ASHLEY JAMES DOWIE

sit 120 aSSEMENT 2: pRACITAL pORTFOLIO

Table of Contents

[Marking Justification 1](#_Toc81753141)

[Week 1 1](#_Toc81753142)

[Reflections 1](#_Toc81753143)

[Practical Tasks 2](#_Toc81753144)

[Task 1-3 2](#_Toc81753145)

[Task 4 3](#_Toc81753146)

[Week 2 4](#_Toc81753147)

[Reflection 4](#_Toc81753148)

[Practical Tasks 4](#_Toc81753149)

[Task 1 5](#_Toc81753150)

[Task 2 6](#_Toc81753151)

[Task 3 8](#_Toc81753152)

[Task 4 8](#_Toc81753153)

[Week 3 9](#_Toc81753154)

[Reflection 9](#_Toc81753155)

[Practical Tasks 11](#_Toc81753156)

[Tasks 1 11](#_Toc81753157)

[Tasks 2 13](#_Toc81753158)

[Tasks 3 13](#_Toc81753159)

[Task 4 14](#_Toc81753160)

[Week 4 15](#_Toc81753161)

[Reflections: 15](#_Toc81753162)

[Practical Tasks 16](#_Toc81753163)

[Task 1 16](#_Toc81753164)

[Task 2 16](#_Toc81753165)

[Task 3 17](#_Toc81753166)

[Week 5 17](#_Toc81753167)

[Reflections 17](#_Toc81753168)

[Practical Tasks 18](#_Toc81753169)

[Task 1 18](#_Toc81753170)

[Task 2 19](#_Toc81753171)

[Task 3 19](#_Toc81753172)

[19](#_Toc81753173)

[Task 4 19](#_Toc81753174)

[Week 6 20](#_Toc81753175)

[Reflections 20](#_Toc81753176)

[Practical Tasks 21](#_Toc81753177)

[Task 1 21](#_Toc81753178)

[Task 2 22](#_Toc81753179)

[Task 3 23](#_Toc81753180)

[Task 4 23](#_Toc81753181)

[Week 7 24](#_Toc81753182)

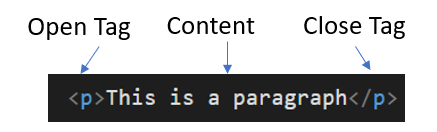
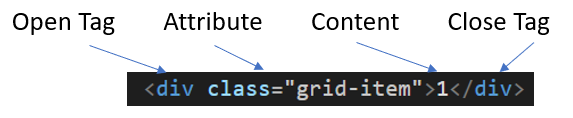
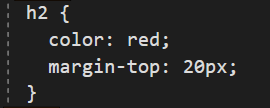
[Reflections 24](#_Toc81753183)

# Marking Justification

# Week 1

## Reflections

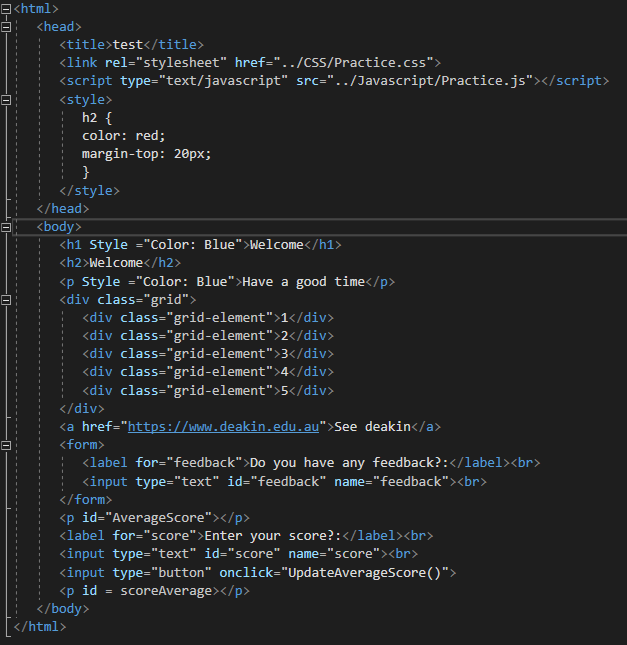
This week was the first week of the unit. As this week was the first week of the unit, we covered the unit structure, assessments and an outline of the content that will be covered. In addition to this we also were introduced to the fundamental building blocks of a web applications.

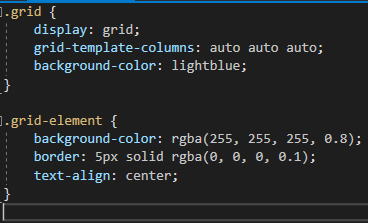
* HTML: Hyper Text Markup Language. HTML gives us the ability to outline the structure and content of our website. HTML is the main building block of any website as all websites require HTML to exist. A HTML document is made up entirely of elements
  + Elements: A HTML element makes up the content of a HTML page. Every element has a tag
    - Tag: A tag defines what type of element a HTML element is. Html tags can be structured in different ways depending on the tag. The below tags are both valid. Tags all begin with an opening tag such as <TAG>, some are then followed by content and then closed by a closing tag </TAG>
      * A paired tag  
         
      * An unpaired tag  
        
      * Tags can also contain further instructions within the opening tag known as attributes.  
        
* CSS: Cascading Style Sheets. CSS is used to modify appearance of websites and web applications. The structure of CSS is based on selectors and declarations.
  + Declarations are key values pairs which have a property and a value formatted as property: value. The property describes the describes the aspect to change such as colour, size, and many other things. The value assigns a value to this property.
  + Selectors contain declarations within them. The selector provides an instruction as to which elements the declarations should apply to A selector with inline CSS is not required as inline CSS is written as CSS declaration appearing within a HTML tag as an attribute as only applies to that single HTML element.
    - This is an example of inline CSS  
      
    - This is an example of a CSS selector The below selector would apply to any element that has the <h2> tag.  
      
* JavaScript: JavaScript is used to make web applications respond to user input. As JavaScript is a programming language, it is not as easy to succinctly define its syntax. JavaScript like most programming languages consists of objects, variables and functions. JavaScript is capable of adding, removing and modifying html elements in response to user input.

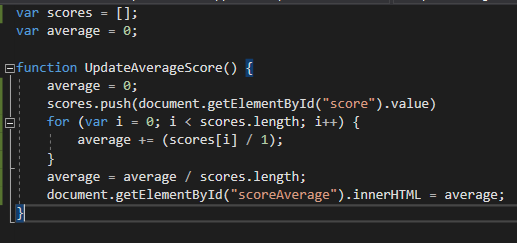
## Practical Tasks

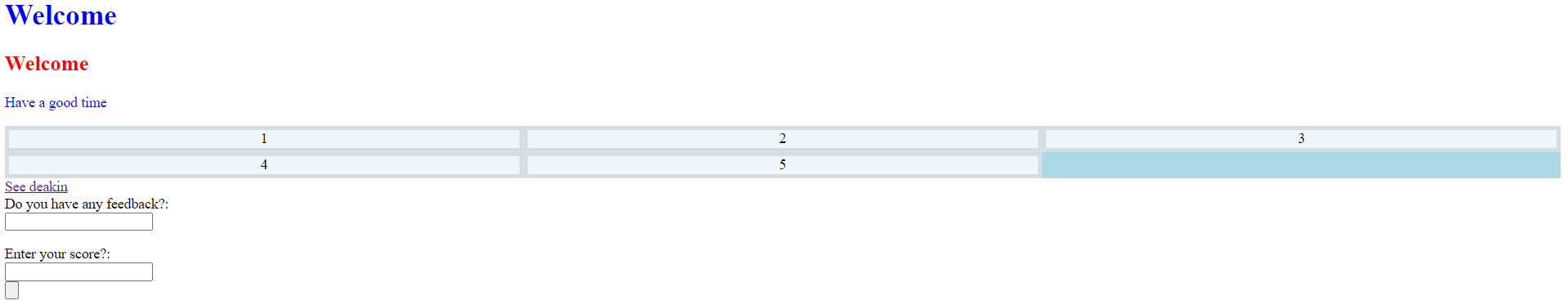
### Task 1-3

When undertaking this task, I was getting strange values when trying to calculate the student score average. I noticed I could fix this by changing scores[i] to scores [i]/1. I did not understand why this worked, only that it did.





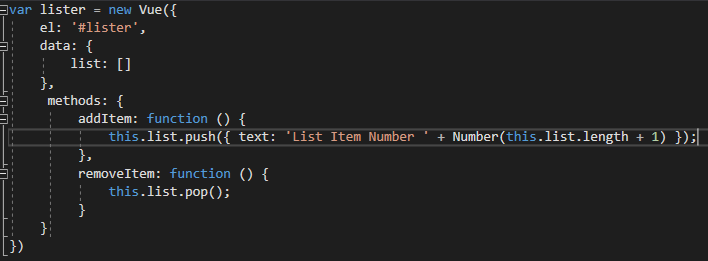




### Task 4

I was getting strange values when pushing new items to the list. Later I realised that the reason for this was because the values were acting as text addition rather than numerical addition. I resolved this by wrapping the values being added in Number() to convert them to a number after reading this documentation [Number - JavaScript | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number) This helped me to realise what was happening with my previous issue that I resolved by changing score[i] to score[i]/1.

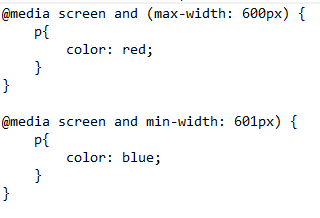




# Week 2

## Reflection

This week we learnt about what responsive websites are and how to create them. Responsive websites are websites which can respond to the size and capabilities of the device being used. A responsive website will respond/ change, so they display optimally on different screen sizes and shapes. Below is a list of ways to make a website more responsive:

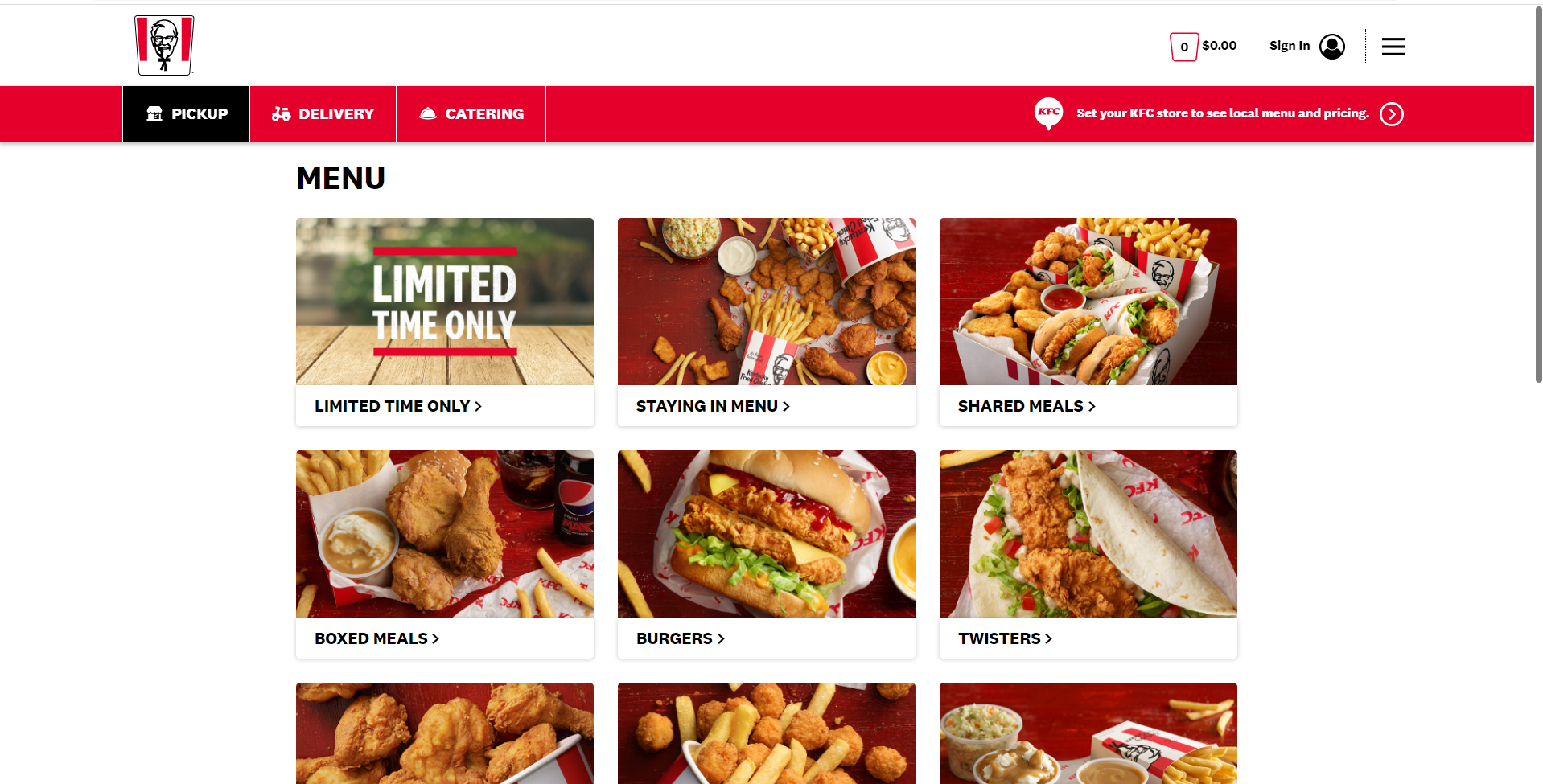
* Use the viewport tag in the html document. Using the viewport tag till allow the content of the website to be displayed in a way that fits the specific devices screen. The viewport tag is written as <meta name="viewport" content="width=device-width, initial-scale=1"> and should be included in the HTML head.
* Do not set content sizes as absolute values such as pixels, instead use percentage so that the data will scale with changes in screen size. This is handled within CSS
* Use breakpoints to determine how the display of the page should change at certain size ranges. Media queries are written in CSS, they are similar to if else statements. A particular screen size in pixels (normally width) is set as the point that if the screen size is above this amount, certain CSS declarations will be user, whereas if the screen size is below this point, other CSS declarations will be used. The following is an example of a media tag that will display paragraphs as blue if the screen is wider than 600 pixels, otherwise paragraphs will be red.   
  

In addition to what responsive website design is and how to achieve it, we also learnt about user stories UX and UI.

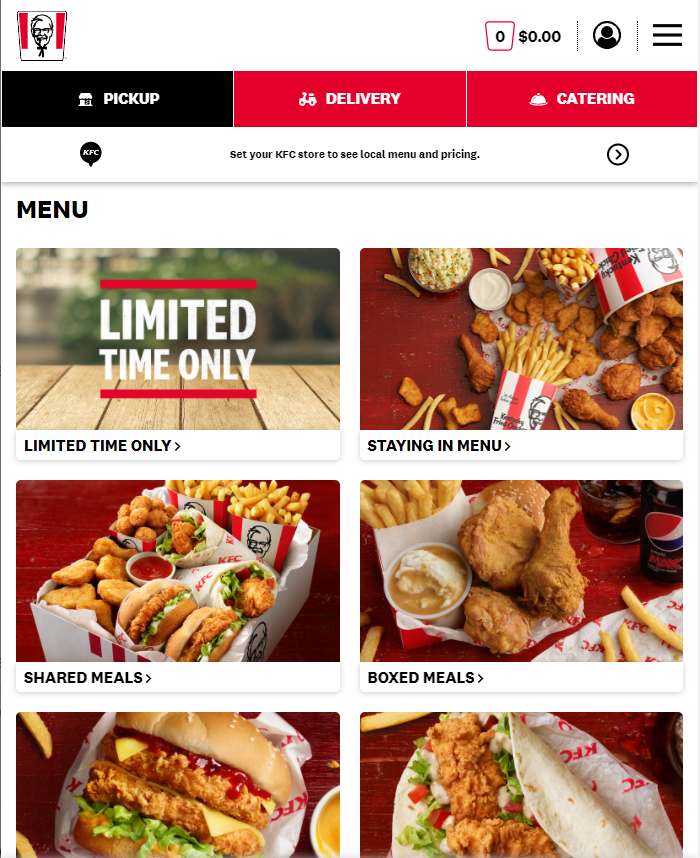
* User stories: User stories (also known as personas) are a description of an imaginary person who uses will use the website. User stories describe how the user likes to interact with websites and who their expectations, needs and desires of the website are. Stories of different types of users can be combined to get an understanding of user requirements and how to make the websites easier to use for these types of users.
* UX: User experience design is about the feelings of humans when interacting within an organisation/ system. In theory user experience design is not only applicable to applications.
* UI: User interface design is about the look and feel of a product that a user interacts with. User interface design is applicable to software applications.

## Practical Tasks

### Task 1

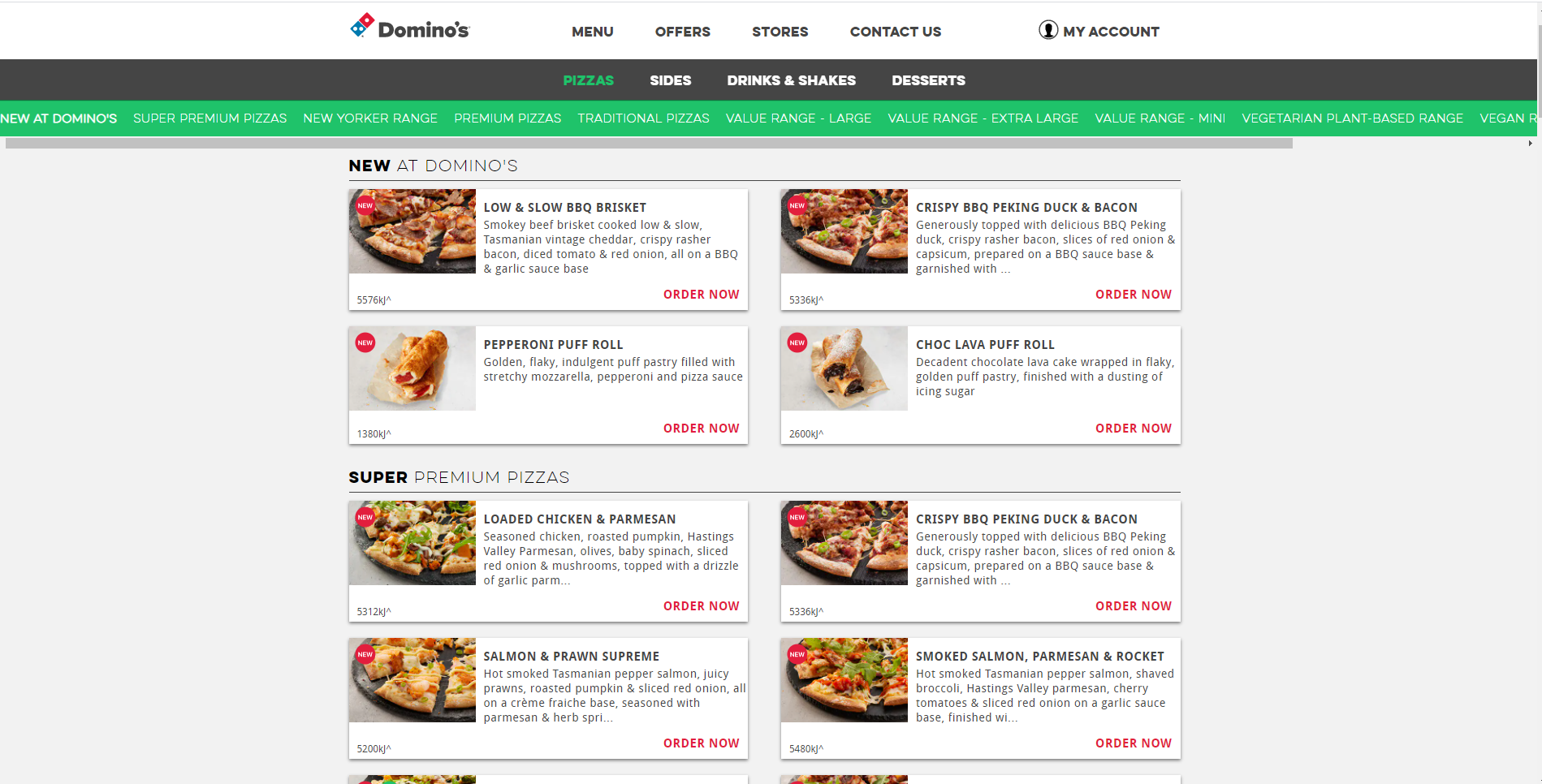
KFC’s website is responsive, particularly its menu. Different menu items are represented by an image. Images respond to image size by reducing image size and the number of images displayed per column as screen size is reduced. Below is the KFC screen at full width:  


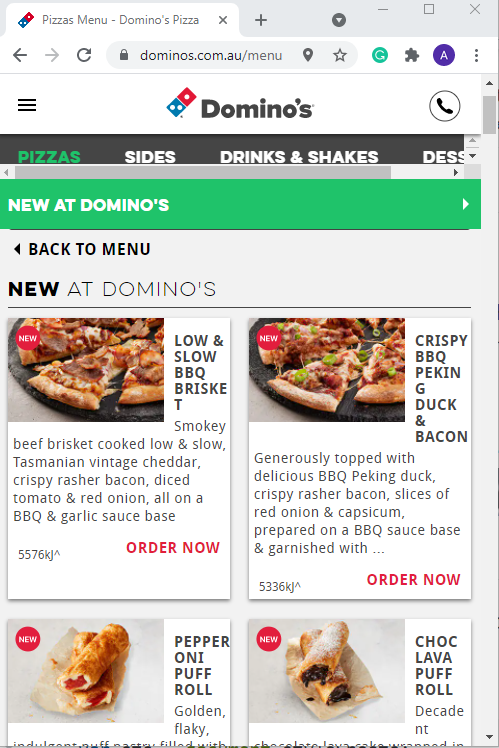
And now here it is at about half width.



As you adjust the screen size you can see where the media controller break points are occurring in the CSS code. KFC achieved this by most likely using flex grids with different break points.

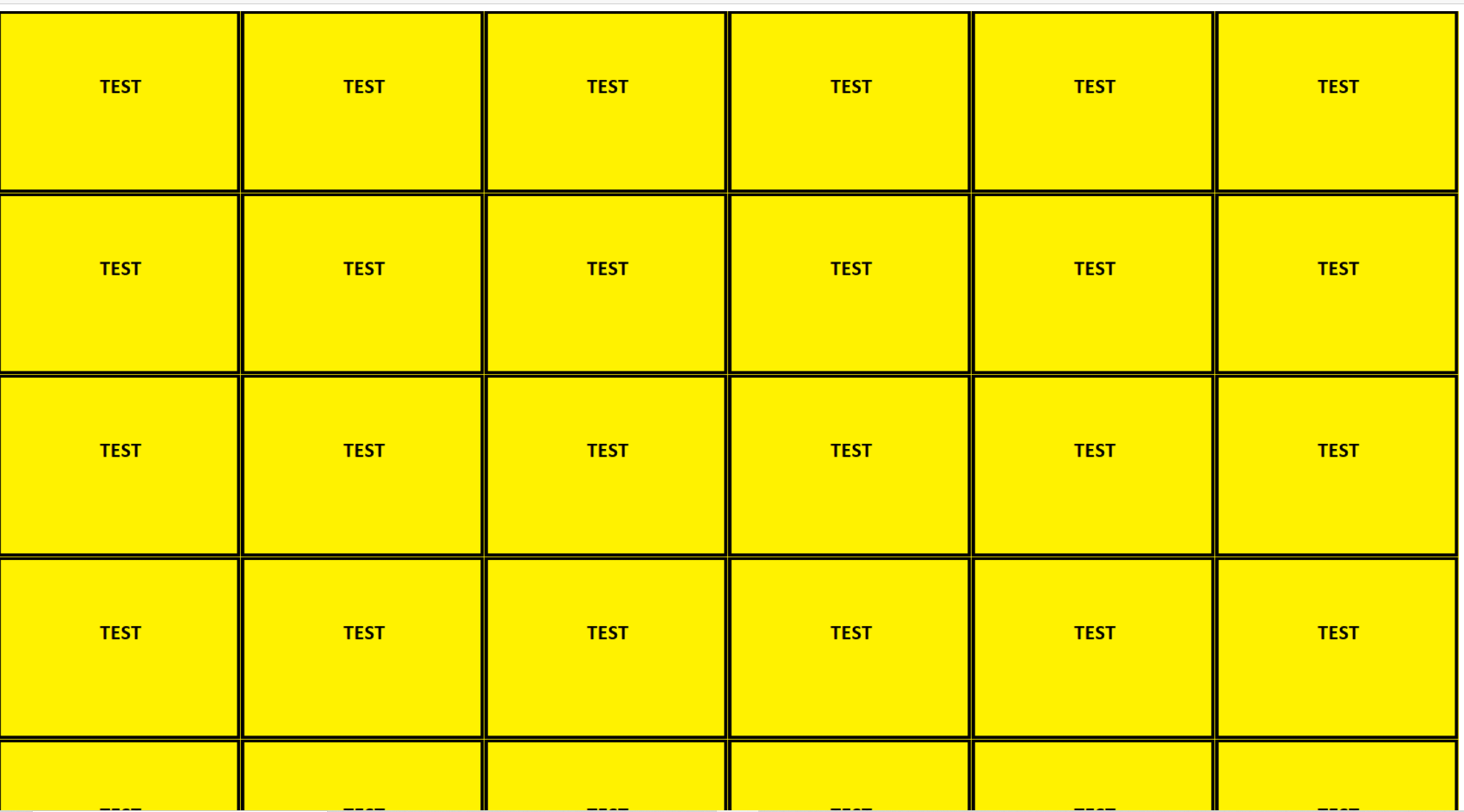
As for the dominos website, the number of columns is not adjusted as the image resolution shrinks. instead, items are stretched vertically. The dominos website can achieve this without stretching the images, as the images take up a small portion of each items content. This means the images themselves to not need to be resized. Most of each item is text, which can be easily spread down vertically.



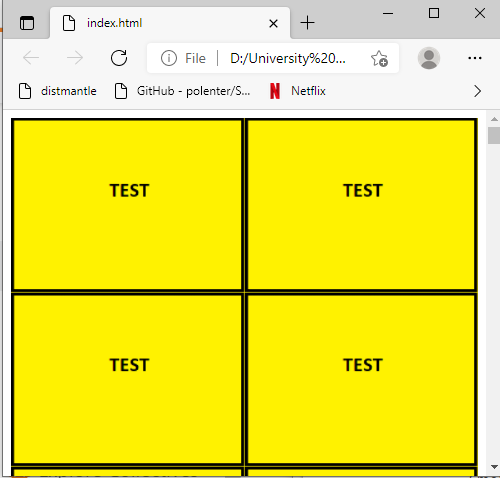


### Task 2

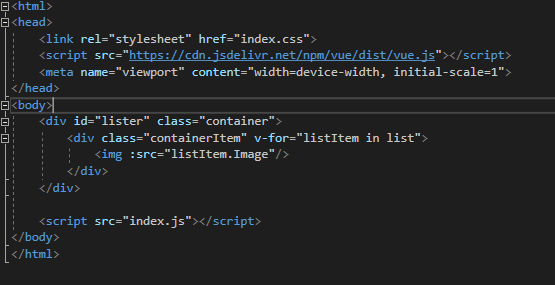
For task 2. I wanted to try and replicate the approach the KFC had taken with their website. I did this by using flex grids and breakpoints with the media controller.

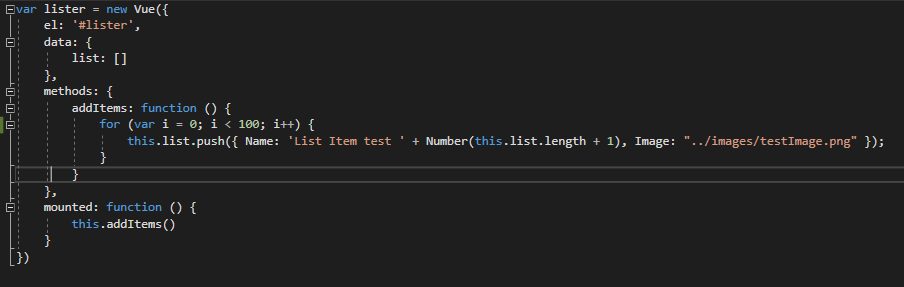
My page looks like this at full screen size:

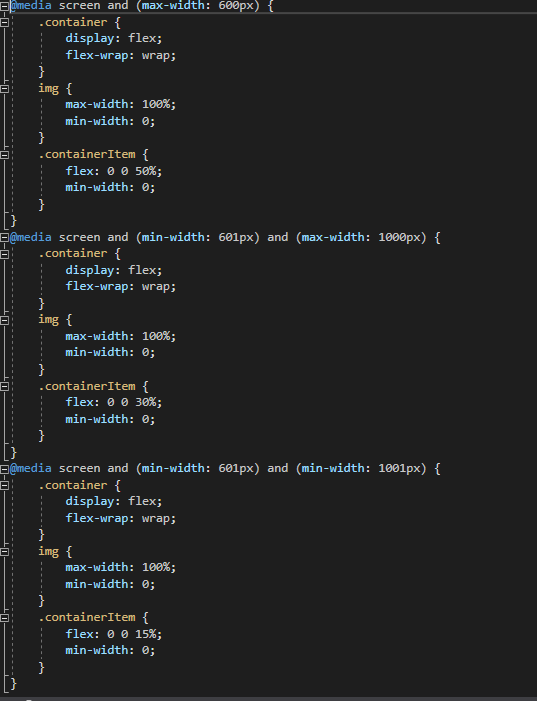
And this at minimum screen size:



I achieved this will the following code:



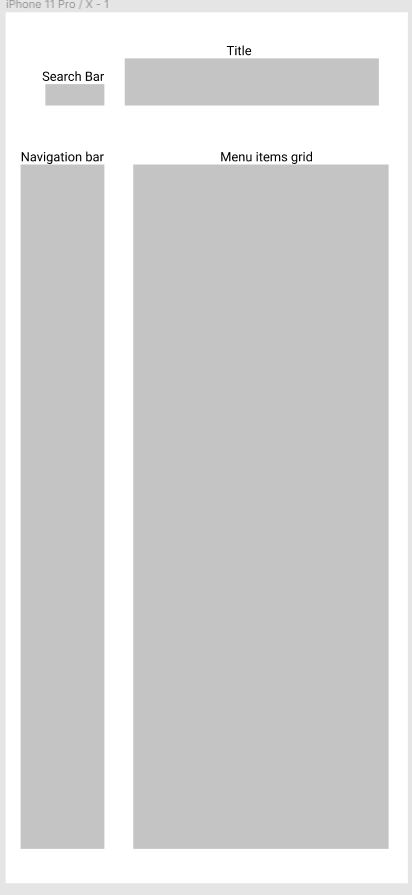




### Task 3

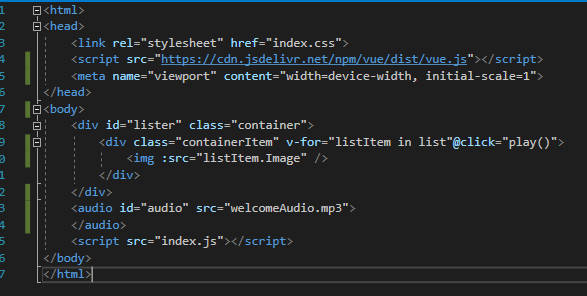
As John is a manual labourer, he gets quite hungry after he finishes work. John often does overtime. As a result of this when he finishes work it is quite late, and he wants food on the go. Unfortunately, the lines at many of the takeaways are quite long around the time that John finishes work. Because of this, it would be better for John to be able to order ahead so his food is ready to pick up as soon as he gets to the takeaway. Also, as John is coming from work, he won’t have access to a computer, only his mobile phone. John doesn’t have the best eyesight and he has little patience for hard to use applications, so the any application he uses needs to be easy to see despite being on the smaller screen of a mobile device.

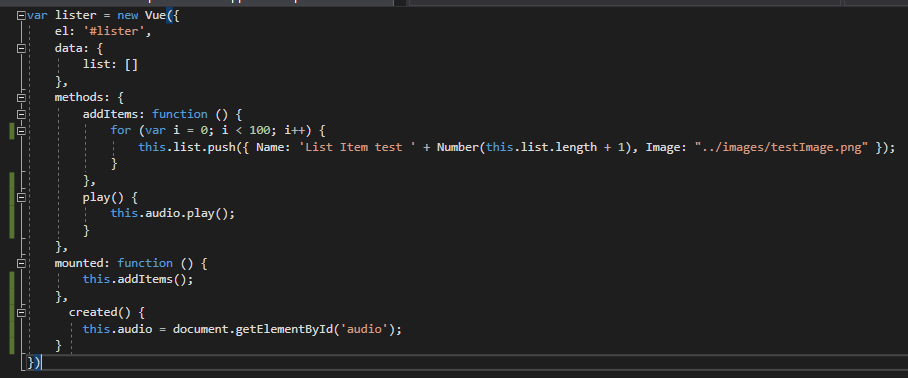
I used Figma to design the below prototype for a food ordering app. This app displays things horizontally for the most part so that users are not required to scrolls horizontally through the navigation bar or menu. I tried to make the app’s layout similar to what you would see in any other typical mobile app as a user like John would not want to spend time having to adjust to a application that breaks standard conventions.



### Task 4

I added some audio functionality to the website. When the user selects any of the menu items, an audio clips now will play. I achieved this by using the following code.





# Week 3

## Reflection

This week we learnt about JavaScript.

We learnt about syntax, variables, control statements, how to use JavaScript to change html elements by using the DOM. Based on JavaScript there are many libraries and frameworks, such as view, jQuery and ajax. Only Vue will be covered in this unit, however knowledge of jQuery and ajax is important. JavaScript is an object orientated based programming language, it is prototype based. designed to make websites content dynamic. JavaScript contains many of the common data structures that most programming languages have, such as arrays. JavaScript also has the typical control statements such as loops, if else logic and switches. JavaScript also has its own common default libraries.

The power of JavaScript is the link between JavaScript and dom.

Ways to declare a variable in JavaScript: var, let, const.

Variables declared with var exist within the scope of the function that declares it. Variables declared with let exist within the scope of the opening and closing brackets {} which contain it. The other difference is that let enables superior debugging to var, as if you try to make use of a variable declared with let after the statement trying to use it, you will receive a reference error alerting you to the problem at its source. If you did the same with a variable declared with var, the program would allow this, however the variable would be set as the type undefined, which may not throw an error at the exact point of the program where it is first used, making debugging harder.

One thing that is slightly different about JavaScript to other languages is the use of === when evaluating equality between two variables. The use of === in JavaScript means that the type of the two variables must be the same as well as the value, whilst the use of == means that the value of two variables is considered only. An example of this is as follows.

var value = true;  
If(value == 1)

would evaluate as true as both variables have a value of 1.

var value == true;

if(value === 1)

would evaluate as false, as although both of these variables have a value of 1, their types are not equal, one being a bool and one being an int.

When using a switch statements, you can make two case values use the same code chunk by simply writing the case statements as

Switch(test) {

case ‘value1’:  
alert(“value1”);  
break;  
case ‘value2’:  
case ‘value3’:  
alert(“value2 and value3”);  
break;  
default:  
alert(“default”);

}

Some of the things I have learnt are the difference between let and var and what the use of === means. I also was not aware of the ability to make two case statements run the same code.

DOM: Document Object Model is a tree structure that looks like this.

(insert image)

Every element is represented as an object which can be accessed by JavaScript using inner html. Just like any other JavaScript object, DOM elements have member variables and methods/functions.

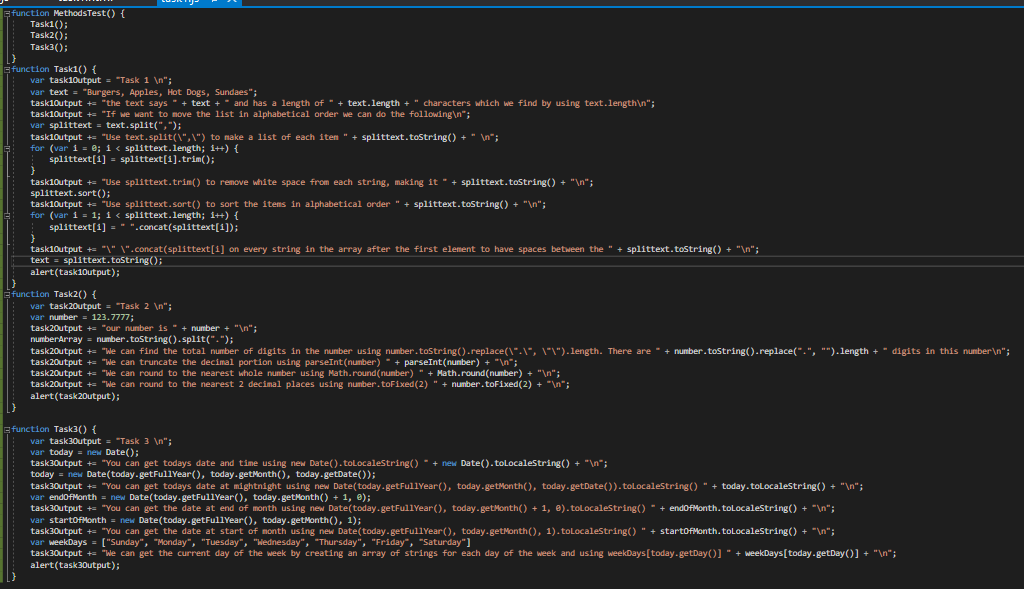
Ajax and jQuery allow us to exchange information between the server and the client. JSON is used as a way to send and receive information from client to server. JSON objects are key value pairs separated by a comma. JSON objects can be converted a string value using JSON.strinfigy(mystring). Strings can also be converted to JSON objects using JSON.parse. JSON object data can be accessed like we would access any property from an object using object.propery.

JQUERY is a JavaScript library which allows html/DOM manipulations, CSS manipulation, html event methods, effects and animations, ajax, utilities.

AJAX: Asynchronous JavaScript and xml. AJAX allows communication between the client and the server. Ajax allows data to be retrieved from the server and displayed on the page without reloading the page. JavaScript allows the client to sent information to the server using post. JavaScript allows the client to request information using post.

## Practical Tasks

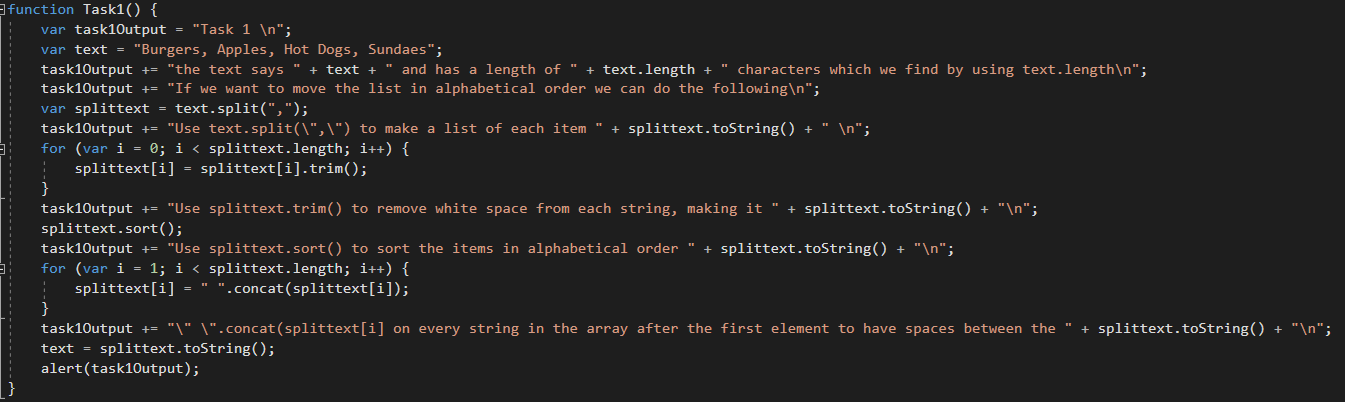
Tasks 1 to 3 were completed in a single JavaScript file which displays an alert when the html file loads. The two files are as below:

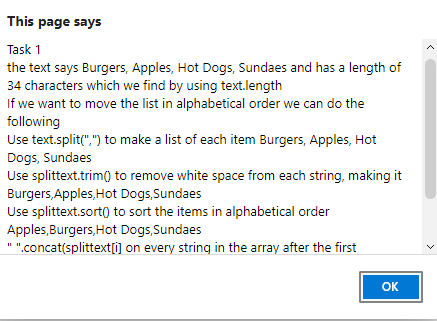


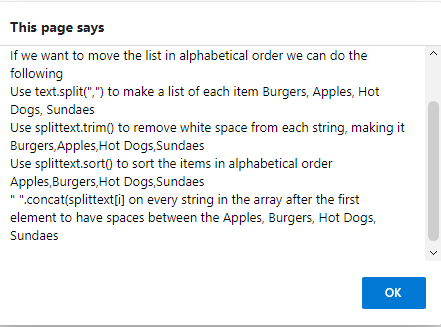


### Tasks 1

For this task I demonstrated how to take a list of items in a string and rearrange them I alphabetical order. I made use of split, trim, sort, toString and contact to achieve this.

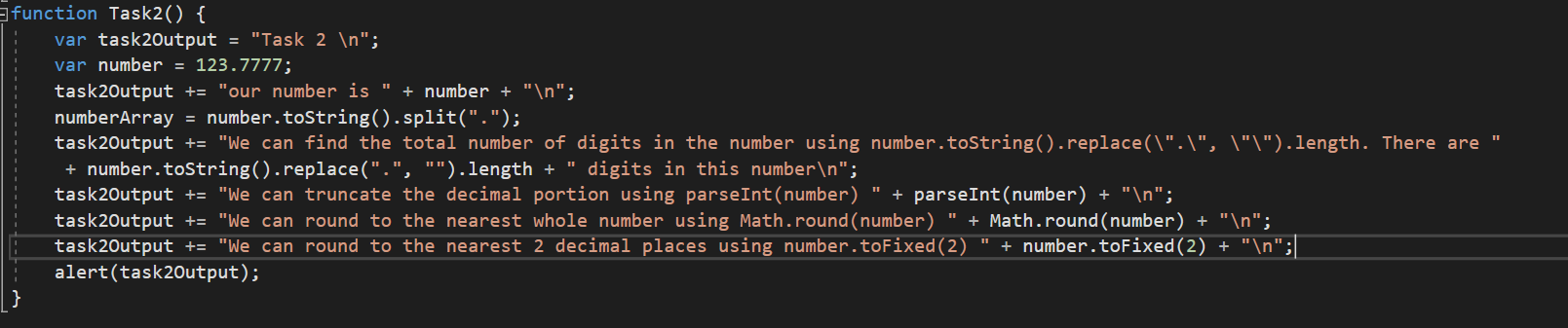


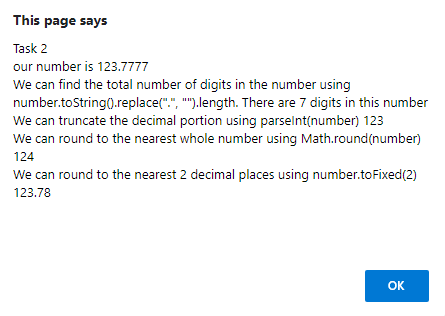




In this task I demonstrated the use of various number functions. I used parseInt to demonstrate truncation, I user Math.Round to show how to avoid truncation and I used toFixed to show how to round a number to the nearest two decimal places. I also used string manipulation to count the total number of digits in the number.

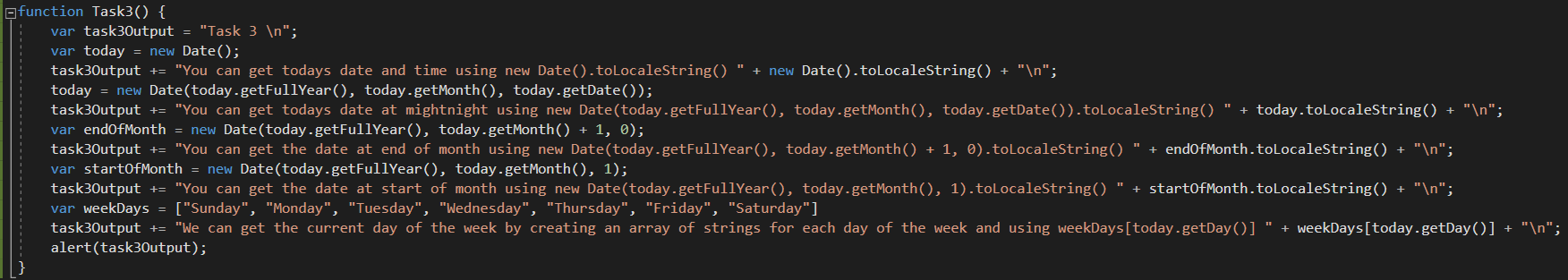
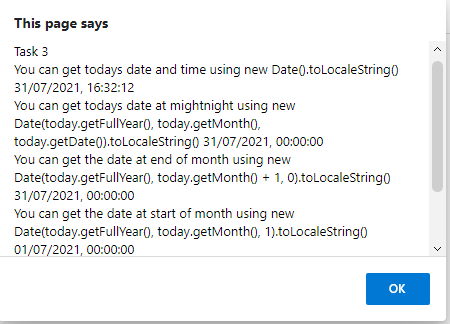
### Tasks 2

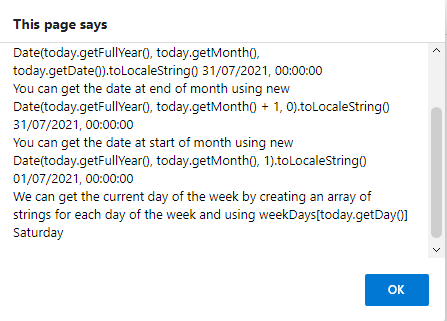




In this take I demonstrated how to find the current date, the date for the end of the month and the date for the start of the month. I did this using the built in JavaScript date functions, such as getDate, getMonth, getFullYear, getDate. I also demonstrated how we can return the current date of the week by creating an array of string for each day of the week and using the result of getDay as the index.

### Tasks 3



### Task 4

* Computer property: A computed property is a value which is calculated based on other values.
* Object and style binding: Object binding allows Vue to manipulate html based on the properties of a Vue component. It is “one way” meaning that html will not change the JavaScript (component) properties, but these properties will change the html. This allows Vue to easily display the value of a JavaScript object.
* Conditional rendering: Conditional rendering allows html elements to be displayed based on an if statement with the condition being the value of a JavaScript (component) property.
* List rendering: List rendering allows us to use foreach loops within our html. This means that elements for each item in a JavaScript array can be displayed.
* Event handling: Even handling allows JavaScript code to be ran based on a html event such as onclick. This is somewhat like the reverse of Object binding. Whilst object binding allows JavaScript to effect html, event handling allows html to effect JavaScript. Like object binding it is also “one way”.
* Form input binding: Form input binding allows input into html to effect JavaScript variables. It also allows JavaScript variables to effect html. It is a “two-way binding” allowing variables to be kept in sync on both the JavaScript side and the html side.
* Component: A component in Vue is a Vue instance which can be used inside of the root Vue instance.
* Component Registration: Component registration allows components to be accessible in other view components templates. Globally registering a component makes that component available to any view components template. Locally registering a component allows us to choose whether that component should be accessible in another components template from that other component.
* Props allow us to pass values to a Vue component. This allows the same Vue component to represent different data and be used in various situations.
* Custom events: Custom events in Vue allow us the ability to emit our own custom events, changing their behaviour.
* Slots allow the content of a component to render differently based on how the component is used.
* Dynamic and async components. Dynamic components cache their state after events, such as switching tabs. Asynchronous components are loaded only as they are required.
* Edge cases: Edge cases are situations where it is appropriate to break the normal Vue rules and conventions to solve them.

## Project Progress

This week I began working on my proof of concept and my propose for assignment 1. I have decided to create a web application called Password Place. Password Place will allow users to generate a unique password for all the different applications that the user uses.

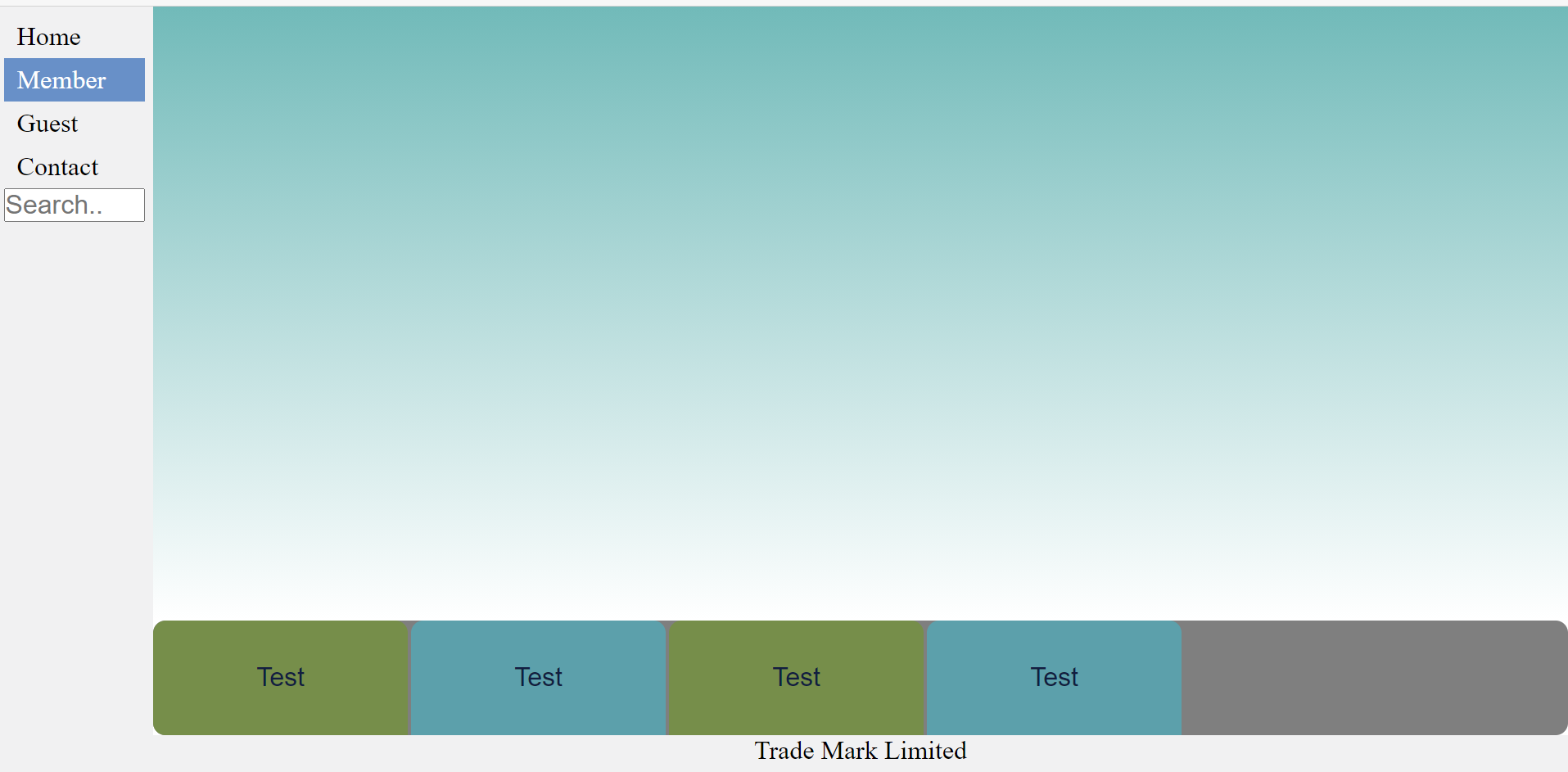
This week I completed the following tasks

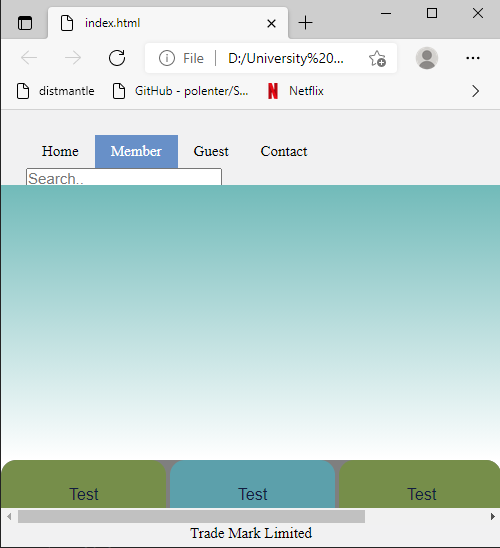
Proof of concept:

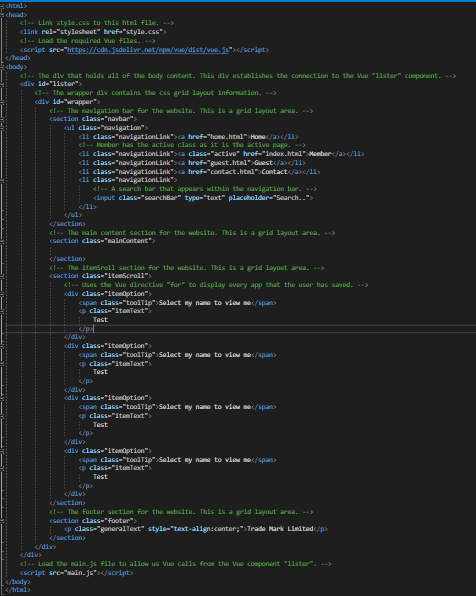
* Created a basic layout for the web application using sections.
* Created a navigation bar.
* Created my desired CSS styling for the page.
* Made the options menu at the bottom independently horizontally scrollable.
* Ensured page is responsive to resolution changes by using media queries.

Proposal:

* Prod Summary.
* Asset List.
* Product Purpose.
* Target Audience.
* Justification for Creativity.







Work for next week:

Proof of concept:

* Make the options selectable. 1h
* Make the main password generation form. 25m
* Make the main password generation form interactable. 45m
* Create the password generation logic. 1h 30m
* Create placeholders for the guest, home, and contact page. 40m

Proposal:

* UX/UI design. 1h
* Major components and their intended behaviour. 1h
* System summary (including flowchart) 2h

# Week 4

## Reflections:

This week we learnt about networking infrastructure and protocols for the web.

Every device connected to the internet has an IP address (internet protocol address). Every IP address is unique for each device. It is essential that every device connected to the internet has a way of identifying it to allow communication between devices to occur.

IP addresses are written to uniquely identify devices; however, they are not user friendly nor easy to remember. To allow people to enter the address of websites in a more human readable way, DNS (domain name system) are required. DNS provides a mapping between a website name and an IP address. For example, if DNS did not exist and you wanted to reach Facebook, you would need to type 69.63.176.13 instead of www.facebook.com.

Every website is hosted by a physical device that is connected to the internet. Every device has multiple ports. Each port is associated with a specific process or application. Websites are generally associated with port 80, there are also many other ports, such as port 20 and 21 which are associated with FTP.

When access a website, a request is sent from your pc to a LAN (local area network) then to many physical devices over a WAN (wide area network). The packet will travel one device to another until it reaches the destination device. When trying to communicate with a computer that is not part of your LAN, your computer uses its default gateway to send the message.

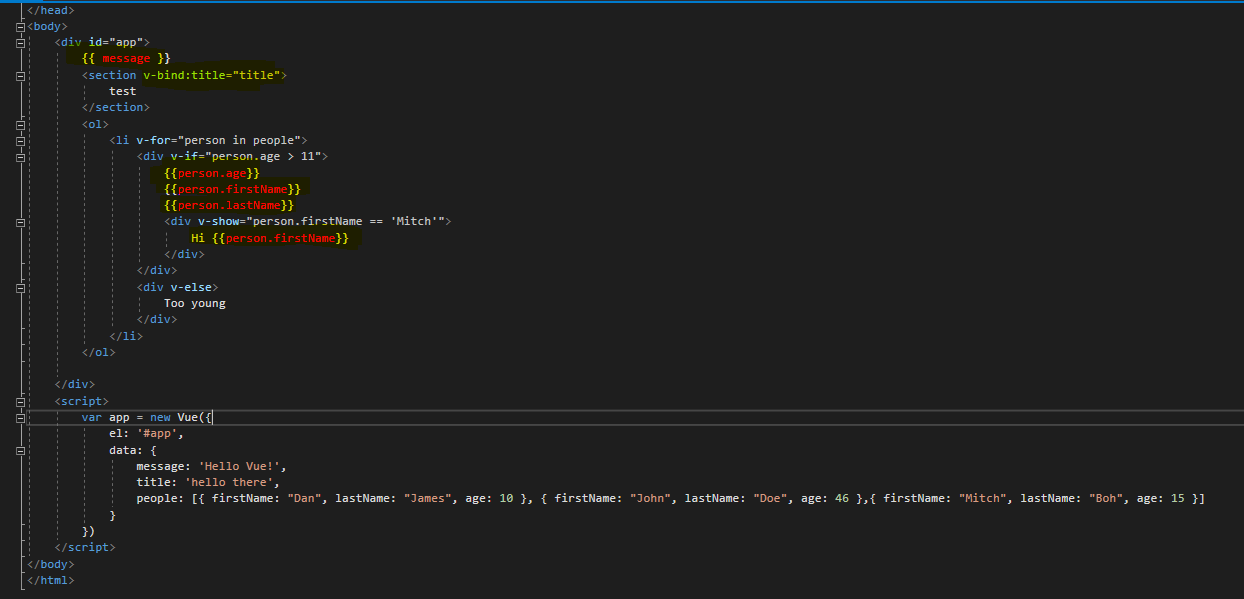
UDP (user datagram protocol) sends communication from the requestor to a server. UDP does not send an acknowledgement from the server to the requestor for packets. TCP (transmission control protocol) is like UDP but does involve the server sending back and acknowledgement to the requestor to confirm that the packet was received.

UDP is an unreliable protocol, meaning that you cannot confirm if a package was received by the server. UDP is faster than TCP as there is no need to wait for an acknowledgement from the server. UDP is used when the loss of a single package does not matter, such as a YouTube video where the loss of the packet would mean losing something unimportant like a single frame of video.

TCP is a reliable protocol, meaning that you can confirm if a package was received by the server. TCP is slower than UDC as there is a need to wait for an acknowledgement from the server. TCIP is used when the loss of a single package is a problem. For example, when sending an email if a packet is not received by the server, the email may not be received by the recipient at all.

## Practical Tasks

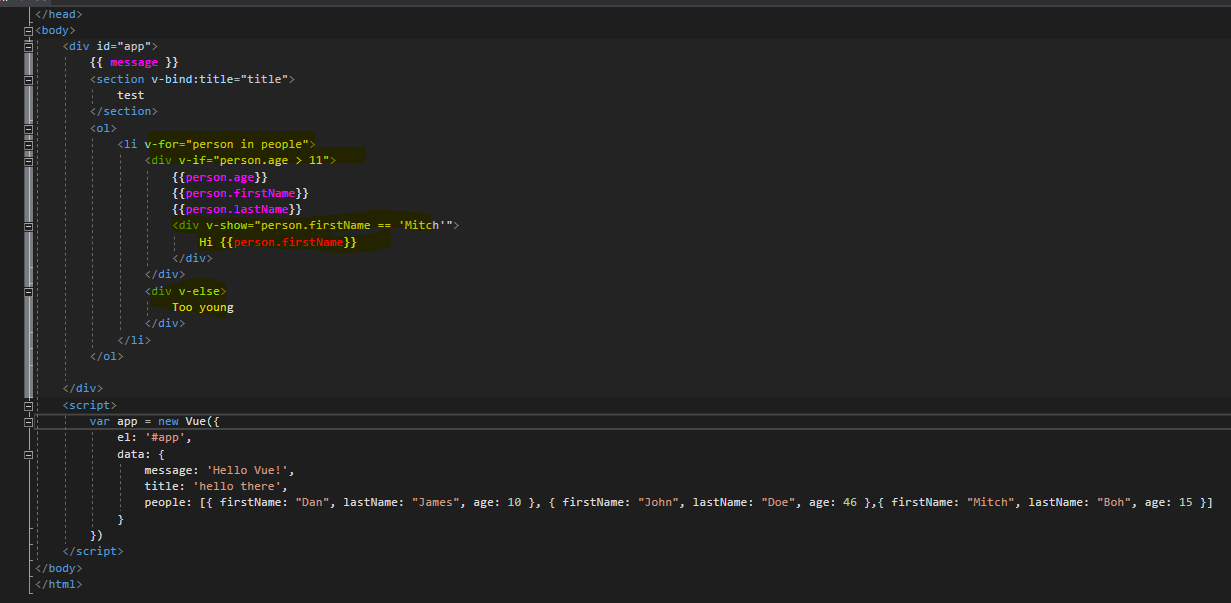
### Task 1



Declarative rendering is a Vue feature which allows us to easily display variables form JavaScript in HTML. Here I have used text interpolation, which is one form of declarative rendering. I have also used v-bind to associate a html attribute with a JavaScript variable.

### Task 2

Vue’s loops and conditionals allow us to use loops and conditional logic inside of a html document in a similar syntax as we would use in a programming language. My code below will go through a list of people and display their name and age. If anyone on the list is below 11 years old, their details will not be displayed. Anyone with the name Mitch will display an additional hello message. When I undertook this task, I learnt the difference between v-show and vi-if. V-if will remove the elements completely if the condition to render them is not met. Whilst v-show will simply use css to stop them from being visible, without removing them form the DOM. I learnt that it is best to use v-if if the content is unlikely to toggle between displayed/ not displayed often. Otherwise it is best to use v-show.



### Task 3

I discussed with some of my peers on teams my plan to make a password generator. Some of them said they thought this would be difficult. However, I have a plan to do this by simply taking a password and an apps name and then converting the characters to ascii numbers, doing some modification, and then converting back to ascii characters. I think I have a solid idea of how this will work but talking with peers to bounce my ideas off has been helpful.

## Project Progress

This week I need to finish my first draft for my assignment 1. Last week I managed to create the main layout and styling for Password Place and finish roughly 50% of the propose. This week I need to focus more on using JavaScript to create a demonstration of Password Place’s functionality.

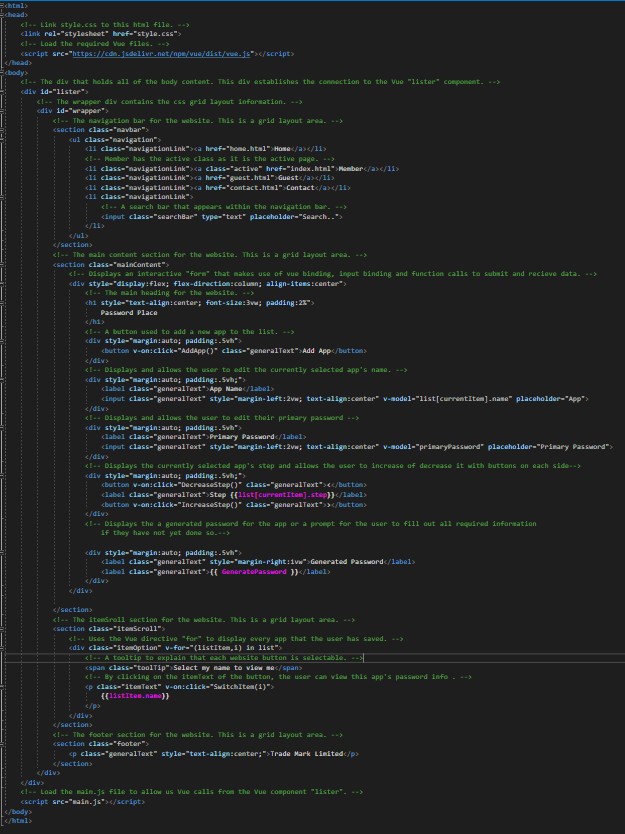
This week I completed the following tasks

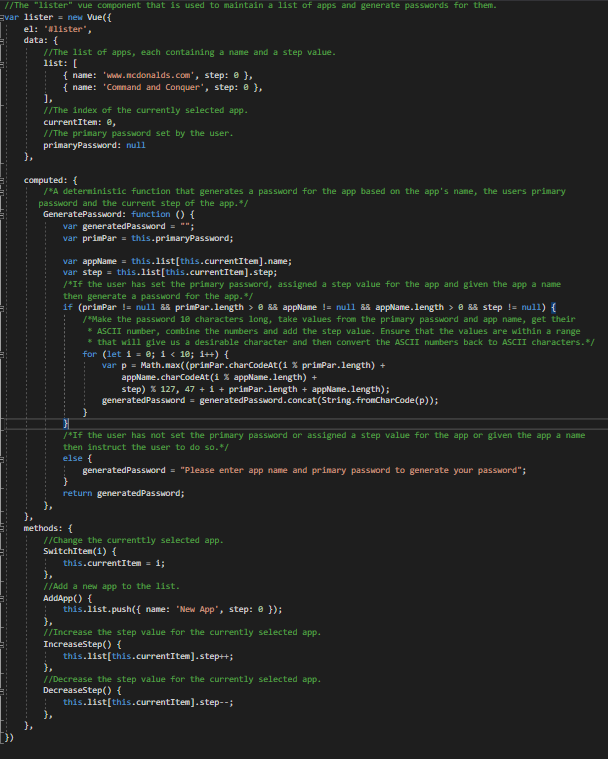
Proof of concept:

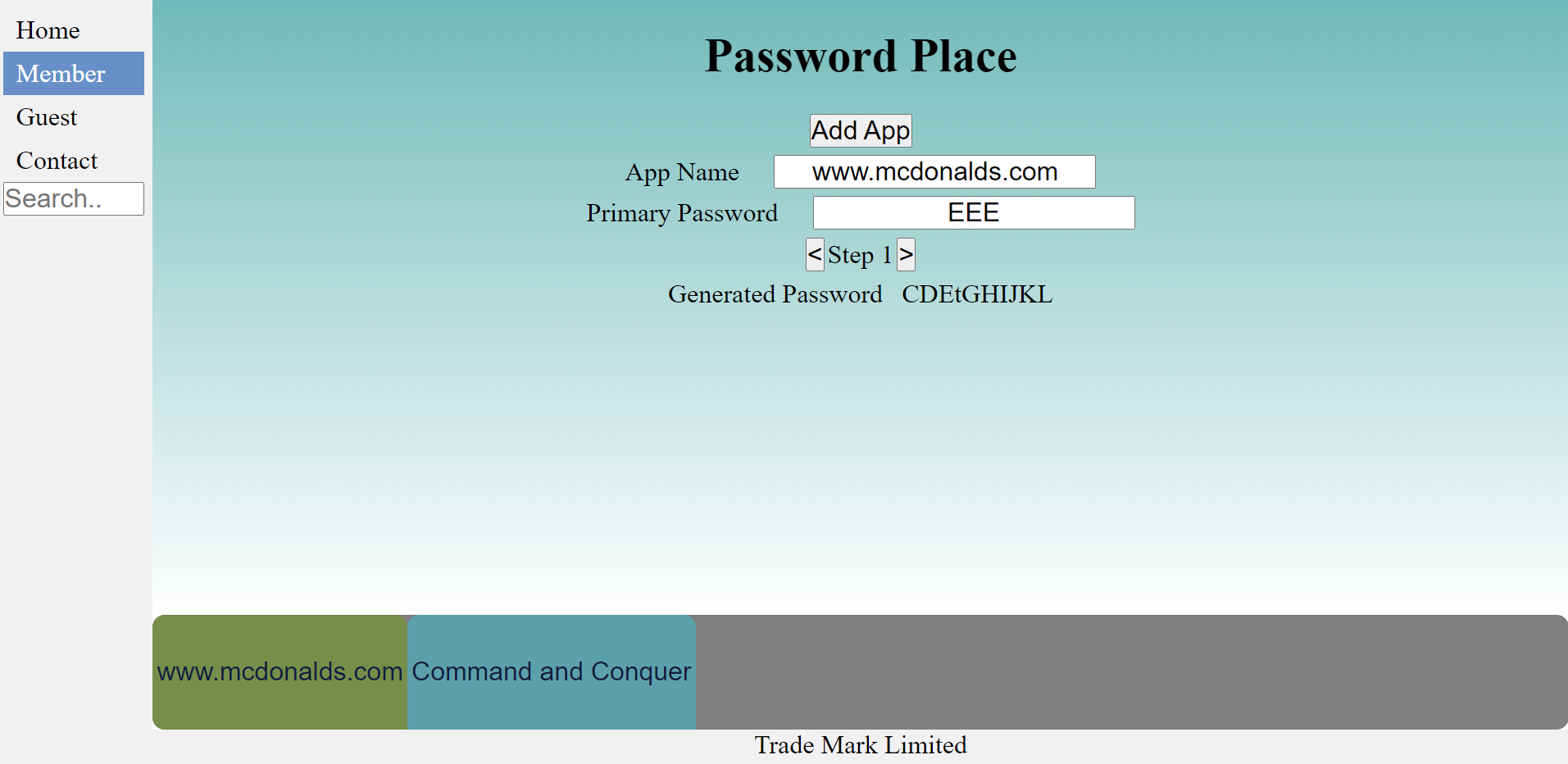
* Options menu is now selectable. Selecting an option updates the main form.
* The main form has been created. The main form scales well with changing screen sizes.
* Made the main password generation form interactable. Password can now be generated from the main form based on the data entered into the main form.
* Created a password generator that takes user input, converts the input to ASCII numbers, scrambles, then returns characters to form a password.
* Created placeholders for the guest, home, and contact page.

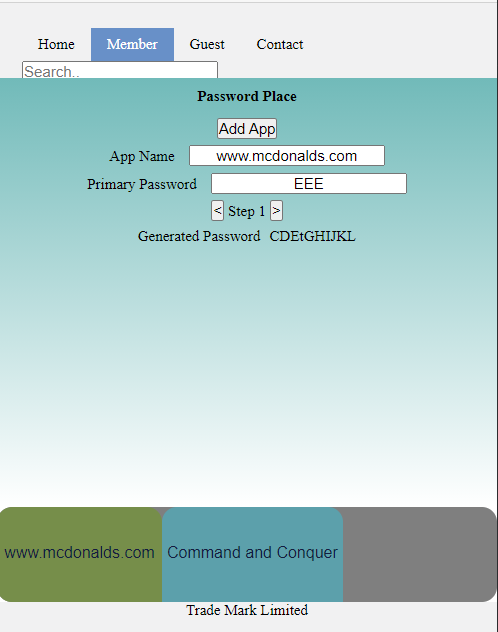
Proposal:

* UX/UI design completed showcasing the look and feel of Password Place.
* Major components and their intended behaviour completed. Explains the member’s page and guest page differences.
* System summary completed, including a flow chart that explains the workflow of the member page, the guest page, and the password generator and how these all interact.









To complete next week:

Proof of concept:

* Make each options its own Vue component. 1h 20m

Proposal:

* Add additional user stories. 40m

# Week 5

## Reflections

Principles of web design:

Availability: Measured by the number of devices, locations, browsers, operating systems and times of the day that the website is available for use to users. The more devices, locations, browsers, operating systems, and times of the day that the website is available to users, the greater the availability.

Performance in terms of web design is mainly determined by response time and transmit time. Response time is the time it takes between a user interacting with the website and to receive a response. Transmit time is the time that it will take user input to be received by the destination/ backend. Response time is a measure of two-way communication, the time for user input to reach the server, the server to process the user input and the server’s response to reach the client. Transmit time is one way communication, the time it takes users input to reach the server for processing. The lower the response and transmit time, the greater the performance.

Reliability: Reliability is measured by the number of times a website fails. The longer the average time between a website failing, the greater the reliability.

Scalability: Scalability is a measurement of how much performance of the website degrades as the number of users increases. Generally, performance will degrade as the amount of the users increases, however the website needs to at least maintain a satisfactory level of performance for the website to be considered sufficiently scalable.

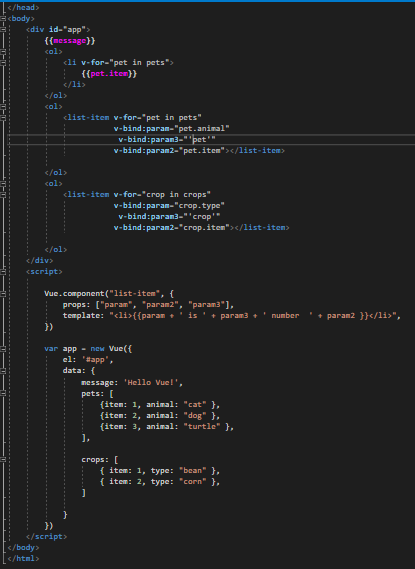
Manageability: Manageability is measured by how difficult it is to modify/ change the website.

Costs: Cost is the amount of financial resources required to create and maintain the website. The goal is to keep cost as low as possible whilst keeping availability, performance, reliability, scalability and manageability as high as possible.

Core components of web applications: All web applications have a front end, backend, and request layer. The front end of a website is made from JavaScript, HTML and CSS. By using JavaScript, HTML and CSS we can manipulate the DOM (Document Object Model). Generally a framework is used to make working with JavaScript, html and CSS easier and more efficient.

## Practical Tasks

### Task 1



In this task I learnt how to use Vue components. I also learnt how to pass a property to a Vue component, to make it more useful for re-use. Above you can see that I have reused the same Vue component for two different lists. I noticed that if I gave any of the list items a property or parameter called key, Vue seemed to fail. I wonder if key is a reserved name.

### Task 2

The Vue-Model example that I found on the Vue getting started page will be useful for the main form on my website. Users will be able to put in the name of the application they wish to generate a password for. The to do-item Vue component example will also provide a good basis for a Vue component for stored user apps for my assignment 1.

### Task 3

For this task I added a counter Vue component. In completing this task, I learnt the difference between created Vue values locally and globally. The count variable declared in the app Vue component is declared globally. Whilst the counter variable declared in the counter Vue component is declared locally. By declaring the Value within a function’s return, we make it a local variable.

### 

### Task 4

* Component Registration: Component registration allows components to be accessible in other view components templates. Globally registering a component makes that component available to any view components template. Locally registering a component allows us to choose whether that component should be accessible in another components template from that other component.
* Props allow us to pass values to a Vue component. This allows the same Vue component to represent different data and be used in various situations.
* Custom events: Custom events in Vue allow us the ability to emit our own custom events, changing their behaviour.
* Slots allow the content of a component to render differently based on how the component is used.
* Dynamic and async components. Dynamic components cache their state after events, such as switching tabs. Asynchronous components are loaded only as they are required.
* Edge cases: Edge cases are situations where it is appropriate to break the normal Vue rules and conventions to solve them.

# Week 6

## Reflections

The focus of this week was learning more about v-bindings and modifiers.

Bindings:

V-bind is a one-way form of binding. This means that changes to a JavaScript variable will update the Html for any HTML element that has been bound to the JavaScript variable. V-model is a two-way binding, it is used to update JavaScript variables based on user input. V-model will keep JavaScript and HTML in sync. This means that if a user changes the text of an input field of a form, and the form has the V-model directive “v-model=’test’”, the variable test on the JavaScript side will be assigned the value of the text that the user entered. Likewise, if the user selected a button that activates a method which runs the code test = “HiFive”, then the input field will display the value “HiFive”. Html and JavaScript will be kept in sync, modifying one will affect the other.

V-model makes use of different properties and events based on the input element the directive is used on.

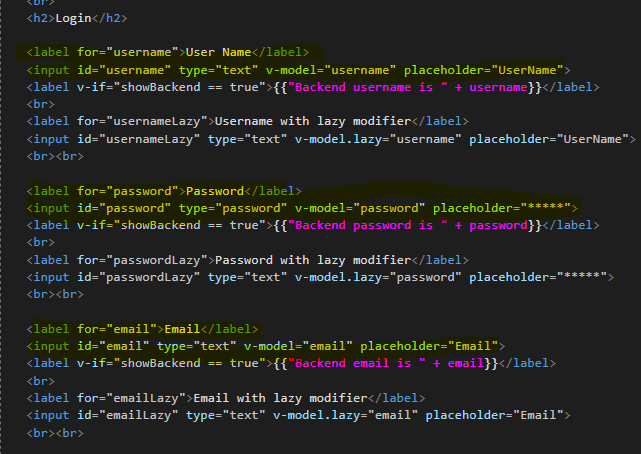
|  |  |  |
| --- | --- | --- |
| **Element type** | **Property** | **Event** |
| Text and text area | Value | Input |
| Checkboxes and radio buttons | Checked | Change |
| Select fields | Value | Change |

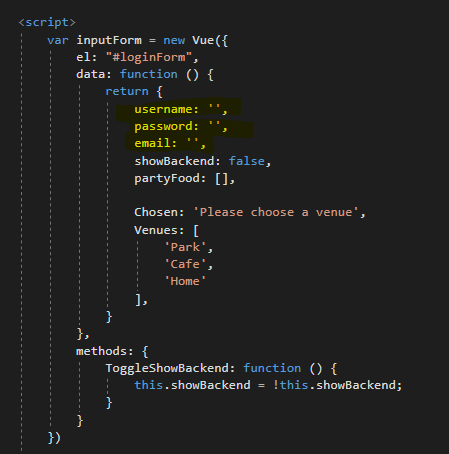
Modifiers:

V-model can also be used with modifiers. These modifiers are the lazy modifier, the number modifier and the trim modifier. The lazy modifier results in the JavaScript value only being updated when the user hits the return key, rather than being updated on key up when entering input. The trim modifier removes whitespace from the text that the user has entered the input element before assigning the value to the JavaScript variable that the element is bound to. The number casts the value that the user has entered into the input element to number, before assigning the value to the JavaScript variable that the element is bound to. Modifiers are applied with the following syntax: v-Model.Modifier=”Value”.

## Practical Tasks

### Task 1



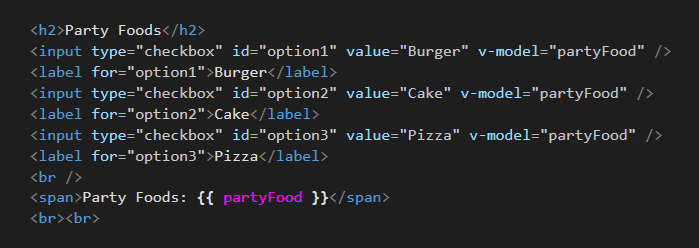


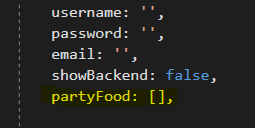


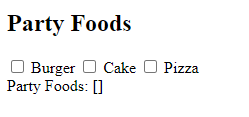
I completed the requirements for task 1, 2, 3 and 4 within the same index.html file. Above I have highlighted all the code that is relevant to task 1. The first image shows the html elements that are bound to JavaScript variables by using V-model. This creates a user login form which allows the user to input their desired username, password, and email. As the user inputs into any of these fields, the JavaScript variable that the element is bound to will be updated in real time. We can see that as a value is entered into the input field, the backend value is instantly updated.

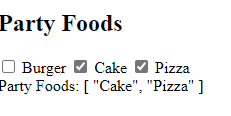


### Task 2



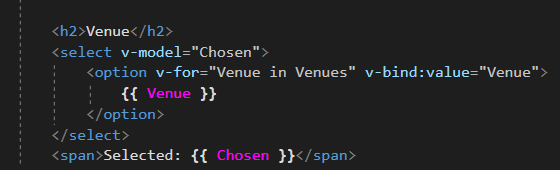


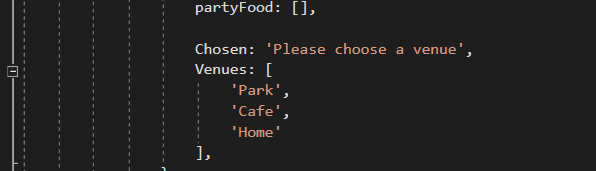


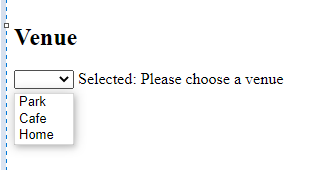


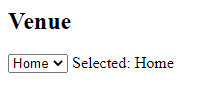
This task required me to allow the user to input information via checkboxes. To achieve this, I first needed to create a JavaScript array variable in the data method of my Vue component. The array does not require elements to be explicitly set on the JavaScript side. Vue will automatically push the value of a checkbox element that is bound to the JavaScript array when the user ticks the checkbox. When the user unticks the checkbox, the value will be removed from the JavaScript array.

### Task 3





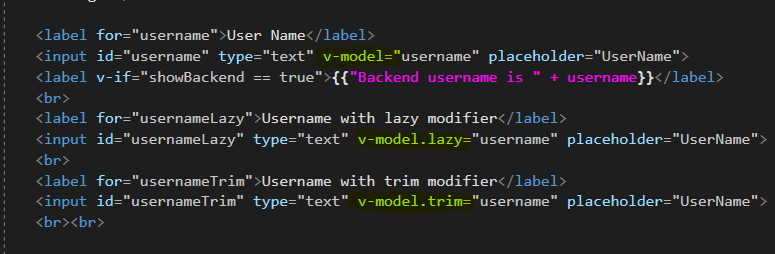




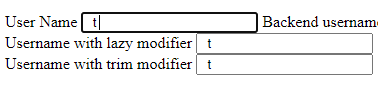
This task required that I dynamically render options using V-for. I demonstrated this by creating a drop-down menu where the user can select from a list of options. The option the user selects will then be displayed next to the drop-down menu. To achieve this, I had to create an array of values and an object to store the selected value in my Vue-Component. I then created a drop-down list with a select tag containing an option tag. Instead of hard coding every option, I simply used V-for to render an option element for every element within the array “Venues”. I bound the select elements value the “Chosen” JavaScript variable using V-model, so that the value chosen by the user would be assigned to this variable.

### Task 4

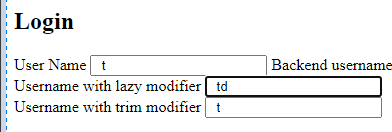
This task required me to demonstrate how to use modifiers and what modifiers do. Below is an example of input without a modifier, input with the lazy modifier and input with the trim modifier.



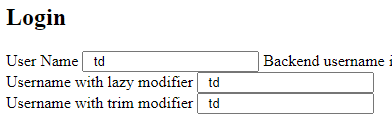
If I type “ t” into the input field without a modifier we get the following:



If I now type type the letter d into the input box with the lazy modifier, we get this:

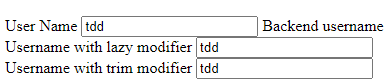


Notice that the other input fields have different values to the input field with the lazy modifier? That is because the JavaScript value that the element is bound to has not yet been updated. If I now hit the enter key, we get this:



The lazy modifier does not modify the value that is bound to it in real time. Instead, the lazy modifier requires the user to hit the return key before the value is updated in the Vue Component/ JavaScript side.

Now if we simply select the input field with the trim modifier and add the letter d we get this:



You will notice that the leading white space has now been removed. That is because the trim modifier removes any leading or trailing white space for the user’s input when it updates the JavaScript variable that is bound to it.

# Week 7

## Reflections

This week we learnt more about component including, component registration, props, custom events and how to manipulate content with dynamic and asynchronous.

Registration:

We globally register a component using



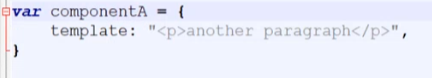
When a Vue component is globally registered, the Vue component is then available from any other component. There are two types of name casing we can use for the name of a Vue component, there are kebab case and pascal case. Html treats the two different casings differently.

Kebab Case: Is written as: “my-component-name”

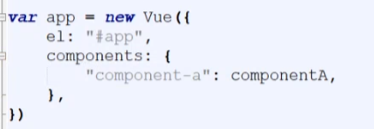
Pascal Case: Is written as: “MyComponentName”

If we use kebab case, the component can be used directly within the DOM. If we use Pascal case, the component can be used within another components template, but not directly in the dom.

In order to locally register a Vue component, we create the Vue component like so:



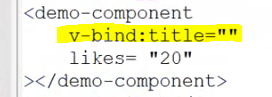
And then we need to register the component to any components that will use it like so:



Props:

Props allow a parent component to pass information to a child component. This allows component to have dynamic behaviour.

To add a prop to a component, we need to declare all of the props in the component like so



When another component uses this component, a value can now be given for the prop like so.



The component with he prop uses the prop in its own template.



The prop can be directly rendered or used in conditionals. For example it would be valid to use the prop like so.

<h1 v-if="title == ‘Mr’">The title is Mr</h1>

We can enforce the type that the prop will accept from the parent component. If a prop has a specified type and a different type is assigned to it by the parent component, an error will be thrown. We can specify the type of a prop with the following syntax.



Props must be lower case. This is because html is not case sensitive, the variable will be treated as lower case in HTML. If the variable is upper case in JavaScript, it can cause the value to not update in HTML as there will be a mismatch.

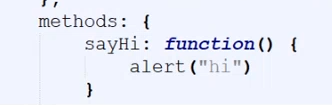
Custom Events:

A custom event in Vue is triggered with the syntax: $emit(‘event-name’). The event needs some way that it can be triggered, either automatically when the website loads or based on user input. Below is an example of a button that would trigger the event “say-hi” after being clicked.



To take some form of action when an event is triggered, there needs to be a listener associated with the event. Just like how above we have a listener that is listening for the event “click” and that emits the “say-hi” event when on click is triggered, we need a listener to listen for “say-hi” and to provide instructions on what to do when “say-hi” has been triggered. Just like with the “click” event listener, we can create a listener for “say-hi” using v-on. Below is an example of an event listener for “say-hi” that will call the method “sayHi” when the event “say-hi” is triggered.



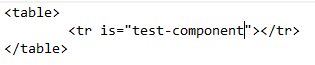


Dynamic and Async Components:

HTML has rules prohibiting certain types of elements from being contained within other certain elements. An example of this it the <table> tag in html. The <table> tagged element cannot have an <input> tagged element put directly inside of it, the <table> tagged element would require an element with a tag such as <tr>. This means that if we have a custom component called “test-component”, the following code would not be valid.



We can however get around this limitation by using the attribute “is”. The “is” attribute allows us to set the tag of an element to a certain type, however, to provide the instruction that the content that we want to use is really our custom component. Below is an example of how we would get around the problem that was demonstrated above. By using an element with the tag “tr” within the tag “table”, we are not going to violate the restrictions set by the table element. However, we are also providing the instruction that our “test-component” Is what is to be used.



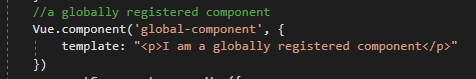
Async Components are components that are used to interact with a backend or server. As it can take time to receive a response from the server, Async components sent a request to the server using a call back function. This means that the rest of the application will continue to be processed while the Async components waits for a response from the server. The Async component will process the information received from the server with a given function based on whether the communication with the server was successful or not. If components that communicated with the server could not act in an asynchronous manner, the application would be brought to a halt until either a response from the server was revived, or the timeout time was reached. Below is an example of an Asynchronous component.



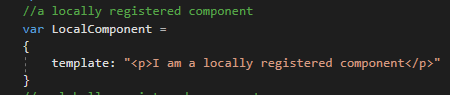
## Practical Tasks:

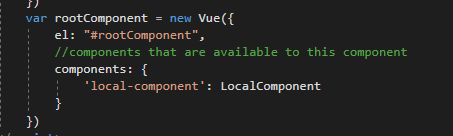
### Task 1

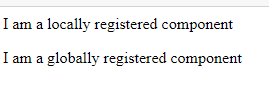
This task required a demonstration of how to register a Vue component globally and how to register a Vue component locally. To register a Vue component globally the Vue component needs to be created using the function Vue.component, providing the component’s name and contents as parameters just like in the screenshot below.

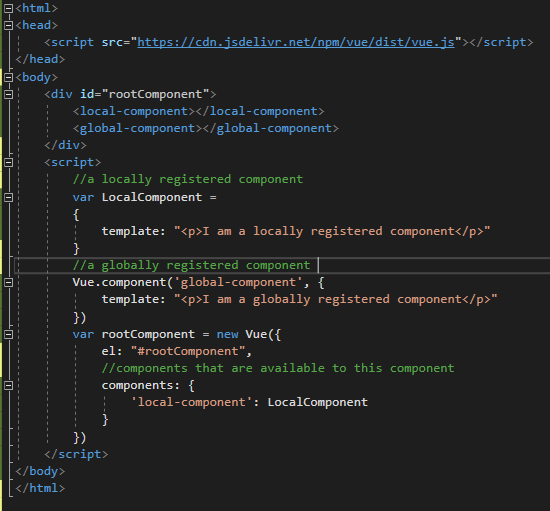


Once the component has been globally registered it is then available from all other components. There are no other steps for globally registering a component. As a globally registered component will use the name given to it directly in the DOM, we need to use kebab case for the name.

To locally register a component, the component is created using a similar syntax to how we would create any other JavaScript object. The name that we give the variable does not need to be in kebab case as there is the dom will not use the name of the variable. 

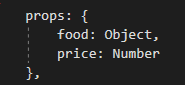
After the variable is created, any component that will use this local component needs to declare the local component within it’s acessable components by providing a name for the component within the dom as we can see below. 

Locally registered components are only accessable to components that have declared them. Below is an example of the usage of both a globally and locally registered components.  




### Task 2

This task required a demontration of how to use props in Vue. Props allow parent components to pass information to child components. This allows the child component to behave dynamically. To use props, the component that will receive the props needs to declare them.



The component can then use these props within its own template in the same way that a component can use variables defined within its own data such as below.



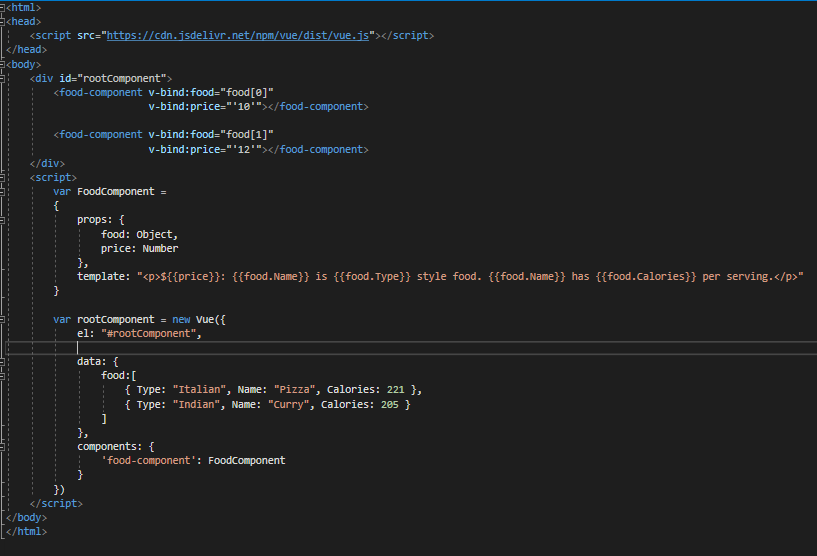
To demonstrate this, I have created a component called “FoodComponent”

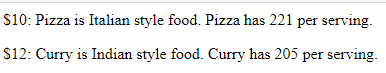


To pass the props to “FoodComponent” we simply need use V-bind within any tag for this component such as below.



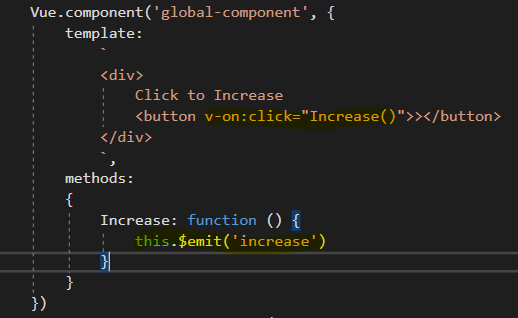
Below is an example of my demonstration of this and the output.



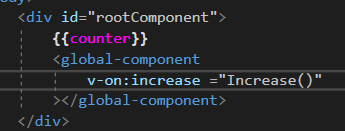


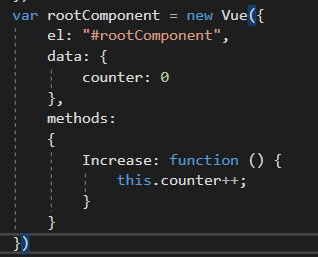
### Task 3

For this task I had to demonstrate how to use a custom event. Custom events are emitted using this.$emit(‘event-name’) (if emitting from a function). This alerts any event listeners that are listening for this event, that the event has been triggered. This example shows an emit being triggered from a component called “global-component”. In this example, the user must select the button, which triggers the “click” event. In response to the click event, the method “Increase()” is called. Increase then emits the event “increase”.



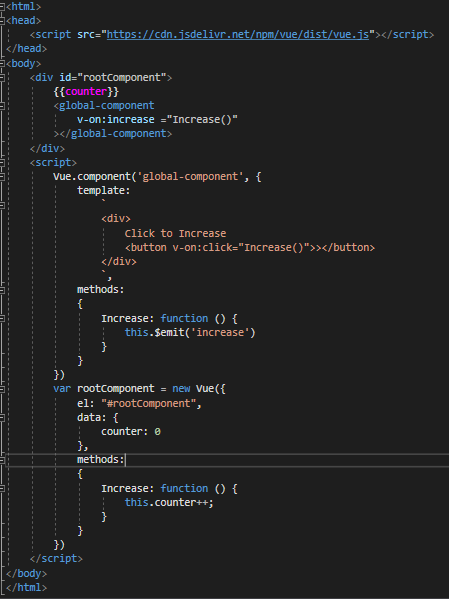
The increase event has been emitted above, but unless there is an event listener listening for the vent, nothing will happen. The root Vue component renders “global-component” and has an event listener within global-component’s tag using v-on. The event listener states the name of the event to listen for and the action to take if the event is triggered. In this case the event being listened for is “increase” and the action to take it to call the method Increase() on the root component.





This results in the counter increasing on the root component whenever the button is clicked on “global-component”.

The full code for this and the result can be seen below



Below we can see the out before and after pushing the button.





### Task 4

Slots are a reserved space that acts as a placeholder. They are very similar to props. Whilst props involve passing JavaScript data to Vue components, slots allow us to specific the elements that will appear within a certain section of the DOM/ Vue component template.

A Vue component can use slots within its templates by using the tag “slot”. Slots have an optional “name” attribute. If a slot has not had it’s name attribute set, slot will be treated as the default slot, meaning that it is the slot that is used if no name is specified when inserting content into the Vue component. Below is an example of a Vue component that has a template with two slots, these slots are “slot1” and “slot2”. We can see that both slots have <p> elements in them by default. The data that we see here is what will be used if the slot is not injected into when using the component from another component.



We can insert into the slot by using the component above in another component. We can either do this directly in the DOM or from the other component’s template. In order to insert into the slot we simply need to use an element within the component’s tag and provide the element with a value for the attribute “slot”. Below we can see an example of the above component being used three times, the first time both slots are injected into, the second time only one slot is injected into and the third time no slots are injected into.



The output of this is below. Take note that the default value for the slot is used if the slot is not injected into.

