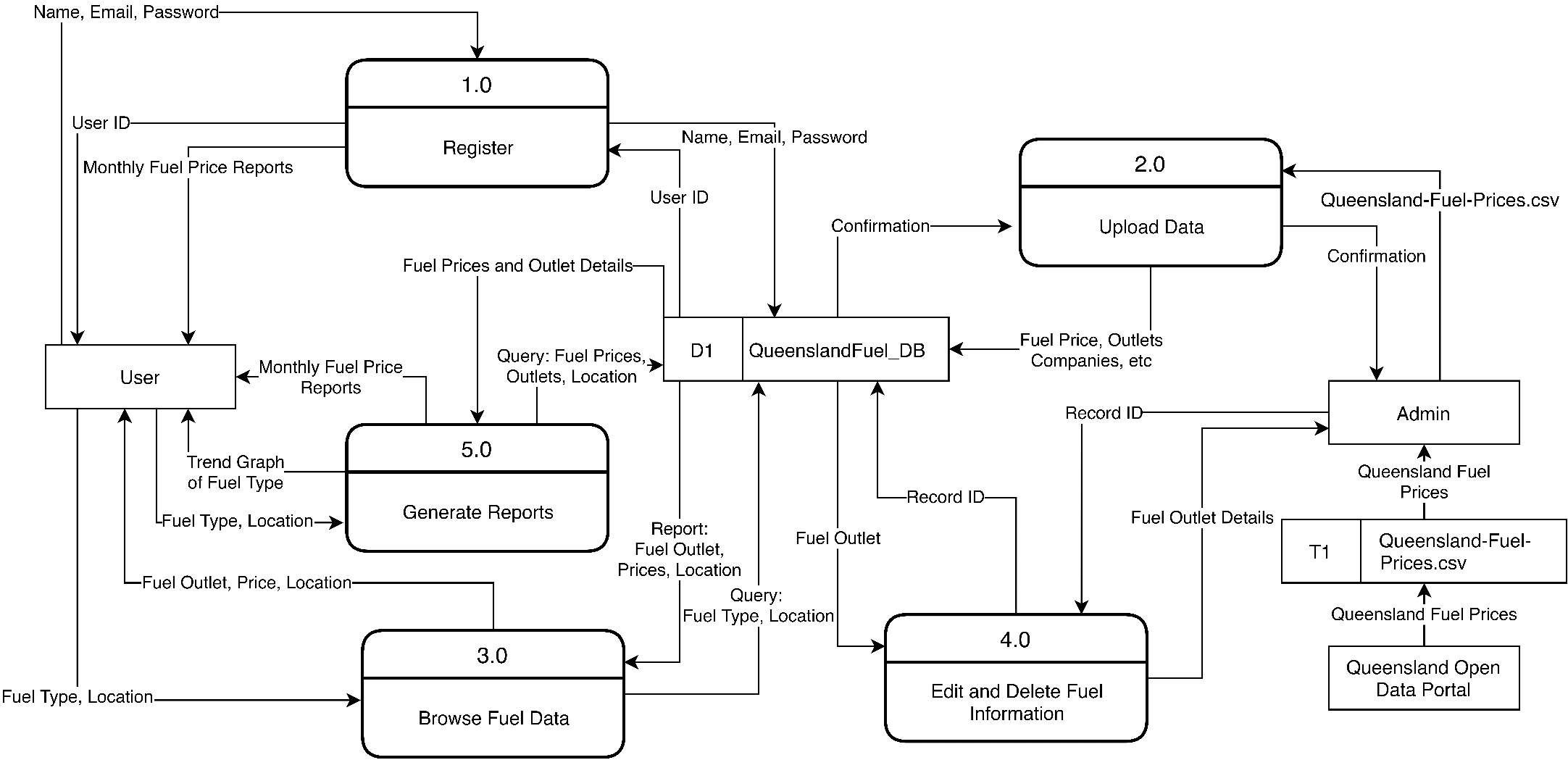
# Exploration of the problem to identify requirements and criteria

# Mind Map

Figure 1: Mind Map

# Development

## Prescribed Criteria, Self-Determined Criteria and Data Flow Diagram



### Requirements

#### Prescribed

* Provide an admin required portal for admirative staff to read and write uploaded data
* Enable database to be updated with new data from the CSV file uploaded
* Allow an automated email to be sent to registered users
* Allow users to view individual fuel outlets that display line graphs to give the user a deeper insight into fuel trends at the specific outlet.
* Contain visual indicators of rising and falling fuel prices
* Include an algorithm to read and write data to the database from an upload CSV file from the administrator.
* Allow users to view the website and data without registering an account.
* Contain the functionality for users to register an account with the website.
* Allow users to view generalised data for a holistic view of the data. (i.e. Display average prices in a certain suburb OR Cheapest suburbs in a state)
* Follow the Web Accessibility Guidelines

#### Self-Prescribed

* Incorporate accessibility through high contrasting colours between the foreground and background.
* Incorporate utility through web availability between multiple platforms. i.e. PC, Tablet, Mobile.
* Incorporate effectiveness through granting users access to specific search filtering when browsing data.
* Incorporate learnability through universal icons to represent functions of the website. i.e. A gear icon for settings.
* Incorporate accessibility and effectiveness through alt text on image and shortcut keys to move about the webpage.
* Incorporate effectiveness through displaying the cheapest fuel prices
* Incorporate effectiveness and learnability through page links being located on the top-left portion of the website for quick and intuitive access.
* Incorporate learnability and repetition through consistent navigation bars being implemented through all the pages of the website.
* Incorporate tone to highlight more important parts of the webpage.
* Incorporate balance and space in the form of white space to bring the users attention to more important parts of the webpage. This is also used in conjunction with proximity.
* Incorporate colour in conjunction with learnability by using different colours to represent meaning. i.e. A gold icon to represent a low price in fuel.

Figure 2: Level 0 Data Flow Diagram (DFD) - Fuel Analysis Tool

The prescribed criteria were generated from the client’s requirements (see mind map) and analysis of the technical proposal. These criteria are set as checkpoints for the developer to include in the final design of the prototype.

The Self-Prescribed criteria was created by following the Web-Accessibility-Guidelines, including the usability principles. Visual communication principles were also incorporated under the usability principle itself.

The admin of the website, on a monthly bases, would upload a CSV file from the Queensland Open Data Portal to the website where the data would be uploaded into the database. The admin is also given the capability to edit and delete records in the database due to anomalies or incorrect data. When a user enters the site, they are met with a login page. Here, the user can log into the website, if they have an account, or choose to register an account if they do not already have one. If the user wishes to view the website without creating an account, they are given the option to continue as a guest. Once the user has chosen an option, they are taken to the home page where the user is able to view fuel outlets close to the user on a map and filter the type of fuel stops that they see (i.e. Petrol vs Diesel). This page acts as a hub for the user to access the other pages of the site such as the browse page. In this page, the user browses fuel data based on filters that were given by the user. The user is then given the option to view the record, outlet or fuel type in greater detail. All these fields will be able to be viewed in greater detail but each showing different data. E.g. Suburb in grater detail will show the average fuel price for each month in that suburb. Outlet will show the fuel price over time.

## Data Source

Figure 3: Queensland Data Portal, Fuel Price Update (January 2020)

The data, attained through the Queensland Data Portal (2018-2020), displays mandatory monthly updates for fuel prices at each outlet. These updates, however, are unnormalized and exported in a .csv flat-file format, making the data inefficient to gather information on a large scale such as trends in fuel prices. This file format also presents the issue of redundancy especially towards information such as fuel types and companies.

These issues fall under the blanket term of Data organisation, as the integrity of data is lost through redundancy and inability to update the data efficiently and effectively. Without relational links between data fields, filtering and browsing data becomes increasingly difficult for the user. Through the data being located in multiple files, not a central database, the user is restricted in what data they are able to view at once. Furthermore, the price is recorded multiple times each outlet updates the fuel price, so a relationship between the outlet field, and the price field must be established.

Data inconvenience can also be solved algorithmically in some circumstances. For example, in each dataset the first row displays headings and can be described as anomalies, can be overlooked when implementing the data. The exportation on data can produce duplicate data, which may prove ineffective. Interestingly, the transaction data-time as well as the outlet acts a composite key that can be used to algorithmically check whether the data is a duplicate piece of data. Finally, data security is essential and in order maintain the safety of the data uploaded, it is recommended that a role-based authentication system be implemented for uploading, updating and deleting data.

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Not Null | Purpose |
| SiteID | nvarchar |  | Differentiates each piece of data and makes them unique |
| Site\_Name | nvarchar |  | The fuel outlets company and location |
| Site\_Brand | nvarchar |  | The fuel outlets company |
| Sites\_Address\_Line\_1 | nvarchar |  | The fuel outlets local address (street and number) |
| Site\_Suburb | nvarchar |  | The fuel outlets suburb name |
| Site\_State | nvarchar |  | The state in which the fuel outlet is located |
| Site\_Post\_Code | int |  | Stores the post code of the fuel outlet |
| Site\_Latitude | float |  | The latitudinal location of the fuel outlet |
| Site\_Longitude | float |  | The longitudinal location of the fuel outlet |
| Fuel\_Type | nvarchar |  | The type of fuel that is being described in the piece of data |
| Price | Int |  | The recorded price of fuel |
| TransactionDateutc | datetime |  | The data and time at which the data was recorded. |

## Relational Database

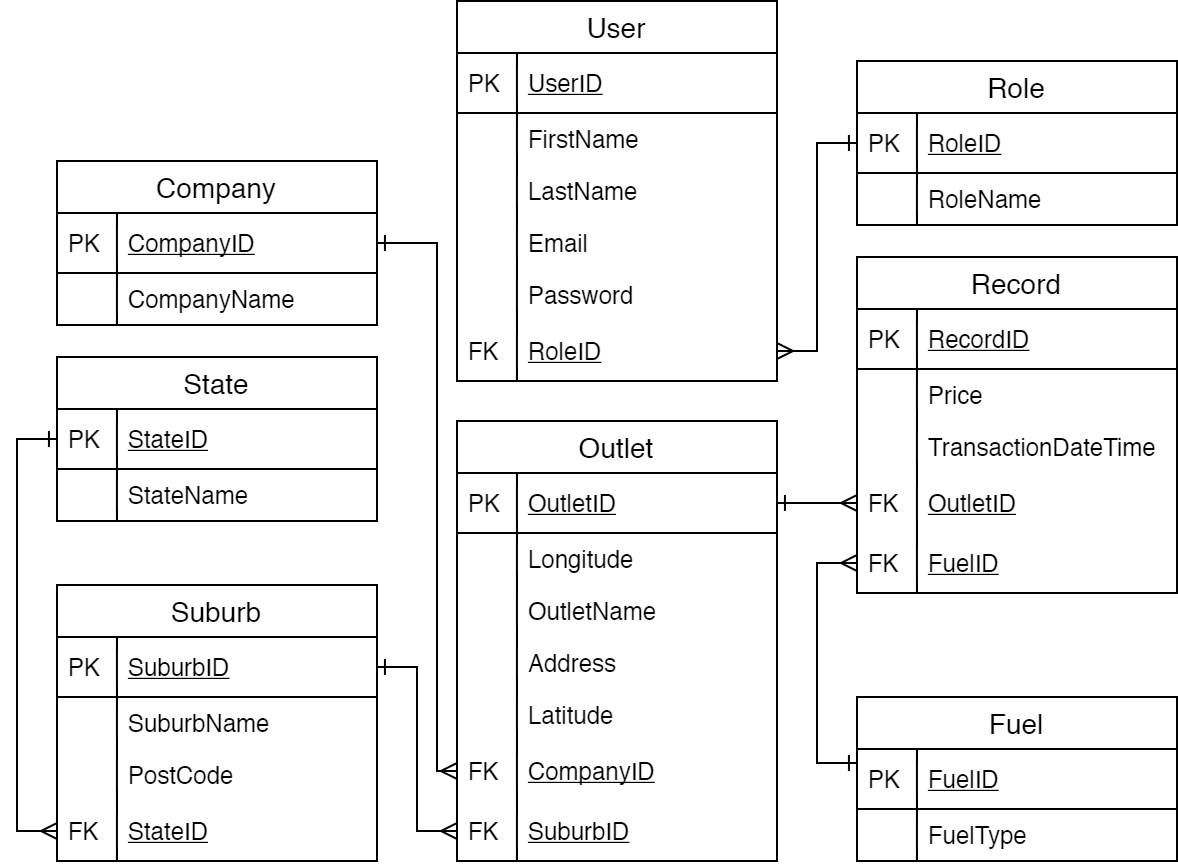


Figure 5: Entity Relationship Diagram (ERD)

CREATE TABLE [dbo].[Record] (

[RecordID] INT IDENTITY (1, 1) NOT NULL,

[Price] INT NOT NULL,

[TransactionDateTime] DATETIME NOT NULL,

[OutletID] INT NOT NULL,

[FuelID] INT NOT NULL,

CONSTRAINT [PK\_dbo.Record] PRIMARY KEY CLUSTERED ([RecordID] ASC),

CONSTRAINT [FK\_dbo.Record\_dbo.Fuel\_FuelID] FOREIGN KEY ([FuelID]) REFERENCES [dbo].[Fuel] ([FuelID]) ON DELETE CASCADE,

CONSTRAINT [FK\_dbo.Record\_dbo.Outlet\_OutletID] FOREIGN KEY ([OutletID]) REFERENCES [dbo].[Outlet] ([OutletID]) ON DELETE CASCADE

);

CREATE TABLE [dbo].[Outlet] (

[OutletID] INT IDENTITY (1, 1) NOT NULL,

[OutletName] NVARCHAR (140) NOT NULL,

[Address] NVARCHAR (MAX) NULL,

[Latitude] FLOAT (53) NOT NULL,

[Longitude] FLOAT (53) NOT NULL,

[CompanyID] INT NOT NULL,

[SuburbID] INT NOT NULL,

[Company\_CompanyID] INT NULL,

[Suburb\_CompanyID] INT NULL,

[Company\_CompanyID1] INT NULL,

CONSTRAINT [PK\_dbo.Outlet] PRIMARY KEY CLUSTERED ([OutletID] ASC),

CONSTRAINT [FK\_dbo.Outlet\_dbo.Company\_Company\_CompanyID] FOREIGN KEY ([Company\_CompanyID]) REFERENCES [dbo].[Company] ([CompanyID]),

CONSTRAINT [FK\_dbo.Outlet\_dbo.Company\_Suburb\_CompanyID] FOREIGN KEY ([Suburb\_CompanyID]) REFERENCES [dbo].[Company] ([CompanyID]),

CONSTRAINT [FK\_dbo.Outlet\_dbo.Company\_Company\_CompanyID1] FOREIGN KEY ([Company\_CompanyID1]) REFERENCES [dbo].[Company] ([CompanyID]),

CONSTRAINT [FK\_dbo.Outlet\_dbo.Suburb\_SuburbID] FOREIGN KEY ([SuburbID]) REFERENCES [dbo].[Suburb] ([SuburbID]) ON DELETE CASCADE

);

CREATE TABLE [dbo].[Company] (

[CompanyID] INT IDENTITY (1, 1) NOT NULL,

[CompanyName] NVARCHAR (140) NOT NULL,

CONSTRAINT [PK\_dbo.Company] PRIMARY KEY CLUSTERED ([CompanyID] ASC)

);

The Fuel Price Data (.csv), retrieved from the Queensland Open Data Portal, was received in first normal form. Each cell of the .csv file contained only one piece of data and there were no repeating columns.   
The solution for solving these issues involved extracting the relevant for the outlets – Names, Addresses, Price, for example – while disregarding other information that doesn’t add value. For example, the *siteID* was removed as the outlet name acts as a unique identifier.   
As each line of data in the file was related to a specific record but an outlet could submit multiple records, it was clear that a record table was needed to be created. Furthermore, a link between the outlet table and the record table needed to also be established. In this case, there was a one-to-many relationship. This is due to an outlet being able submit multiple records of fuel prices but a record can only have one outlet.  
To achieve this relationship, a foreign key to the outlet is added to the records table. This relationship is then repeated between the suburb and outlets as well as the company and outlet, satisfying second normal form. Furthermore, to satisfy third normal form, record specific results are attached to the record entity to ensure that no non-key field is dependant on any other non-key field.  
Finally, the entities for the ASP.NET identity were created to allow a role base system that stored users email and passwords (Microsoft2019. The one-to-many relationship between users and role is a necessary feature to allow authentication of staff when accessing sensitive or high-risk data.

The sample DDL commands shown above are written in Transact-SQL (T-SQL), which is used in Microsoft Visual Studio SQL Server Express relational database management system (RDBMS).  
Each datatype is designed for their specific field, while containing additional attributes for such an auto-increment for more important tasks such as primary keys.  
Each datatype in the model is design to tailer to their specific, including additional attributes such as an autoincrement or validation for important variables. For example, in the *Outlet* table, the OutletID is set to an IDENTITY and cannot be null. Furthermore, the OutletName cannot be more than 140 character. A sample SQL insertion command is located below.

INSERT INTO Record (Price, TransactionDataTime, OutletID, FuelID)

VALUES (1254, ‘03/04/2018’, 3, 1);

Figure 4: Sample DDL commands for creating the database structure

## User-Interface (UI) Wireframes and Site Map

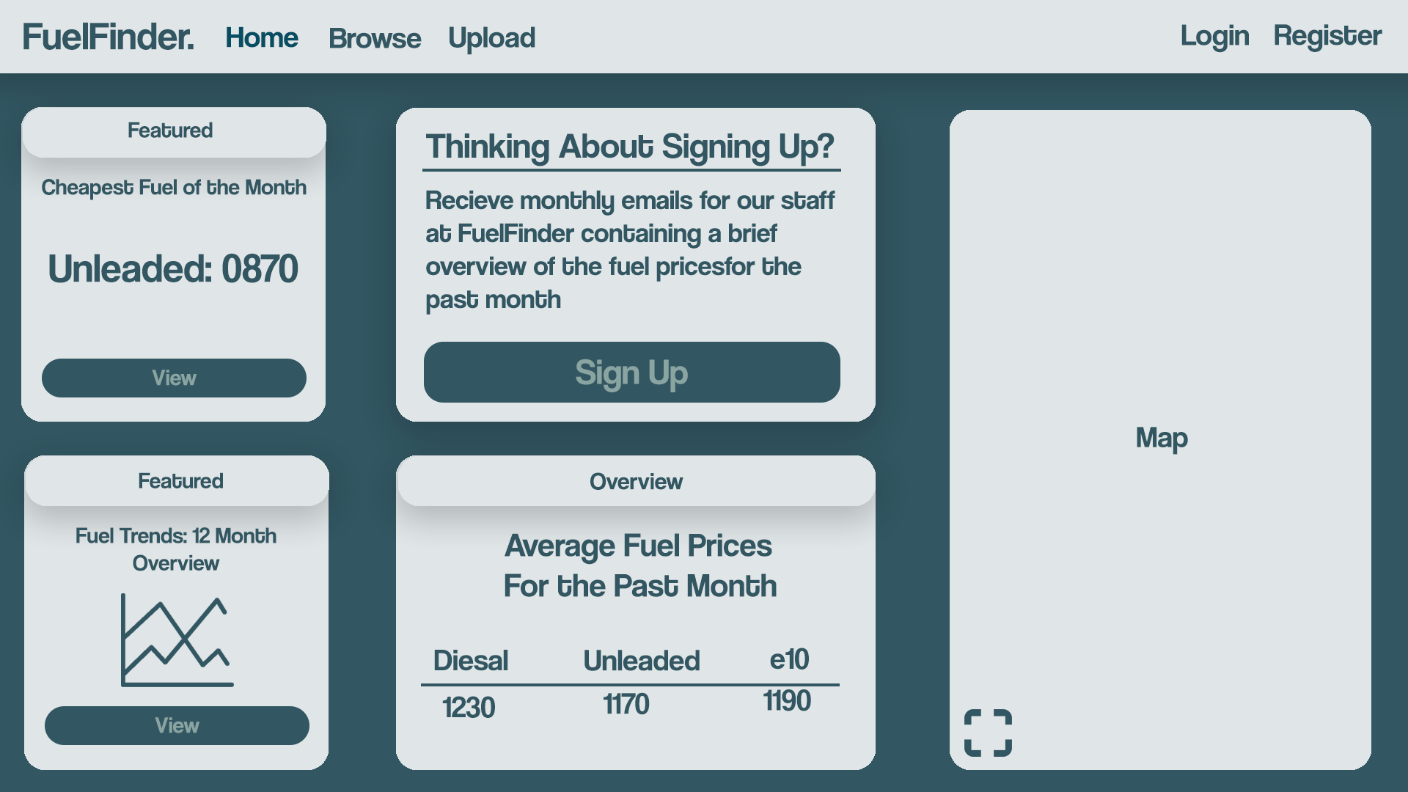
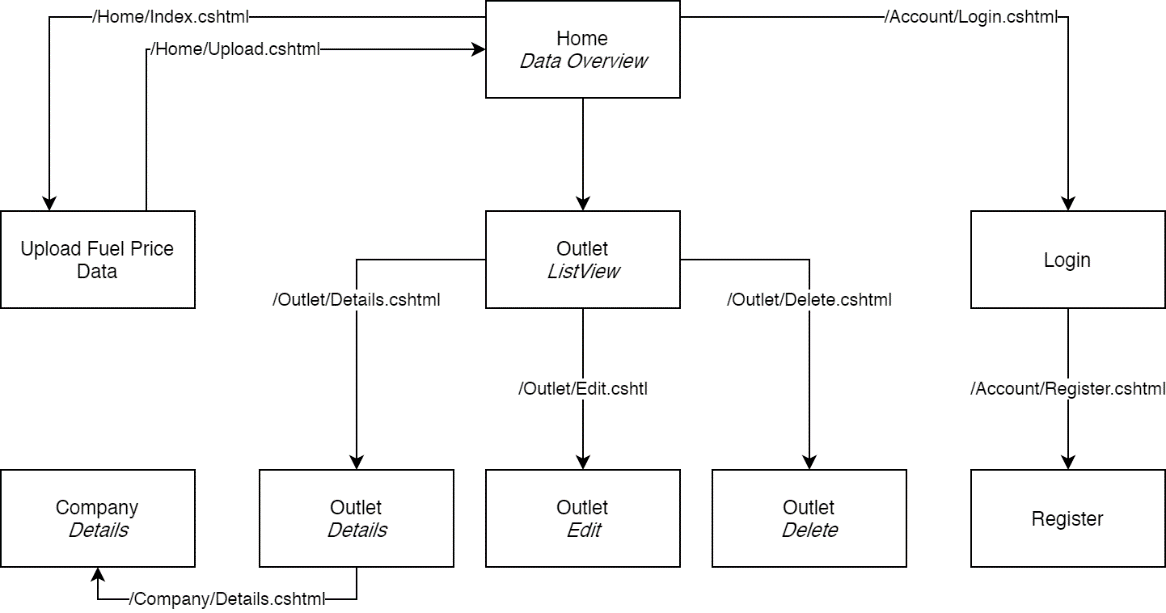


Figure 6: Site Map



Figure 7: Outlet Individual View

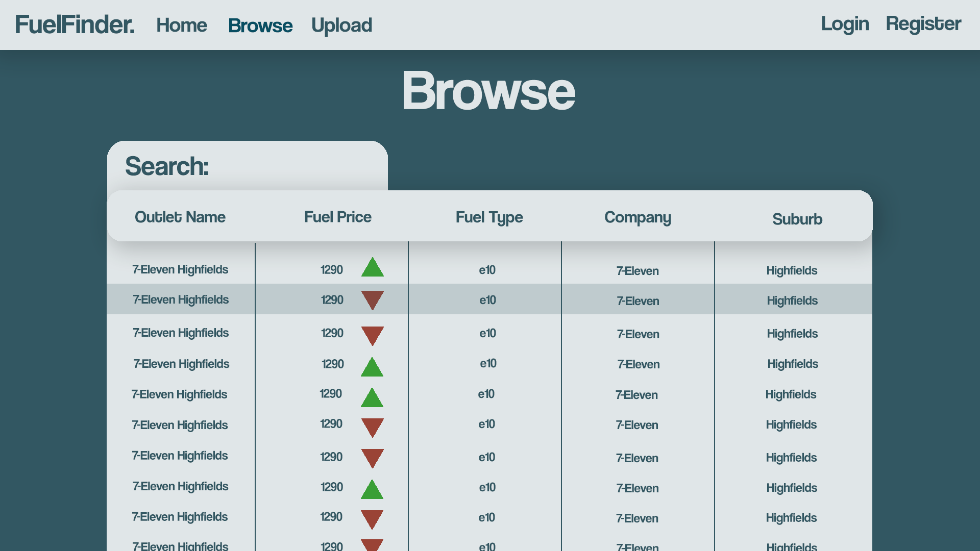


Figure 8: Record List View

Key components, such as the cheapest fuel of the month and the average fuel prices, are included on the front page of the website (Figure 9) and allow for effective viewing of the webpage. The users are able to quickly gauge the average fuel prices of the month, with links to the outlet and company associated with the cheapest fuel. Furthermore, they are able to view all outlets on a map to quickly gauge the number of fuel outlets close by. The encapsulation of data allows for effective viewing of the webpage. The user is able to understand the data being shown through reading the title rather than the data itself. The hierarchy of information incorporates learnability into the site as the most important information, featured tabs, are left aligned to support the effectiveness of the site. This is important as users tend to scan information from top-to-bottom and left-to-right (Queensland Government, 2016). Due to the encapsulation of data, the information can be quickly implemented for mobile viewing and mitigate scrolling to view the data. Each of these structures were influenced by the elements of visual communication. For instance, space has been implemented to clearly differentiate between components of the webpage. When images are implemented, alt text will be implemented for accessibility.

Below is an example query that finds the cheapest fuel price on the month.

SELECT Fuel.FuelType, Record.Price  
FROM Record  
INNER JOIN Outlet ON Outlet.OutletID = Record.RecordID  
INNER JOIN Fuel ON Fuel.FuelID = Record.FuelID  
ORDER BY Record.Price  
LIMIT 1;

*Figure 7* focusing on not cluttering the user and only show what is necessary. This implementation of space enhances the effectiveness of the site. The button colours for fuel prices support learnability, using high contrasting colours to clearly differentiate between the different fuel types. Shadows were used to further contrast the foreground elements from the background. The following query was used to populate the fuel trend graph of the previous 10 months:

SELECT \*  
FROM Record  
INNER JOIN Outlet ON Outlet.OutletID = Record.RecordID  
INNER JOIN Fuel ON Fuel.FuelID = Record.FuelID  
GROUP BY Fuel.FuelType  
ORDER BY Record.TransactionDateTime  
LIMIT 10;

Accessibility is evident in the design of the webpage (Figure 8), through high contrasting colours representing the trend in the fuel prices the user can view data quickly and effectively. Form is established in the dataset through the implementation of data-tables for each view on the webpage. This allows for filtration, pagination and sorting of data. The consistent background colours, fonts and highlights support learnability when it comes to visually impaired users. These conventions are repeated across each UI. Below is a sample query for populating the database.

SELECT \*  
FROM Record  
INNER JOIN Outlet ON Outlet.OutletID = Record.RecordID  
INNER JOIN Fuel ON Fuel.FuelID = Record.FuelID  
INNER JOIN Company ON Company.CompanyID = Outlet.CompanyID  
INNER JOIN Suburb ON Suburb.SuburbID = Outlet.SuburbID;

Figure 9: Home View

## Algorithms

**BEGIN** Main (arrNum)

**INITIALISE** arr = arrNum

**CALL** quickSort(arr, 0, arr.length - 1)

**END**

**BEGIN** quickSort(arr, left, right)

**INSTANTIATE** pivot

**IF** left < right **THEN**

**SET** pivot = CALL Partition(arr, left, right)

**IF** pivot > 1 **THEN**

CALL quickSort(arr, left, pivot - 1)

**ENDIF**

**IF** pivot + 1 < right **THEN**

**CALL** quickSort(arr, pivot + 1, right)

**ENDIF**

**ENDIF**

**END**

**BEGIN** Partition (arr left, right)

**INSTANTIATE** pivot = arr[left]

**WHILE** true DO

**WHILE** arr[left] < pivot **DO**

**SET** left = left + 1

**ENDWHILE**

**WHILE** arr[right] > pivot **DO**

**SET** right = right - 1

**ENDWHILE**

**IF** left < right **THEN**

**INSTANTIATE** temp = arr[right]

**SET** arr[right] = arr[left]

**SET** arr[left] = temp

**ELSE**

**RETURN** right

**ENDIF**

**ENDWHILE**

**END**

**BEGIN** GetSampleRecords(iNumRecord)

**DECLARE** rand = new Random()

**SET** Records = new List<Record>()

**DECLARE** i = 0

**FOR** i TO iNumRecord STEP 1 DO

**DECLARE** iRandom = rand.Next(1, \_db.Records.Count());

**DECLARE** oRecord = **SELECT** \*

**FROM** DBRecord

**WHERE** RecordID == iRandom

**LIMIT** 1;

**DECLARE** nAmount = **SELECT** Count(\*)

**FROM** RecordList

**WHERE** RecordID == oRecord.RecordID

**IF** nAmount != 0 THEN

**CALL** Records.Add(oRecord);

**ELSE**

**SET** i = i - 1

**ENDIF**

**ENDFOR**

**END**

**BEGIN** FindFuelTypePrices(id)

**IF** id != null **THEN**

**DECLARE** aRecPr = **SELECT** Record.Price

**FROM** Outlet

**INNER** **JOIN** Fuel ON Fuel.FuelID == Outlet.FuelID

**INNER** **JOIN** Record ON Record.OutletID == Outlet.OutletID

**WHERE** Outlet.OutletID == id

**GROUP** **BY** Fuel.FuelType

**ORDER** **BY** TransactionDateTime

**LIMIT** 12

**RETURN** aRecPr

**ENDIF**

**END**

# Generation

When a user is looking at fuel prices for a specific outlet, they only wish to see prices for their fuel type. To allow for this, the algorithm above (Figure 10) finds the latest 12 fuel prices for the specified outlet grouping by each fuel type.

This was achieved through first checking that the id inputted was not a null value. If not, then the outlet with that id is found and then selecting the latest 12 fuel prices for each fuel type and places them in an array that is returned to the view to be displayed to the user.

This process takes place each time the user views an outlet in the details view. These values are used to populate a line graph from Chat.js that is then shown to the user.

The client requests that the user be able to browse records and this algorithm delivers the user the requested number of records (Figure 11). Note that these are all random, the algorithm has the capacity to further filter the records that are shown, but for this application, random records are suitable. The user requests a certain number of records and then this variable is then placed into a for-loop. For each iteration, a new random value is generated between 1 and the number of records. A record is then found with this value as an id. A check is then completed to make sure the same record isn’t chosen twice. If there is no sign of the record in the list, the record is added to the list and the loop continues. If the same record is chosen twice, then that iteration of the loop is ran again. This ensures there are exactly the amount of records the user requested.

The algorithm above demonstrates an abstract implementation of the quick sort for any 1-dimensional array of numbers. There a various application of such a sort including ordering the list view and ordering the fuel prices by transaction data. The pseudocode consists of a 1-dimensional array of numbers being input and then a recursive function is used to iteratively sort the array.

The function first sets left and right positions as the start and end of the array. Then the Partition function is called which sets the pivot position to the start point. The function then moves through array values from start point checking whether each of the values are less than the pivot. If they aren’t, then the loop is stopped and the values position in the array is recorded. The function then moves through array value from end point checking whether each of the values are more than the pivot, if they aren’t, then the loop is stopped and the values position in the array is recorded. The two values are then switched in the array, making the smaller number closer to the start of the array and the larger value closer to the end. The pivot position is then set to the position of the smaller value. Then the pivot is checked to see whether the position is the first in the array so that the right position can be set to pivot – 1. If this is not the case, than it is then checked to see whether it is smaller than the right position minus 1. If this is the case, the function is called again with the starting point being 1 larger as it is known that the first value is the smallest. If this is not the case, the quick sort has stopped swapping values and has finished, so the function isn’t called again.

Figure 10: Fuel Prices by Category Algorithm

Figure 11: Unique Number of Record Algorithm

Figure 12: Quick Sort Algorithm

## Screenshots with Annotations

Figure 13: Outlet Edit Page

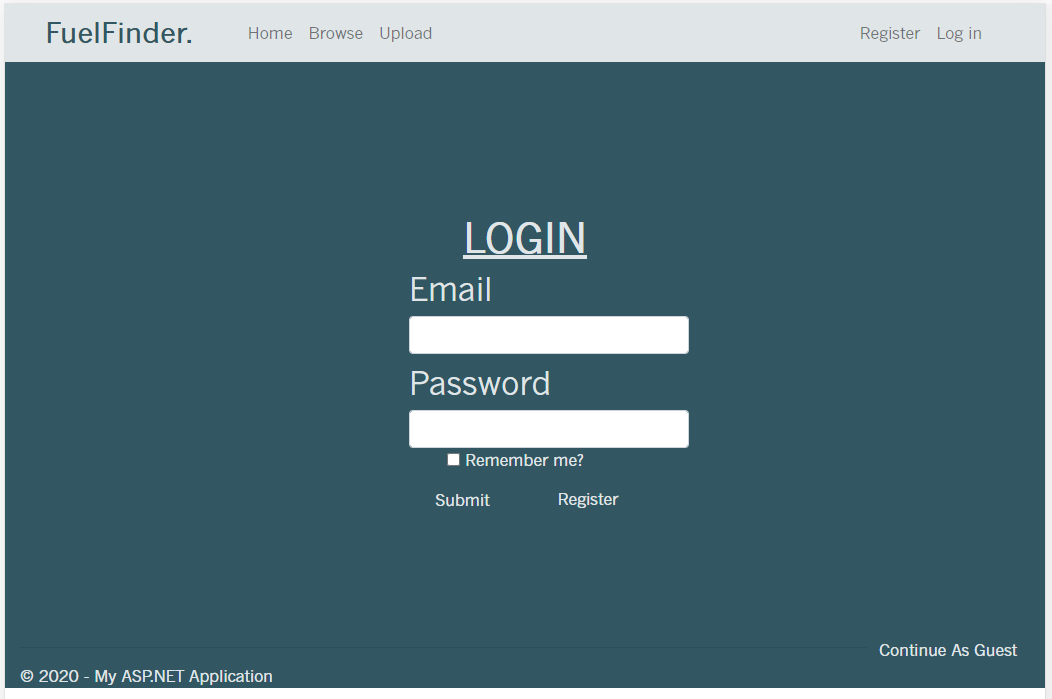
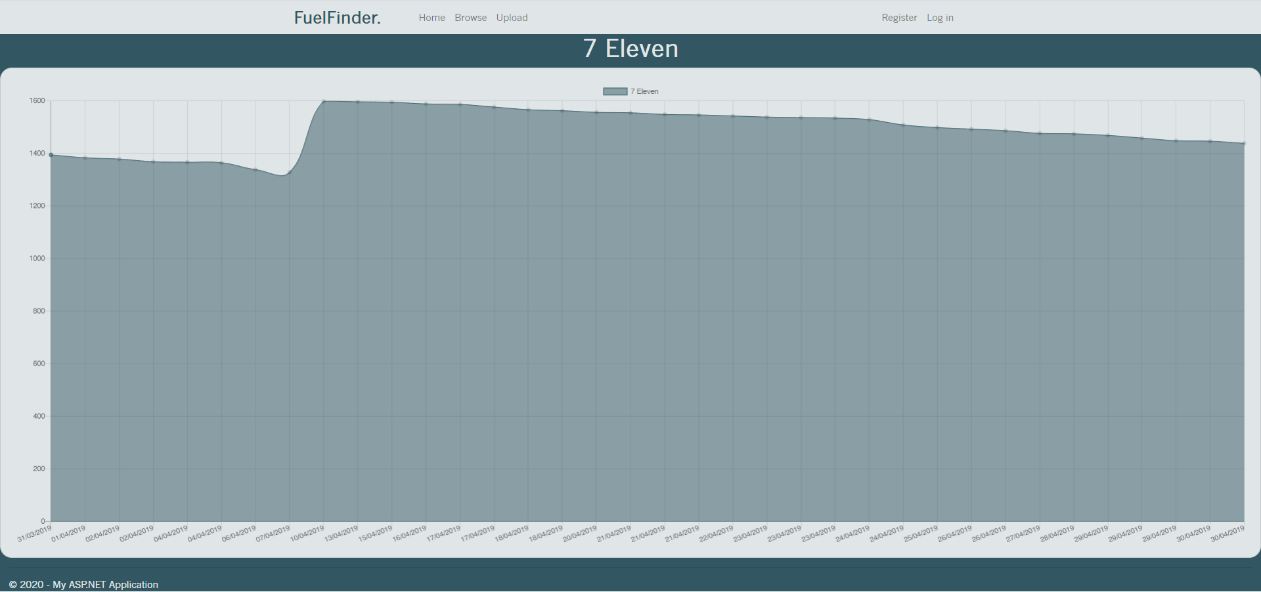
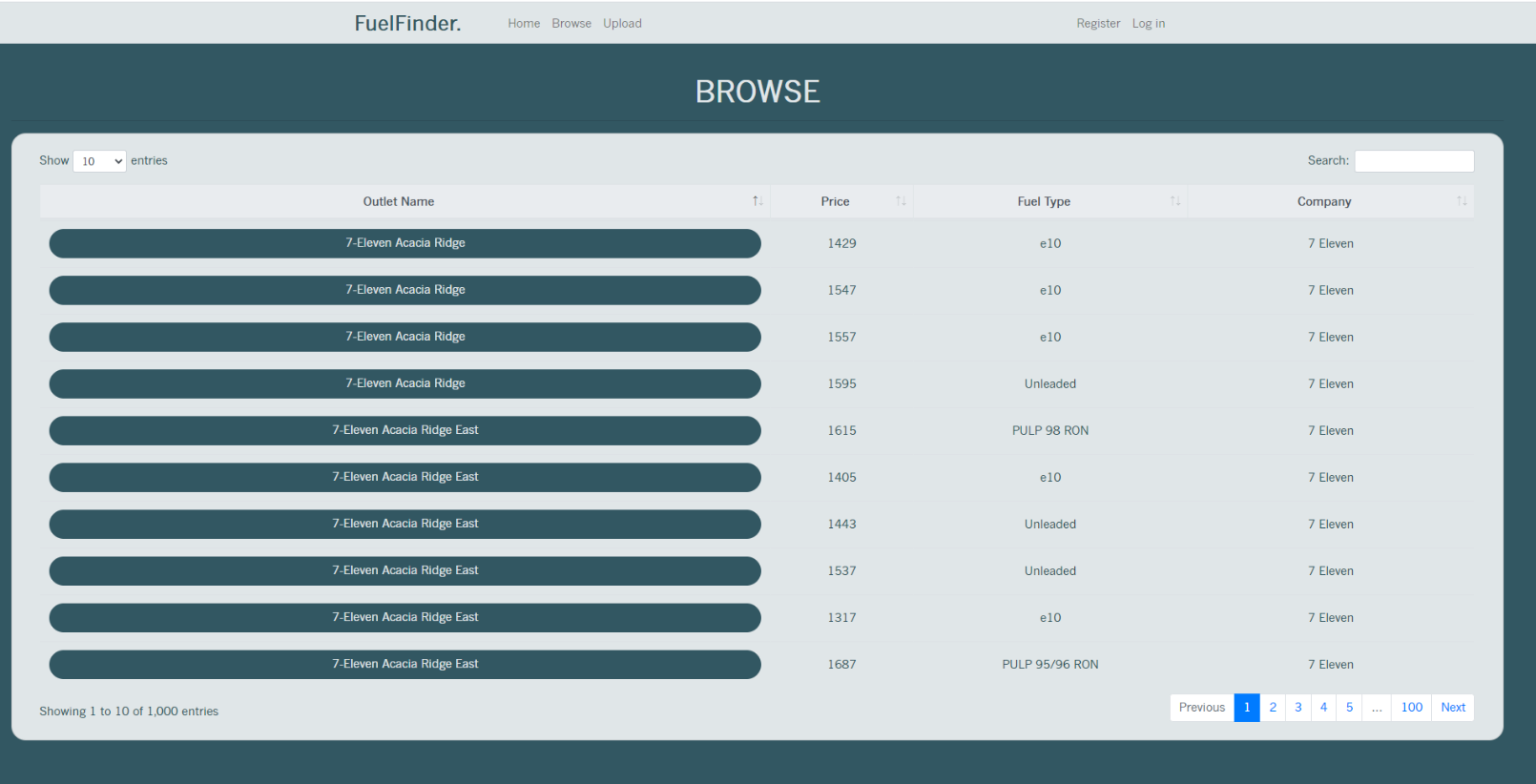
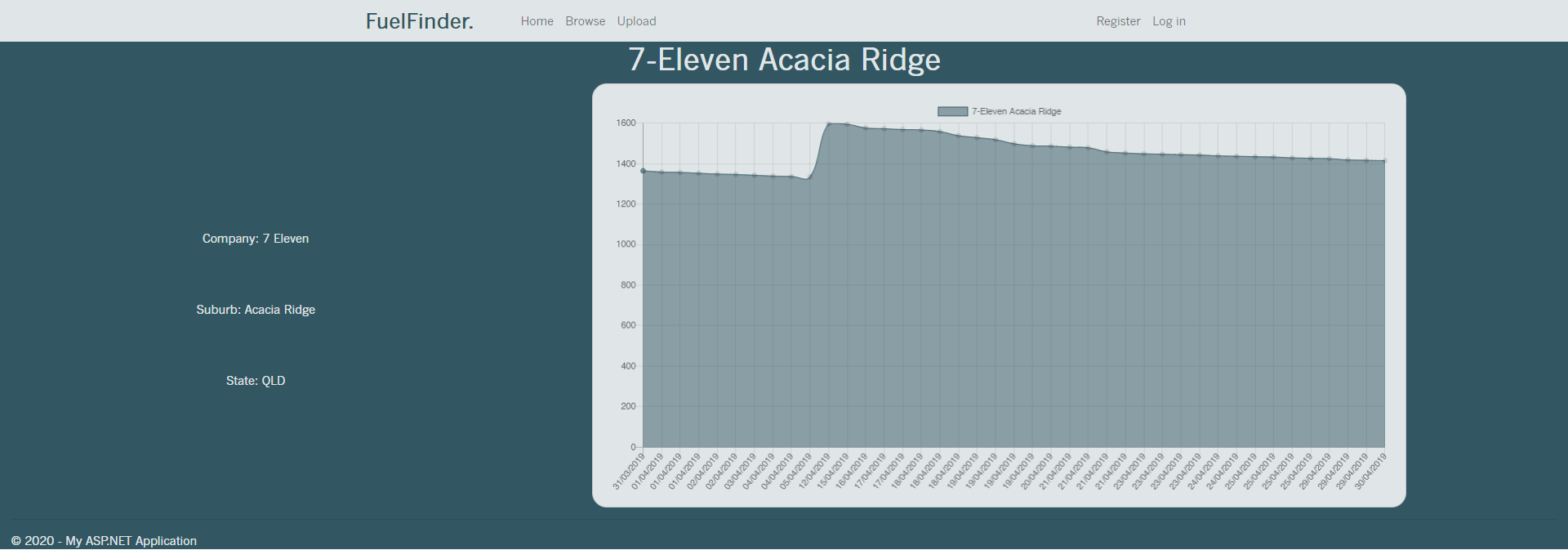


Figure 14: Company Details Page

Figure 15: Login Page



**~/Browse/Index.cshtml:** This screenshot demonstrates the browse page of the webpage and shows the records of the previous month. The list is initially in alphabetical order of the outlet but the user can change what column is ordered.

**~/Outlet/Details.cshtml:** This screenshot demonstrates the details page of the outlet and shows all the prices from the past month in a line graph. Hovering over a specific point will show the exact price.

**~/Company/Details.cshtml:** This screenshot demonstrates the details page of the company and shows all prices from the past month in a line graph.

**~/Outlet/Create.cshtml:** This screenshot demonstrates requirements when creating an outlet (Safety). This page was also viewed on an iPhoneX

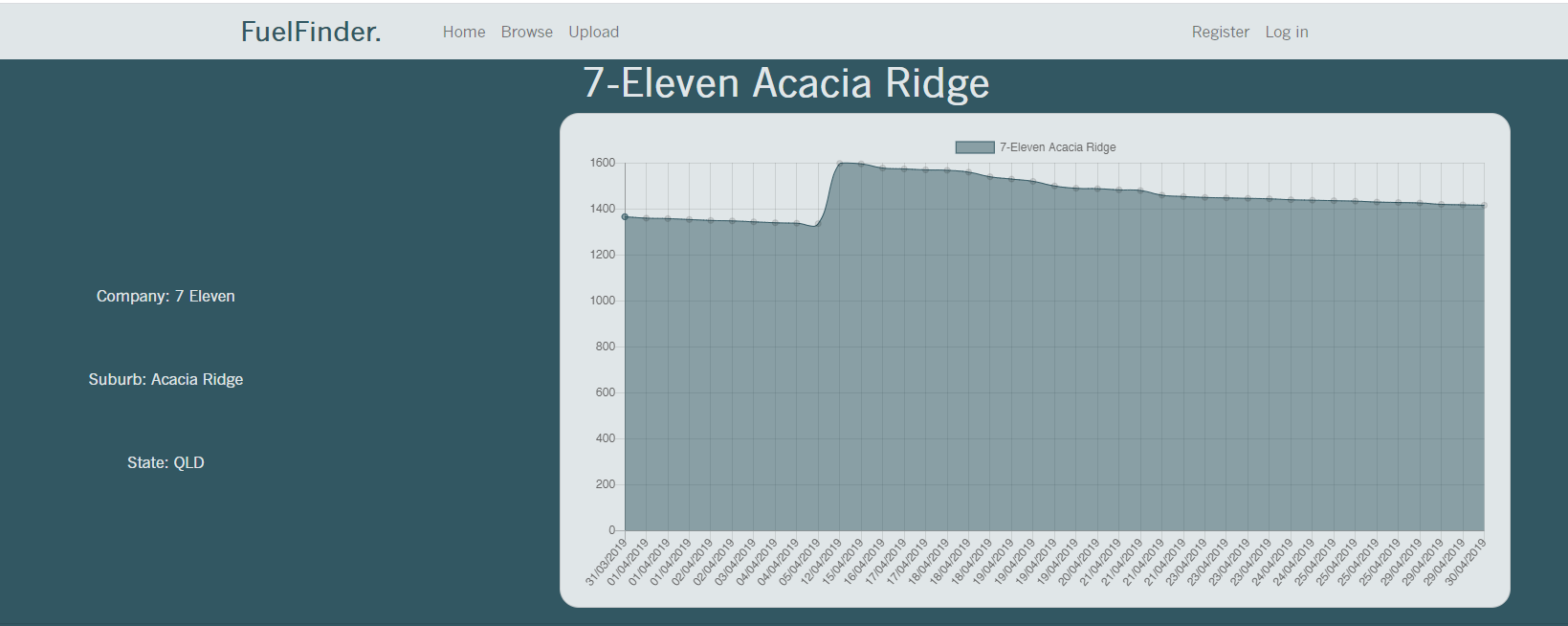
**~/Account/Login.cshtml:** This screenshot demonstrates the login page on an iPad Screen (Responsive Design).

Figure 16: Outlet Details Page

Figure 17: Browse Page for Records

## Model-View-Controller (MVC) Architectural Pattern

### View(~Views/Outlet/Details.cshtml)



The MVC format is the name given to the format where a Model, View and Controller are used. These three components work together to produce an interactive and function webpage. The Model handles the structure of each of the tables in the database. The View controls what the user sees such as the layout of webpage and the colours. The controller is the bridge between the two components and sends and receives messages from the Model to the View and vice versa.

Pascal casing for variable naming is used as the class id relies on the name of the class which uses Pascal casing as a default. So, to keep a consistent naming convention throughout the project, Pascal casing was used.

The controller uses lambda expressions to find the selected ID that needs to be used to view the page. Once found, the view model is passed through to the view.

### Controller (~/Controllers/OutletController.cs)

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Outlet outlet = db.Outlets.Find(id);

if (outlet == null)

{

return HttpNotFound();

}

else

{

//Finding the outletViewModels

OutletViewModel outletViewModel = new OutletViewModel(db, id);

return View(outletViewModel);

}

}

### Model (~/Models/Outlet.cs)

public class Outlet

{

public int OutletID { get; set; }

[Required]

[StringLength(140, ErrorMessage = "The name cannot exceed 140 character in length")]

[Display(Name = "Fuel Outlet Title")]

public string OutletName { get; set; }

public string Address { get; set; }

public double Latitude { get; set; }

public double Longitude { get; set; }

public int CompanyID { get; set; }

public virtual Company Company { get; set; }

public int SuburbID { get; set; }

public virtual Suburb Suburb { get; set; }

public virtual ICollection<Record> Record { get; set; }

}

Figure 18: View of ~/Outlet/Details.cshtml from a wide screen monitor

## 6 x A4 Pages of Code with Annotations

//Same as the company but for the state.

Outlet oOutlet = \_db.Outlets.FirstOrDefault(o => o.OutletName == sOutletName);

if (oOutlet == null)

{

oOutlet = new Outlet();

oOutlet.OutletName = sOutletName;

oOutlet.Address = sOutletAddress;

oOutlet.Latitude = Convert.ToDouble(sLatitude);

oOutlet.Longitude = Convert.ToDouble(sLongitude);

//Links the outlet to the company through a foreign key

oOutlet.CompanyID = oCompany.CompanyID;

oOutlet.SuburbID = oSuburb.SuburbID;

oOutlet.Company = oCompany;

oOutlet.Suburb = oSuburb;

\_db.Outlets.Add(oOutlet);

\_db.SaveChanges();

oCompany.Outlet.Add(oOutlet);

oSuburb.Outlet.Add(oOutlet);

}

//As each record is assumed to be unique, no measures are used to check for duplicates.

Record oRecord = new Record();

oRecord.Price = Convert.ToInt32(sPrice);

oRecord.TransactionDateTime = dtTransactionDateTime;

oRecord.FuelID = oFuel.FuelID;

oRecord.OutletID = oOutlet.OutletID;

\_db.Records.Add(oRecord);

\_db.SaveChanges();

oFuel.Record.Add(oRecord);

oOutlet.Record.Add(oRecord);

//Batching logic for efficiency

if (i % 100 == 0)

{

\_db.Dispose();

\_db = new ApplicationDbContext();

}

i++;

}

\_db.SaveChanges();

}

}

}

### Uploading Data (sample):

public class UploadViewModel

{

public HttpPostedFileBase UploadFile { get; set; }

public void ReadCSVDataToDB (string \_fileLocation, ApplicationDbContext \_db)

{

// Logic for reading the fuel database upon upload

//Store each record as an index in the array

string[] arrRecords = File.ReadAllLines(\_fileLocation);

int i = 0; //Batch Counter

//Go through each row, but skip the header

foreach(var row in arrRecords.Skip(1))

{

//Break the row into columns

string[] arrFuelDetails = Regex.Split(row, ",(?=(?:[^\"]\*\"[^\"]\*\")\*[^\"]\*$)");

//Record is the main entity

string sOutletID = arrFuelDetails[0];

string sOutletName = arrFuelDetails[1];

string sCompanyName = arrFuelDetails[2];

string sOutletAddress = arrFuelDetails[3];

string sSuburbName = arrFuelDetails[4];

string sStateName = arrFuelDetails[5];

string sPostCode = arrFuelDetails[6];

string sLatitude = arrFuelDetails[7];

string sLongitude = arrFuelDetails[8];

string sFuelType = arrFuelDetails[9];

string sPrice = arrFuelDetails[10];

DateTime dtTransactionDateTime = Convert.ToDateTime(arrFuelDetails[11]);

//Finds a compnay containing the sCompanyName in the database.

Company oCompany = \_db.Companys.FirstOrDefault(o => o.CompanyName == sCompanyName);

//If there isnt a company with that name, then create another company.

if(oCompany == null)

{

oCompany = new Company();

oCompany.CompanyName = sCompanyName;

\_db.Companys.Add(oCompany);

\_db.SaveChanges();

}

//Same as the company but for the state.

State oState = \_db.States.FirstOrDefault(o => o.StateName == sStateName);

if (oState == null)

{

oState = new State();

oState.StateName = sStateName;

\_db.States.Add(oState);

\_db.SaveChanges();

}

//Same as the company but for the fuel…

@model FuelFinder.ViewModels.FeatureViewModel

@{

ViewBag.Title = "Index";

}

<div class="row w-100" style="padding-top:2%; padding-bottom:5%">

<div class="col-sm-12 col-md-12 col-lg-12">

<h1 style="text-align:center; font-weight:900">BROWSE</h1>

<hr />

<div class="card home-box back-light">

<table id="featured" class="table">

<thead class="thead-light">

<tr>

<th class=" text-center align-middle">Outlet Name</th>

<th class=" text-center align-middle">Price</th>

<th class=" text-center align-middle">Fuel Type</th>

<th class=" text-center align-middle">Company</th>

</tr>

</thead>

<tbody>

@foreach (var record in Model.Records)

{

<tr style="color:#325762">

<td class=" text-center align-middle">

@Html.ActionLink(record.Outlet.OutletName, "Details", "Outlet", new { id = record.Outlet.OutletID }, new { @class = "btn cus-btn-drk" })

</td>

<td class=" text-center align-middle">

@record.Price

</td>

<td class=" text-center align-middle">

@record.Fuel.FuelType

</td>

<td class=" text-center align-middle">

@record.Outlet.Company.CompanyName

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

@section scripts

{

<script>

$(document).ready(function () {

$("#featured").DataTable();

})

</script>

}

### Upload Data (HttpGet and HttpPost):

//Get: Upload

[Authorize(Roles = "Admin")] //Only allow admin to access webpage

public ActionResult Upload()

{

return View();

}

[HttpPost]

public ActionResult Upload (UploadViewModel \_upload)

{

//Fail to upload?

if(\_upload.UploadFile == null)

{

return HttpNotFound();

}

//Createa a file name and path

string sFileName = Path.GetFileName(\_upload.UploadFile.FileName);

string sPath = Path.Combine(Server.MapPath("~/Content/Upload"), sFileName);

//Save the file to the server

\_upload.UploadFile.SaveAs(sPath);

\_upload.DeleteAllRecords(db);

//Execute the ReadCSVToDB method

\_upload.ReadCSVDataToDB(sPath, db);

//Redirectthe user to the home index

return RedirectToAction("Index");

}

### Outlet Controller’s HttpGet for the Details Page:

// GET: Outlet

public ActionResult Index()

{

return View(db.Outlets.ToList());

}

// GET: Outlet/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Outlet outlet = db.Outlets.Find(id);

if (outlet == null)

{

return HttpNotFound();

}

else

{

//Finding the outletViewModels

OutletViewModel outletViewModel = new OutletViewModel(db, id);

return View(outletViewModel);

}

}

### ~/Home/Index.cshtml:

@model FuelFinder.ViewModels.HomeViewModel

@{

ViewBag.Title = "Home Page";

}

<div class="row w-100" style="padding-top:1.125%">

<div class="col-lg-3 home-box">

<div class="card back-light h-50 shadow-lg" style="border-radius:20px">

<div class="card-header shadow" style="height:15%; border-radius:20px;">

<h3 class="card-heading">Featured</h3>

</div>

<div class="card-body" style="height:75%">

<p class="sml-heading" style="text-align:center">Cheapest Fuel of the Month</p>

<p>UNLEADED: @Model.lowestRecord</p>

</div>

<div style="height:10%">

<button class="button-group cus-btn-drk">

View

</button>

</div>

</div>//Repeated for the fuel trends box, sign up box, overview box… </div>

<div class="col-lg-3 home-box">//The map box that holds the map of outlets and creates the map

<div class="card-body back-light shadow-lg h-100" style="border-radius:20px">

@if (Model.Outlets != null){

<div id="mapid"></div>

<p>@Model.Outlets.Count() Outlets are being shown.</p>

}

else {

<p class="alert-danger">No data found!</p>

}

</div>

</div>

</div>

@section scripts

{

<script>

var json = @Html.Raw(Newtonsoft.Json.JsonConvert.SerializeObject(Model.lstOutlets, Newtonsoft.Json.Formatting.Indented));

var mymap = L.map('mapid').setView([json[0].Latitude, json[0].Longitude], 7);

L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access\_token={accessToken}', {

attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors, <a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, Imagery © <a href="https://www.mapbox.com/">Mapbox</a>',

maxZoom: 18,

id: 'mapbox/streets-v11',

tileSize: 512,

zoomOffset: -1,

accessToken: 'pk.eyJ1IjoiY3JhbGViMDMiLCJhIjoiY2thdzRvNmdwMHNzNzJxcXJlaGI2cHhpMSJ9.YeNp5nbgNEugARD9KARtKA'

}).addTo(mymap);

for (var i = 0; i < json.length; i++) {

var marker = L.marker([json[i].Latitude, json[i].Longitude]).addTo(mymap);

}

</script>

}

### Browse Record List View (HttpGet):

public ActionResult Browse(string search, int? dropdown)

{

List<Record> lstRecord = new List<Record>();

if(search != null){

lstRecord = db.Records.Where(r => r.Outlet.OutletName.Contains(search)).ToList(); //Contains(search)

}

else

{

lstRecord = db.Records.ToList();

}

if(dropdown != null)

{

if(!lstRecord.Any(r => r.RecordID == dropdown))

{

Record oRecord = db.Records.FirstOrDefault(r => r.Outlet.OutletID == dropdown);

lstRecord.Add(oRecord);

}

}

return View(lstRecord.OrderBy(r => r.Price));

}

### Company Controller’s HttpGet for the Index Page:

// GET: Company/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Company company = db.Companys.Find(id);

if (company == null)

{

return HttpNotFound();

}

else

{

CompanyViewModel companyViewModel = new CompanyViewModel(db, id);

return View(companyViewModel);

}

}

### Home Controller’s HttpGet for the Index Page:

private ApplicationDbContext db = new ApplicationDbContext();

public ActionResult Index()

{

HomeViewModel homeViewModel = new HomeViewModel(db);

List<Outlet> lstOutlets = new List<Outlet>();

return View(homeViewModel);

}

# Evaluation and References

## Ongoing and Impacts

## Error Identification and Refinements

#### Issue

Initially, the code for adding data points to the map on the home page would not render the map:

var mymap = L.map('mapid').setView([json[0].Latitude, json[0].Longitude, 7);

The issue is that a square bracket was left off the end of the *Longitude* variable causing the ‘*7’* to be interpreted as a part of the original *json* variable.

#### Refinement

The code was updated to fix this error through adding the square bracket after the *Longitude* variable.

### Impacts

The goal of the webpage was to improve the user experience when searching for fuel prices and locations near the user. As this is the main goal for the webpage, the personal impacts are significant. Users who move to different, unknown locations can use this app to find fuel close to the user and at a cheap price. Furthermore, users may wish to track prices for a certain outlet or company. This was previously unavailable in the flat-file format the data was received in. Due to the requirement of entering an email the user then relies on the privacy of their details that are entered. The Australian Privacy Act (1988) is used to protect citizens from illegitimate data collection, ensuring the users privacy when entering personal details as well as allowing them to be revoked that privilege at any time.

#### Social

Providing a competitive edge when it comes to cheap fuel prices is in the best interest of society. Due to easy access to fuel prices, fuel outlets are required to reduce their prices if they wish to not be undercut by another fuel outlet. This added competition hinders on regular updates to keep the prices on the webpage current.

#### Economical

The main goal of the webpage is to allow the user to make an educated decision on the location and price of the fuel they wish to buy. This allows the user to take control of the money they spend on fuel. Furthermore, fuel price webpages can often require uses to pay to use the app, completely disregarding the purpose of the webpage. This application will be free for the user to access making their financial benefits be maximised.

#### Recommendations for impacts

It is clear that that impacts of this digital solution adheres to the person, social and economic impacts of Australian citizens. This solution was tailored to the needs of the users given by the client, but these functionalities can apply to most users. As privacy of data is essential, a role-based authentication system was implemented for uploading data as well as the passwords being stored in the database in an encrypted state (SHA-256). There is no risk of copyright laws.

### Ongoing

#### Prescribed criteria

The web application must:

|  |  |  |
| --- | --- | --- |
| Criteria | ✓/🗶 | Status |
| Include admin portal for staff to read write uploaded data | ✓ | Further validation requirements may still be implemented |
| Enable database to be updated with new data from the CSV file uploaded | ✓ | Further validation of duplicate files may be defined |
| Allow an automated email to be send to registered users. | 🗶 | The algorithm for dealing with automated emails has not yet been implemented |
| Allow users to view individual fuel outlets that display line graphs to give the user a deeper insight into fuel trends at the specific outlet. | ✓ | Ability to further filter the number and type of results can still be implemented. |
| Include an algorithm to read and write data to the database from an upload CSV file from the administrator. | ✓ | No further changes are required. |
| Allow users to view the website and data without registering an account. | ✓ | No further changes are required |
| Contain the functionality for users to register an account with the website. | ✓ | No further changes are required |
| Allow users to view generalised data for a holistic view of the data. | ✓ | Further filtration option when interacting with the graph |
| Follow the Web Accessibility Guidelines | ✓ | More icons could be implemented to replace text. |

#### Self-Prescribed criteria

The web application should:

|  |  |  |
| --- | --- | --- |
| Criteria | ✓/🗶 | Status |
| Contain visual indicators of rising and falling fuel prices | 🗶 | Logic to determine the trends with the data still need to be implemented |
| Incorporate accessibility through high contrasting colours between the foreground and background. | ✓ | Further contrast between the line graph and the background |
| Incorporate utility through web availability between multiple platforms. i.e. PC, Tablet, Mobile. | ✓ | Further support through showing less data points on smaller devices |
| Incorporate effectiveness through granting users access to specific search filtering when browsing data. | ✓ | Further filtration options for the user |
| Incorporate learnability through universal icons to represent functions of the website. i.e. A gear icon for settings. | 🗶 | Map icons are used by more universal icons can be user to replace words. |
| Incorporate accessibility and effectiveness through alt text on image and shortcut keys to move about the webpage. | ✓ | No further changes are required |
| Incorporate effectiveness through displaying the cheapest fuel prices | 🗶 | The algorithm used to find the cheapest fuel needs to be implemented |
| Incorporate effectiveness and learnability through page links being located on the top-left portion of the website for quick and intuitive access. | ✓ | No further changes are required |
| Incorporate learnability and repetition through consistent navigation bars being implemented through all the pages of the website. | ✓ | No further changes are required |
| Incorporate balance and space in the form of white space to bring the users attention to more important parts of the webpage. This is also used in conjunction with proximity. | ✓ | No further changes are required |
| Incorporate colour in conjunction with learnability by using different colours to represent meaning. i.e. A gold icon to represent a low price in fuel. | 🗶 | There needs to be a colour difference between the fuel types when viewing line graph of fuel prices. |

To move forward with the project, further investigation is need into publishing to a domain as well as registering with Google Words to make the site more accessible.

## Testing, Recommendations and Concluding Comments

### Test Plan and Recommendations

|  |  |  |  |
| --- | --- | --- | --- |
| Action | Expected Outcome | Actual Outcome | Recommendations |
| Login to website as an administrative user and edit the fuel data | The user will enter the required credentials (Email, Password), navigate to browse page, select an outlet, click edit make a change and press save. | * Credentials were added successfully * Browse page accessed * Outlet Accessed * Edit button found and accessed * Changes were made and saved | The user was unsure of what types of data could be in the variables.  Data validation could be implemented to correct the user if the wrong data type was implemented. This feedback needs to be instantaneous to be effective. |
| Upload fuel price data from .csv file provided by the Queensland data portal | The user will select the upload link, will have already successfully logged in. Select the choose file button, select the required file, and press submit. | * Upload link accessed * User chose the file button * Selected .csv file with fuel prices * File was submitted. | The user was unsure of what file to upload. Data validation could be implemented to ensure the same data isn’t uploaded twice. |
| Identify the most recent prices for the outlet: 7-eleven Acacia Ridge | The user will access the browse link, search for 7-eleven Acacia Ridge, select the outlet and hover over the most recent data point on the line graph. | * Browse link accessed * User searched for outlet * Outlet accessed * User found more recent price | The user found multiple outlets of the same name, this is due to each month adding multiple records. Filtration can be added to only show most recent price. The price was hard for the user to find on the line graph, it is recommended that a table be located underneath containing a more readable version of the data. |
| View a company’s fuel price trend | The user will select the company in the outlet view. | * Company link accessed | The trend of the data is hard to read as the values are so large, it is recommended that the range of numbers that are shown on the line graph are of a smaller sample size. i.e instead of 0 – largest price, use the smallest price as the lowest value. |
| Go to the second page of the browse page ordered by price descending | The user will locate the browse page and press the second page. The user presses the price icon to order by descending | * Second page located * Price was pressed twice | There was little feedback when the user changes the order of the results. It is also recommended that the page buttons be located at the top of the table for quick access by the user. |

### Accessibility Checklist

|  |  |  |
| --- | --- | --- |
| WCAG (W3C, 2018) | ✓/🗶 | Evidence |
| Provide text alternatives for non-text content | N/A | - |
| Provide captions and alternatives for multimedia | N/A | - |
| Make information adaptable and available to assistive technologies | N/A | - |
| Use sufficient contrast to make things easy to see and hear | ✓ | Figure 16 |
| Make all functionality keyboard accessible | ✓ | Figure 13 & 15 |
| Give users enough time to read and use content | ✓ | No time limit on how long the user can stay on a certain page |
| Do not cause seizures | ✓ | The use of a consistent background and an off white |
| Help users navigate and find content | ✓ | Encapsulation of each feature of website |
| Make functionality accessible through devices other than keyboards | ✓ | Can be accessed through touch devices |
| Make text readable and understandable | ✓ | High contrasting font colour with use of small easy to read words |
| Make content appear and operate in predictable ways | ✓ | The map is scrolled the same way all maps are on a computer and mobile device |
| Help users avoid and correct mistakes | ✓ | Requirements when creating data |
| Maximise compatibility with current and future technologies. | ✓ | Figure 13 & 15 |

### Concluding Comments

The project was very successful. Having a user test the solution resulted in further recommendation to improve the site. The site still has area to improve including adding icons and user feedback when certain actions are performed. Further enhancements to the visual representation of data can also be added, including further filtration options, types of graphs and data that is show (diesel, petrol, e10). Additionally, the email system that was

The project was quite successful with users testing the sites giving positive feedback and further recommendations to improve it. The still has areas to improve including adding icons and user feedback when certain actions are performed. Furthermore, the must have feature, *sending an automated email to registered users* has not been implemented at this stage but will certainly be added in the future. Additional enhancements to the visual representation of data will also be added. This includes filtrations options, types of graphs and types of data that is shown (diesel, petrol, e10). This will allow users to gain a more educated insight into fuel prices around them. The MVC architecture allows for a more streamline navigation through data as well as giving a better insight into the data trends than the flat-file .csv format the data is received in. For future projects, it is recommended that other database management systems be investigated and evaluated to achieve a better website. A big limitation that comes with this data management format is that this webpage is unable to run without a licence. Master Data Management (MDM) is another data management system that the client may be able to use.

## References

Atzeni, G. (2020, April 21). Introduction to ASP.NET MVC. Retrieved from Highfields State Secondary College e-courses, DIS12, <https://stileapp.com/au/HSSC-QLD-5611/2020-231097/lesson-1547262/worksheet-11629435>

Atzeni, G. (2020, April 21). Introduction to ASP.NET MVC [Video file]. Retrieved from <https://stileapp.com/au/HSSC-QLD-5611/2020-231097/lesson-1547262/worksheet-11629435>

Australian Government (1988). The Privacy Act. Retrieved from <https://www.oaic.gov.au/privacy/the-privacy-act/#:~:text=The%20Privacy%20Act%201988%20(Privacy,other%20organisations%2C%20handle%20personal%20information>.

Chart.js. (n.d.). Chart.js documentation [Source code]. Retrieved from <https://www.chartjs.org/docs/latest/>

Microsoft. (2019). Introduction to ASP.NET identity. Retrieved from the Microsoft .NET documentation website: <https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity>

Profisee. (2018). MASTER DATA MANAGEMENT – WHAT, WHY, HOW & WHO. Retrieved from <https://profisee.com/master-data-management-what-why-how-who/>

W3C. (2018). Web Content Accessibility Guidelines (WCAG) 2.1. Retrieved from <https://www.w3.org/TR/WCAG21/>