

# PRACTICAL PORTFOLIO

---

Evidence of Learning, Up-skilling and Project Progress

Vittorio Truglio - 219179512

SIT120 Introduction to Responsive Web Apps

Assessment 2: Practical Portfolio and Project Progress

# Marking Justification

In completing SIT120 Assessment 2, I am aiming for a Distinction in the submission and eventual graded return of this assessment task. I believe challenging myself will allow me to not only bring out the best of my own personal abilities, but also fully receive the maximized benefits of this unit, as I delve into a myriad of differing elements of responsive web app development. However, I also acknowledge being in my first year of my Bachelor of Information Technology, with no previous experience in coding with web design or understanding the fundamental theory concepts of it, I need to keep a levelheaded approach to this unit with the ability to balance it with its counterparts. As such, a Distinction level grade I feel will best allow me to achieve these goals with a sense of worth and fulfillment.

The aim of this practical portfolio is to reflect a standard befitting of the aforementioned target grade in mind, doing so by providing task results through screenshots and GitHub links to relevant programming documentation. Further to this, each week will provide an in-depth reflection of what content was learned as to showcase my own personal understanding of the provided material and lab exercises.

# Week 1

## Reflections

The first week of SIT120 covers an introductory approach to web applications and the basics of web design operating in tandem with it. As such, this week focused on developing a base understanding and skill foundation of technologies and web languages.

The first amongst these being HTML, standing for Hypertext Markup Language. This technology is dedicated to providing the basic structure of sites, being the focus of Pass Task 1 in the utilisation of HTML tags for a web page.

CSS, standing for Cascading Style Sheets, is the programming language focused on control presentation and webpage formatting and layout. This language, alongside JavaScript, is explained as the enhancing technology to HTML, through visual look and stylization. This particular language was the focus of Pass Task 2, providing the aforementioned 'style' to the task's webpage.

JavaScript is the final of the three provided languages, being fully utilised to control the behaviour of elements provided by a developer on their webpage. This language was the focus of Pass Task 3, through the modification of website content and its responsiveness to user action.

Further building upon this framework, the remaining Pass and then Credit tasks focused on critical and additional skills past the base understanding of the three web development languages. Due to the need of GitHub throughout this unit, Week 1 showcased the installation process as well as the use of the software in creating a repository and 'pushing' code to upload it to my own personal storage. Credit Task 5 then offered a look into additional content, being the creation of a basic Todo component achieved through the use of Vue.js, a view model front end for JavaScript framework.

## Lab Exercise Answers

### Pass Task 1 – Create an HTML Page

#### Screenshots:

#### Code

```
1 <!DOCTYPE html>
2 <html>
3
4   <head>
5     <title>My First Website</title>
6   </head>
7
8   <body>
9     <h1>My First Web Page</h1>
10    <h2>Websites are built with HTML</h2>
11    <p>I created a webpage</p>
12    
13  </body>
14 </html>
```

#### Output

##### My First Web Page

Websites are built with HTML

I created a webpage



### Pass Task 2 – Add CSS to your HTML page

#### Screenshots:

#### Code

```
1 <!DOCTYPE html>
2 <html>
3
4   <head>
5     <title>My First Website</title>
6   </head>
7
8   <style>
9     body {
10       background-color: lightgray;
11     }
12   </style>
13   <body>
14     <h1 style="font-size:36px;text-align:center;">My First Web Page</h1>
15     <link rel="stylesheet" href="mystyle.css">
16     <h2>Websites are built with HTML</h2>
17     <p style="color:red;">I created a webpage</p>
18     
19   </body>
20
21 </html>
```

```
# mystyle.css X
# mystyle.css > ...
1
2   h2 {
3     color: blueviolet;
4     margin-left: 20px;
5   }
```

## Output



## Pass Task 3 – Adding JavaScript

### Screenshots:

### Code

```
1  <!DOCTYPE html>
2  <html>
3
4    <head>
5      <title>My First Website</title>
6    </head>
7
8    <style>
9      body {
10        background-color: lightgray;
11      }
12    </style>
13    <body>
14      <h1 style="font-size:36px;text-align:center;">My First Web Page</h1>
15      <link rel="stylesheet" href="mystyle.css">
16      <h2>Websites are built with HTML</h2>
17      <p style="color:red;">I created a webpage</p>
18      <p id="Live_Time"></p>
19      
20    </body>
21    <script>
22      const d = new Date();
23      document.getElementById("Live_Time").innerHTML = d;
24    </script>
25  </html>
```

```
# mystyle.css X
# mystyle.css > ...
1
2   h2 {
3     color: blueviolet;
4     margin-left: 20px;
5   }
```

## Output



## Credit Task 5 – Vue.js Framework

### Screenshots:

#### Code

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4   <head>
5     <meta charset="UTF-8" />
6     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
7     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
8     <title>Week 1 Task 5 Vue To Do List</title>
9     <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
10  </head>
11  <body>
12    <div id="app">
13
14      <div
15        v-for="todo in todos"
16        :key="todo.title"
17        :style="{ textDecoration: todo.checked? 'line-through': 'none' }">
18
19        <input type="checkbox" v-model="todo.checked" />
20        {{todo.title}}
21      </div>
22    </div>
23  </body>
24
25 </html>
26
27 <script src="./VueToDoList.js"></script>
28
```

```
JS VueToDoList.js X
JS VueToDoList.js > ...
1  let vue = new Vue({
2    el: "#app",
3    data: {
4      todos: [
5        {title: 'SIT120 Assignment 1', checked: true},
6        {title: 'SIT120 Assignment 2', checked: false},
7        {title: 'SIT120 Assignment 3', checked: false},
8      ],
9    },
10  });
```

#### Output

- ☒ ~~SIT120 Assignment 1~~
- ☒ ~~SIT120 Assignment 2~~
- ☐ SIT120 Assignment 3

#### GitHub Link:

<https://github.com/219179512/SIT120-Weekly-Tasks/tree/Week-1>

# Week 2

## Reflections

In week 2, the focus of the provided tasks is to build upon the base foundation constructed from the previous weeks learning. As such, this began with a focus on the new concept of 'responsive web design' and its incorporation into a webpage through the use of HTML, CSS and JavaScript.

Illustrating responsive web design first begins with an understanding of the concept, as showcased through Pass Task 1, where I offered a more direct reflection upon how this mode of web design operates and adds value to the experience for users. Pass Task 2 then required me to showcase this understanding through a more practical approach, where I utilised CSS to implement visual improvements to my page and flexible screen width measurements. As then displayed below, this resulted in the ability to display my webpage from a multitude of desktop and mobile devices with ease and no display or resolution faults. Finally, Credit Task 3 focuses on predicted user requirements of my prototype web page, with user stories specifying where my needs lie and the use of Figma to visualise solutions to the problems.

## Lab Exercise Answers

### Pass Task 1 – Understand about responsive web Pages and apps

#### Reflection:

Responsive web design, by definition, is the innate response a designed web page has to the needs of users and the devices they are accessing said page on (LePage and Andrew, 2020). In practice, this can be applied to a myriad of devices, changing in appearance to specifically suit a variety of technologies, from phones to full screen monitors. For example, a phone user would be provided access to a web page with its content being visible through a single column view, while a tablet user would be provided with two columns to best appeal to their device.

### Pass Task 2 – Understanding how responsive web design can be implemented using CSS

#### Reflection:

The implementation of proper responsive web design primarily focuses on the element of screen resolution, and fitting a page not to specific given values, but rather to a flexible percentage mode of measurement. When coding to consider for this, namely in the illustration of columns on my web page, as provided below, it was necessary to set the width of these columns to the percentage value of '33.3%' along with awarded float and padding values. By doing this, I could ensure that responsive web design was implemented



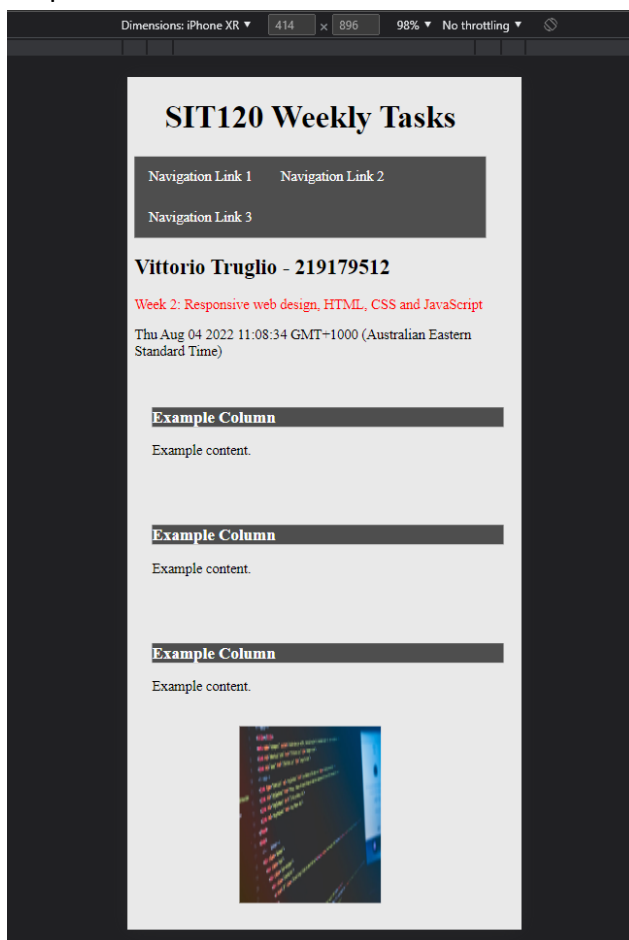
in changing for the value of a screen, be it on a desktop or mobile device, rather than a specific size that would not account for variation.

## Screenshots:

## Code

```
31     .column {
32         float: left;
33         width: 33.33%;
34         padding: 20px;
35     }
36
37     .row:after {
38         content: "";
39         display: table;
40         clear: both;
41     }
42
43     @media screen and (max-width:500px) {
44         .column {
45             width: 100%;
46         }
47     }
48
```

## Output





### Credit Task 3 – User stories and UI/UX design for your project

#### Reflection:

User stories are important both during and post the production stage of web development and design. User stories offer critical insights into the needs of the target demographic that a designed web page is steered to, meaning adhering to the desires through their statements are one of the most valuable sources of information on what features to implement into web design. A UI/UX prototype is then just as significant, as it allows for a testing bed from which a web developer can measure how well they have met the requirements of these prioritised user stories and thus, what level of success they can receive from their endeavors.

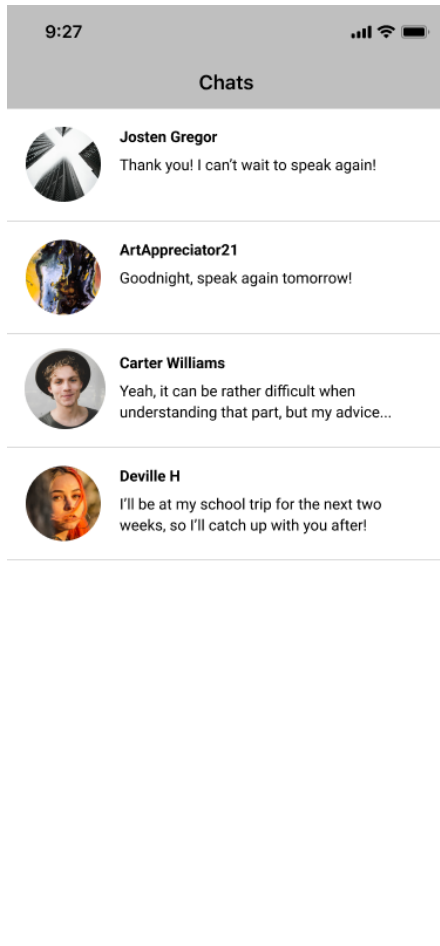
#### Screenshots:

##### User Stories

Statement	Acceptance Criteria	Priority
As a user I want to be able to see a history of people I have chatted with.	<ol style="list-style-type: none"> <li>1. User can see identifying features of other users by name and profile picture.</li> <li>2. User can see last message sent or received from identified user.</li> </ol>	Priority: 1/High
As a user I want to be able to access this webpage on my mobile device when I'm away from home.	<ol style="list-style-type: none"> <li>1. Website is designed within code to be responsive, providing design features that operate on differing device displays.</li> </ol>	Priority: 1/High

	<ol style="list-style-type: none"> <li>2. All interactable features still operate correctly with dimension changes on displays.</li> <li>3. User can access and navigate webpage without fault.</li> </ol>	
--	--	--

## Figma Demo



## GitHub Link:

<https://github.com/219179512/SIT120-Weekly-Tasks/tree/Week-2>

# Week 3

## Reflections

The content of week 3 covers a wide variety of methods, all applied to JavaScript string, number, array and date values. The intention of this week was to understand how some values that would standalone only provide some worth, or other values that would provide worth in their own right, would all be greatly built upon with the application to the aforementioned methods.

The first pass task elucidated upon JavaScript string methods, being the utilisation of methods to alter and give further value to immediately primitive values. In an application to strings within a webpage, these methods could showcase how these values could interact with one another, be counted or weighed upon to their numerical value or be joined or altered. The next pass task then applied differing methods to standalone numerical values, as well as arrays. This allowed for similar features of adding, removing and altering the nature of string or numerical values that were provided through arrays. Finally, the credit task for this week involved a similar application of JavaScript related methods to displaying times and dates. Here, through the utilisation of flexible methodology, numerical dates for days, months, years and a combination of these as a statement can be written to a console with the appropriate date methods. Similar to the tasks before, this also allows for the altering of numerical values to display desired measurements of time where needed.

## Lab Exercise Answers

### Pass Task 1 – JavaScript String Methods

#### Reflection:

JavaScript string methods provide a means for allowing primitive values to adopt methods and properties they would otherwise be unable to utilise. As showcased below, this allows for adding quality to this element of web design, be it through adding numerical values to text-based ones or all of the additional features this then allows for.

## Screenshots:

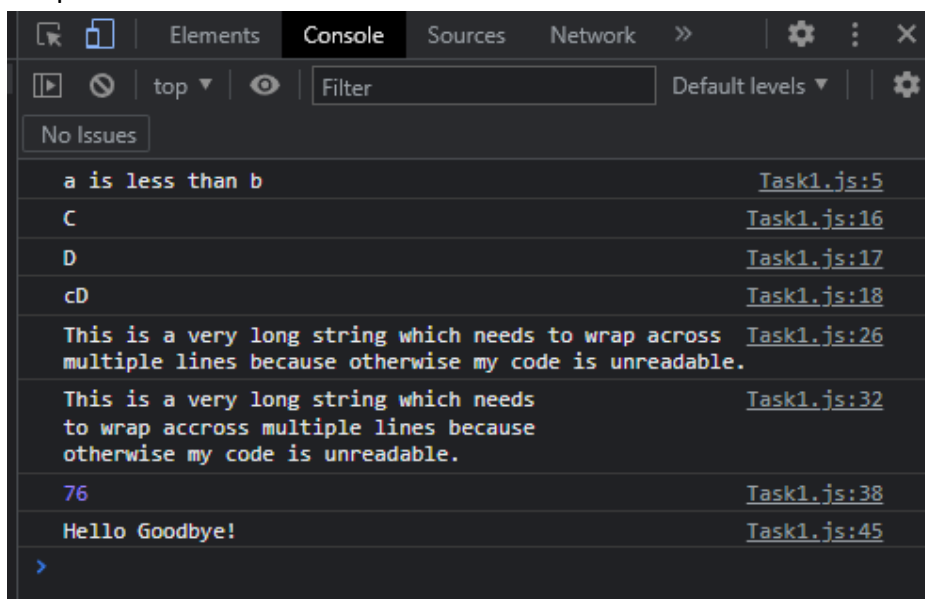
### Code

```
1 // String Method 1: Comparing Strings
2 let a = 'a';
3 let b = 'b';
4 if (a < b) {
5   console.log(a + ' is less than ' + b);
6 } else if (a > b) {
7   console.log(a + ' is greater than ' + b);
8 } else {
9   console.log(a + ' and ' + b + ' are equal.');
```

```
10 }
11
12 // String Method 2: Lowercase and uppercase
13 let c = 'c';
14 let d = 'D';
15
16 console.log(c.toUpperCase());
17 console.log(d.toUpperCase());
18 console.log(c + d);
19
20 // String Method 3: How to write long strings
21 let longString =
22   'This is a very long string which needs ' +
23   'to wrap across multiple lines because ' +
24   'otherwise my code is unreadable.';
25
26 console.log(longString);
27 longString =
28   'This is a very long string which needs \
29   to wrap accross multiple lines because \
30   otherwise my code is unreadable. ';
```

```
31
32   console.log(longString);
33
34 // String Method 4: Return the length of a string
35 let countString = "This is a very long string which needs to have its character count returned."
36 let length = countString.length;
37
38 console.log(countString.length);
39
40 // String Method 5: Joins two or more strings
41 let string1 = "Hello";
42 let string2 = "Goodbye!";
43 let string3 = string1.concat(" ", string2);
44
45 console.log(string3);
```

### Output



## Pass Task 2 – JavaScript Number, Number Methods and Array Methods

### Reflection:

Utilising varying array methods, arrays and the variables they contain can be altered and displayed in all manners of fashion. As illustrated below, with the respective array method, variables can be removed from an array, added to it, sliced to receive a sample of data and more where desired.

### Screenshots:

#### Code

```
1 // String Array
2 let fish = ['Carp', 'Salmon', 'Amberjack', 'Tuna'];
3 console.log(fish);
4
5 // Find Method
6 console.log(fish.find((fish) => fish === 'Amberjack'));
7 console.log(fish.find((fish) => fish === 'Amberjack1'));
8
9 // Push Method
10 // Push adds an element to the end of the array and then returns the new length of the array
11 console.log(fish.push('Sardine'));
12 console.log(fish);
13
14 // Pop Method
15 // Pop removes the last element of the array and then returns the removed element
16 console.log(fish.pop());
17 console.log(fish);
18
19 // Number Array
20 let numbers = [6,7,2,4,8];
21
22 // Sort method
23 // Sort method sorts the array
24 console.log(numbers.sort());
25
26 // Slice method
27 // Slice method cuts out part of the array
28 console.log(numbers.slice(1, 3));
```

## Output

```
Task2.js:3
▼ (4) ['Carp', 'Salmon', 'Amberjack', 'Tuna']
  0: "Carp"
  1: "Salmon"
  2: "Amberjack"
  3: "Tuna"
  length: 4
  ▶ [[Prototype]]: Array(0)

Amberjack Task2.js:6
undefined Task2.js:7
5 Task2.js:11
Task2.js:12
▼ (5) ['Carp', 'Salmon', 'Amberjack', 'Tuna', 'Sardine']
  0: "Carp"
  1: "Salmon"
  2: "Amberjack"
  3: "Tuna"
  length: 4
  ▶ [[Prototype]]: Array(0)

Sardine Task2.js:16
▼ (4) ['Carp', 'Salmon', 'Amberjack', 'Tuna']
  0: "Carp"
  1: "Salmon"
  2: "Amberjack"
  3: "Tuna"
  length: 4
  ▶ [[Prototype]]: Array(0)

Task2.js:24
▼ (5) [2, 4, 6, 7, 8]
  0: 2
  1: 4
  2: 6
  3: 7
  4: 8
  length: 5
  ▶ [[Prototype]]: Array(0)

Task2.js:28
▼ (2) [4, 6]
  0: 4
  1: 6
  length: 2
  ▶ [[Prototype]]: Array(0)
```

## Credit Task 3 – JavaScript Date API

### Reflection:

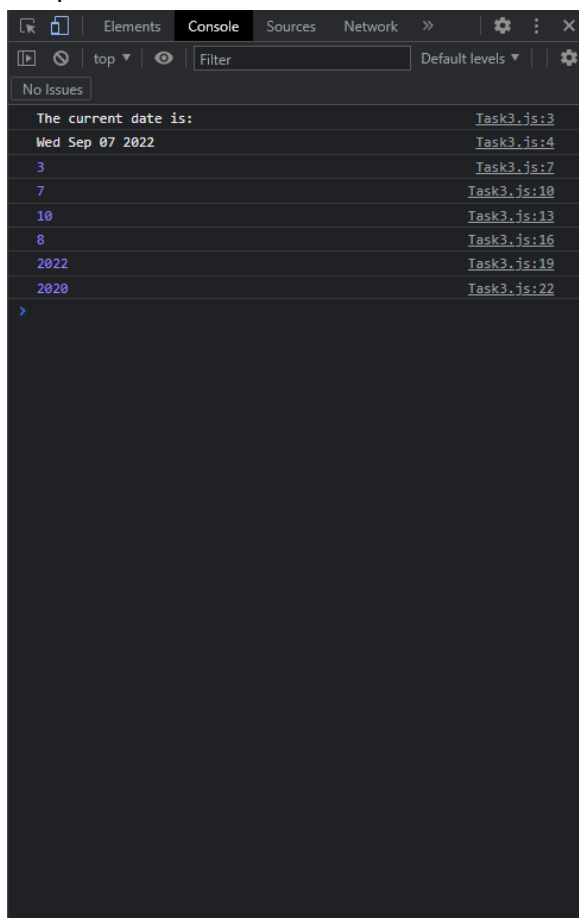
Provided through a myriad of differing Date methods, the current measurement of time elements is available to be returned and changed whenever required through JavaScript. Date methods are innately flexible, in providing single values for attributes such as the current year, month or day, and can be set to specific variables depending on the needs of the user.

## Screenshots:

### Code

```
1 // Date Method 1: Get current time as per constant value
2 const now = new Date(Date.now());
3 console.log('The current date is: ');
4 console.log(now.toString());
5
6 // Date Method 2: Get current weekday as a number
7 console.log(now.getDay());
8
9 // Date Method 3: Get current day value as a number
10 console.log(now.getDate());
11 // Alternatively, set date and return input value
12 now.setDate(10);
13 console.log(now.getDate());
14
15 // Date Method 4: Get current month as a number
16 console.log(now.getMonth());
17
18 // Date Method 5: Get current year as a number
19 console.log(now.getFullYear());
20 // Alternatively, set full year and return input
21 now.setFullYear(2020);
22 console.log(now.getFullYear());
```

### Output



### GitHub Link:

<https://github.com/219179512/SIT120-Weekly-Tasks/tree/Week-3>



# Week 4

## Reflections

This week focused on enriching the capabilities of JavaScript through the utilisation of Vue, a frontend framework for the web design language (Vue.js, *Introduction*, n.d.). In its application, Vue provides a myriad of benefits in its unique contribution to web design beyond the scope of vanilla JavaScript. This sentiment was showcased throughout both pass and credit tasks.

Pass Task 1 for week 4 elucidated upon Vue's capabilities in regard to string blocks and templates, shifting from the need from rendering as required by JavaScript by itself to linking both data and the DOM to write text that is innately reactive (Vue.js, *Conditional Rendering*, n.d.). This provides a means of interacting with the content of the webpage through a created Vue instance, rather than requiring HTML to provide for the entirety of all input, rendering and output (Vue.js, *Conditional Rendering*, n.d.). Credit Task 2 then further build upon this reactive relation between Vue and string values, showcasing through condition rendering how block strings can be displayed through a variety of Vue based terminology. This provided another showing of the flexibility of the framework, be it through simply printing values through it instead of in a traditional sense or illustrating values depending on conditions as provided by myself.

## Lab Exercise Answers

### Pass Task 1 – Create basic Vue application

#### Reflection:

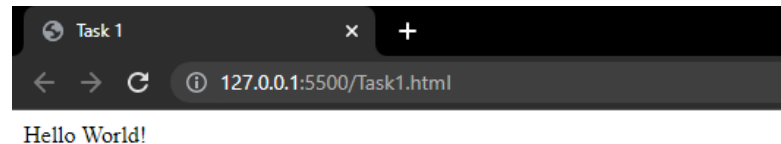
Utilising Vue as a frontend framework for JavaScript, this specific task showcased a simple example of how the features of this program could be implemented in web design. As the below code showcases, this is an example of a very simple Vue app, rendering a string template and linking the data and the DOM to make the implemented text now reactive (Vue.js, *Conditional Rendering*, n.d.).

## Screenshots:

### Code

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <title> Task 1 </title>
5    </head>
6    <body>
7      <div id="app"></div>
8    </body>
9  </html>
10 <script>
11   let text = 'test';
12   document.getElementById('app').innerHTML = text;
13   text = 'Hello World!';
14   document.getElementById('app').innerHTML = text;
15 </script>
```

### Output



## Credit Task 2 – Conditionals and Loops

### Reflection:

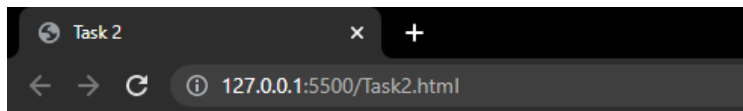
This task moved focus to conditional rendering as a provided feature of Vue utilisation through JavaScript. With the four provided features of v-if, v-else, v-show and v-for, there is no shortage in the flexibility of this rendering of block strings and means they can be achieved.

## Screenshots:

## Code

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <title> Task 2 </title>
5      <script src="https://cdn.jsdelivr.net/npm/vue@2.6.14/dist/vue.js"></script>
6    </head>
7    <body>
8      <div id="app">
9        <p><b> V-IF IMPLEMENTATION </b></p>
10       <p v-if="assignments[0].dueTomorrow">{{ assignments[0].name}}</p>
11       <p v-if="assignments[1].dueTomorrow">{{ assignments[1].name}}</p>
12       <p v-if="assignments[2].dueTomorrow">{{ assignments[2].name}}</p>
13       <p><b> V-ELSE IMPLEMENTATION </b></p>
14       <p v-if="dueTomorrowAssignments">
15         SIT120: Assignment 1 and SIT120: Assignment 3
16       </p>
17       <p v-else>
18         SIT120: Assignment 2
19       </p>
20       <p><b> V-SHOW IMPLEMENTATION </b></p>
21       <p v-show="dueTomorrowAssignments">SIT120: Assignment 1 and SIT120: Assignment 3</p>
22       <p><b> V-FOR IMPLEMENTATION </b></p>
23       <p v-for="assignment in dueTomorrowAssignments" :key="assignment.id">
24         {{ assignment.name }}
25       </p>
26     </div>
27   </body>
28 </html>
29 <script>
30   let app = new Vue({
31     el: '#app',
32     data: {
33       assignments: [
34         {
35           name: 'SIT120: Assignment 1',
36           dueTomorrow: true,
37           id: 1,
38         },
39         {
40           name: 'SIT120: Assignment 2',
41           dueTomorrow: false,
42           id: 2,
43         },
44         {
45           name: 'SIT120: Assignment 3',
46           dueTomorrow: true,
47           id: 3,
48         },
49       ],
50       computed: {
51         dueTomorrowAssignments() {
52           return this.assignments.filter((assignment) => assignment.dueTomorrow);
53         },
54       },
55     });
56   app.count = 2;
57 </script>
```

## Output



### **V-IF IMPLEMENTATION**

SIT120: Assignment 1

SIT120: Assignment 3

### **V-ELSE IMPLEMENTATION**

SIT120: Assignment 1 and SIT120: Assignment 3

### **V-SHOW IMPLEMENTATION**

SIT120: Assignment 1 and SIT120: Assignment 3

### **V-FOR IMPLEMENTATION**

SIT120: Assignment 1

SIT120: Assignment 3

## **GitHub Link:**

<https://github.com/219179512/SIT120-Weekly-Tasks/tree/Week-4>

# Week 5

## Reflections

The content of week 5 shifts focus to building upon a simplified and novice understanding of the Vue.js Framework. As such, the tasks provided this week concentrated on a theory-based approach to elucidating upon the basics and more intricate processes of Vue components and their application to web design.

Pass Task 1 for week 5 illustrated the theory concepts of Vue.js and its components. Defining these components, their processes and examples of what features they could then implement to a web page, the benefits provided by their application are extremely versatile and flexible in nature. Credit Task 2 then moved focus to the directive 'v-model', this feature being an example of a Vue component. Exploring upon this feature in the theory of its application and then seeing real examples of its utilisation in practice, this component makes up the backbone of many commonly seen features across the world wide web. Features such as placeholder text placed with individual text boxes or checkboxes individual placed or amongst multiple choices made up examples of this extremely reactive feature, and the use such Vue components contain in engaging users.

## Lab Exercise Answers

### Pass Task 1 – Components Basic

#### Reflection:

Vue components can be defined as 'reusable Vue instances with a name' (Vue.js, *Component Basics*, n.d.). These components are reusable Vue instances, as such, offering a range of options that once applied to them, can feature buttons or other similar type features that may enrich a web page.

### Credit Task 2 – Understanding Handling User Input

#### Reflection:

To handle user input, Vue can make use of the 'v-model' directive, this unique feature creating a two-way data binding relationship for 'form input, textarea and select elements' (Vue.js, *Form Input Bindings*, n.d.). As such, through its use of Vue internalized properties, it provides as an extremely versatile feature creating commonly utilised web design features such as placeholder text, checkboxes and select options from a drop-down menu to name a simple few (Vue.js, *Basic Usage*, n.d.).

# Week 6

## Reflections

Week 6 required me to complete tasks that closely related to that of the previous week. While week 5 concentrated on the theory of the v-model directive within the creation of Vue applications, the tasks detailed for this week shifted this focus to that of practical application.

Pass Task 1 first required an example of implementing the v-model directive for handling user input. An understanding of this was illustrated through the creation of a Vue application that required log-in details of an email address and password. Through the means in which the v-model directive reads and writes input data to a web page, this particular task was effective in showcasing how input data in its original or altered state would write to both the body code and web page simultaneously. Both Pass Task 2 and Credit Task 3 then moved to further the capabilities of the v-model directive in the creation of two more unique Vue applications. Firstly, the implementation of checkboxes into a webpage would allow for the displaying and collection of data rendered 'true' where they were selected by a user. Second to this, the final task's Vue application operated in tandem with the directive 'v-for' to again display information, but from a multitude of data variables rather than a standalone input value from the code or webpage.

## Lab Exercise Answers

### Pass Task 1 – Using v-model directive for Handling User Input (Form Input Bindings)

#### Reflection:

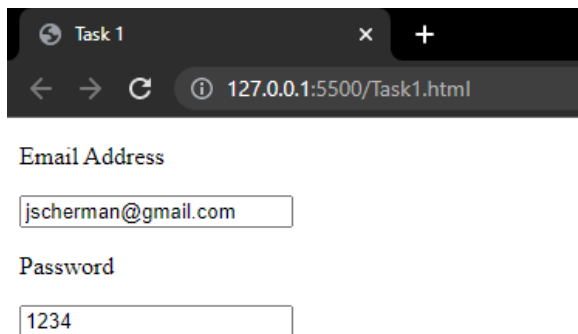
Utilising the v-model directive as required in this task, a Vue application is created which accepts a text string for an email and password variable. As showcased below, this directive allows not only for this input to be registered and assigned to the needs of the webpage, but also through this, allows for text changed in either the code or web page's input to change the values of one another based on their data binding relationship.

## Screenshots:

### Code

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title> Task 1 </title>
5     <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
6   </head>
7   <body>
8     <div id="app">
9       <p>Email Address</p>
10      <input
11        v-model="email"
12        type="text"
13        placeholder="Please enter your email address"
14      />
15      <p>Password</p>
16      <input
17        v-model="password"
18        type="text"
19        placeholder="Please enter your password"
20      />
21    </div>
22  </body>
23 </html>
24 <script>
25   let app = new Vue({
26     el: '#app',
27     data: {
28       email: 'jscherman@gmail.com',
29       password: '1234',
30     },
31   });
32   app.count = 2;
33 </script>
```

### Output



Task 1

Email Address

jscherman@gmail.com

Password

1234

## Pass Task 2 – Checkbox in your project

### Reflection:

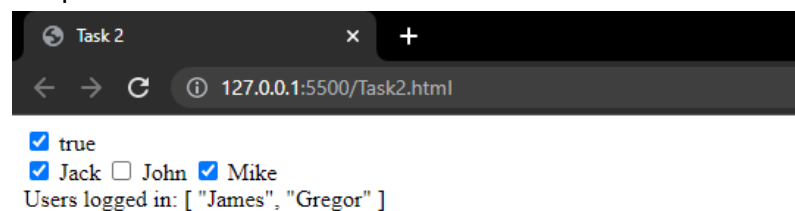
To create a Vue application with a checkbox, like the task before, the v-model directive is implemented into the given web code to display a checkbox and value assigned to it, that changes when clicked. In this respective, the flexibility of this directive is then further illustrated, as values assigned to a collective of checkboxes can be individually selected and placed to a data array.

### Screenshots:

#### Code

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title> Task 2 </title>
5     <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
6   </head>
7   <body>
8     <div id="app">
9       <input type="checkbox" id="checkbox" v-model="checked" />
10      <label for="checkbox">{{ checked }}</label>
11    </div></div>
12    <input type="checkbox" id="James" value="James" v-model="checkedNames">
13    <label for="jack">Jack</label>
14    <input type="checkbox" id="Josten" value="Josten" v-model="checkedNames">
15    <label for="john">John</label>
16    <input type="checkbox" id="Gregor" value="Gregor" v-model="checkedNames">
17    <label for="mike">Mike</label>
18    <br />
19    <span>Users logged in: {{ checkedNames }}</span>
20  </div>
21 </body>
22 </html>
23 <script>
24   let app = new Vue({
25     el: '#app',
26     data: {
27       checked: false,
28       checkedNames: [],
29     },
30   });
31   app.count = 2;
32 </script>
```

#### Output





### Credit Task 3 – Dynamic options rendering using v-for

#### Reflection:

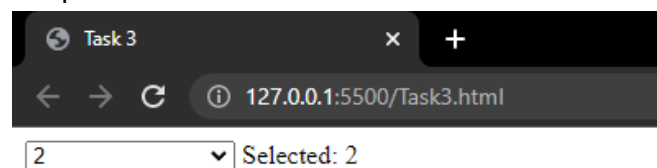
Utilising both v-model and v-for directives, a Vue application could be created with the ability of choosing an option value and displaying it to a page. As is visible in the image below, the utilisation v-model allowed for the creation of a Vue application with a drop-down menu and the v-for directive enabled this menu to select from a sample data and display the desired value.

#### Screenshots:

##### Code

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title> Task 3 </title>
5     <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
6   </head>
7   <body>
8     <div id="app">
9       <select v-model="selected">
10        <option value="" disabled> please select one</option>
11        <option v-for="n in 4">{{ n }}</option>
12      </select>
13      <span> Selected: {{selected}}</span>
14    </div>
15  </body>
16 </html>
17 <script>
18   let app = new Vue({
19     el: '#app',
20     data: {
21       selected: '',
22     },
23   });
24   app.count = 2;
25 </script>
```

##### Output



#### GitHub link:

<https://github.com/219179512/SIT120-Weekly-Tasks/tree/Week-6>

# Week 7

## Reflections

Week 7, being the final required week of the practical portfolio, concludes with an approach to components, their registration and the utilisation of coding props. With these new concepts in mind applied to this coding artefact, this week finalizes with a largely more in depth understanding of the Vue.js component of web design, and the mediums of its coding overall.

Pass Task 1 delved into the registration of components that could, as required here, portray strings to a web page. Beyond this, as required by the task, one component was registered as a local component to the page, while its counterpart was a global component. Finally, Credit Task 2 explored the implementation of coding props and their capabilities in transferring data from a parent component to that of a child one. In illustrating this relationship through a provided Vue application, string variables as data could be passed through this relationship and written to the web page as needed.

## Lab Exercise Answers

### Pass Task 1 – Component Registration

#### Reflection:

Components takes the form of independent code that is reusable at any point within a web pages larger code. As such, with two components created as showcased below, they can be implemented anywhere necessary as read to a webpage as a global component and a local component.

#### Screenshots:

##### Code

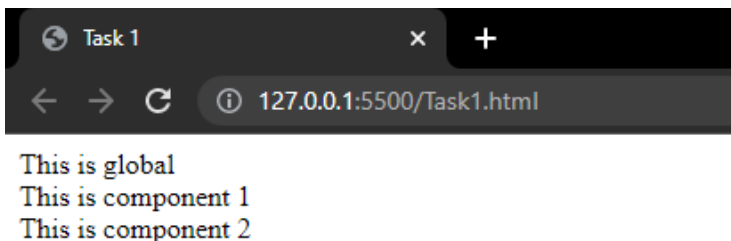
```
Task1.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <title> Task 1 </title>
5      <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
6    </head>
7    <body>
8      <div id="app">
9        <component1></component1>
10       <component2></component2>
11      </div>
12    </body>
13    <script src="./Task1.js"></script>
14  </html>
```

```

JS Task1.js > ...
1  Vue.component('global', {
2    |    template: '<div>This is global</div>',
3    |  });
4  var component1 = {
5    |    template: '<div><global></global>This is component 1</div>',
6    |  };
7  var component2 = {
8    |    template: '<div>{{message}}</div>',
9    |    data: function () {
10     |      |    return {
11     |      |      |    message: 'This is component 2',
12     |      |    };
13     |    },
14  };
15  var app = new Vue({
16    |    el: '#app',
17    |    data: {
18    |      |
19    |    },
20    |    components: {
21    |      |    component1,
22    |      |    component2,
23    |    },
24  });

```

## Output



Task 1

127.0.0.1:5500/Task1.html

This is global  
This is component 1  
This is component 2

## Credit Task 2 – Coding Props

### Reflection:

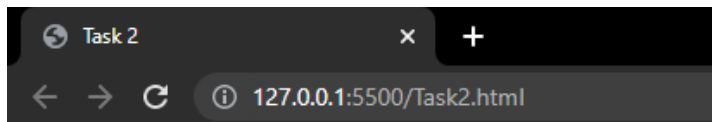
Coding props offer a means of passing data from a parent component to that of a child component. Here, a parent component is identified and passed to component one, pulling from data provided through the code.

## Screenshots:

### Code

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title> Task 2 </title>
5     <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
6   </head>
7   <body>
8     <div id="app">
9       <component1 :person="person"></component1>
10    </div>
11  </body>
12  <script>
13    var component1 = {
14      template: '<div>I am {{ person.firstname }} {{ person.lastname }}</div>',
15      props: ["person"]
16    };
17    var app = new Vue({
18      el: '#app',
19
20      data: {
21        person: {
22          firstname: 'Jane',
23          lastname: 'Doe',
24        },
25      },
26      components: {
27        component1,
28      },
29    });
30  </script>
31 </html>
```

### Output



I am Jane Doe

### GitHub Link:

<https://github.com/219179512/SIT120-Weekly-Tasks/tree/Week-7>

# References

Lepage P, Andrew R (14 May 2020) *Responsive web design basics*, web.dev, accessed 3 August 2022

Vue.js (n.d.) *Introduction*, Vue.js, accessed 12 August 2022

Vue.js (n.d.) *Conditional Rendering*, Vue.js, accessed 12 August 2022

Vue.js (n.d.) *Components Basics*, Vue.js, accessed 19 August 2022

Vue.js (n.d.) *Form Input Bindings*, Vue.js, accessed 19 August 2022