

Report of Appendix 1, Appendix 2 and Appendix 3

Student Name/ Student ID	Trần Hoàng Vũ - GCS220851
Unit Number and Title	Web Development 2 - COMP1842
Academic Year	2024-2025
Project Title	Report of Appendix 1, Appendix 2 and Appendix 3

TABLE OF CONTENTS

Report of Appendix 1, Appendix 2 and Appendix 3.....	1
Task 1.....	3
Task 2.....	4
Task 3.....	5
Task 4.....	6
Appendix 1.....	7
Task 5.....	8
Task 6.....	9
Task 7.....	10
Appendix 2.....	11
Task 8.....	12
Task 9.....	13
Task 10.....	14
Appendix 3.....	15
References.....	16

Task 1

The screenshot shows a code editor interface with two files open: `index.html` and `main.js`. The `index.html` file contains an HTML template for a product page, including a placeholder image and a product info section. The `main.js` file contains a Vue.js application setup with a challenge comment and a description object.

```
index.html
task1 > index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Tran Hoang Vu - GCS220851</title>
8  </head>
9
10 <body>
11   <div id="app">
12
13     <div class="product-image">
14       <img src="" />
15     </div>
16
17     <div class="product-info">
18       <h1>{{ product }}</h1>
19       <!-- CHALLENGE -->
20       <p>{{ description }}</p>
21     </div>
22
23   </div>
24   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
25   <script src="main.js"></script>
26 </body>
27 </html>
```

```
main.js
task1 > main.js > [e] app > data
1 //Add a description to the data object with the value "A pair of warm, fuzzy
2
3 var app = new Vue({
4   el: '#app',
5   data: {
6     product: 'Socks',
7     // CHALLENGE
8     description: 'A pair of warm, fuzzy socks'
9   }
10
11
12
```

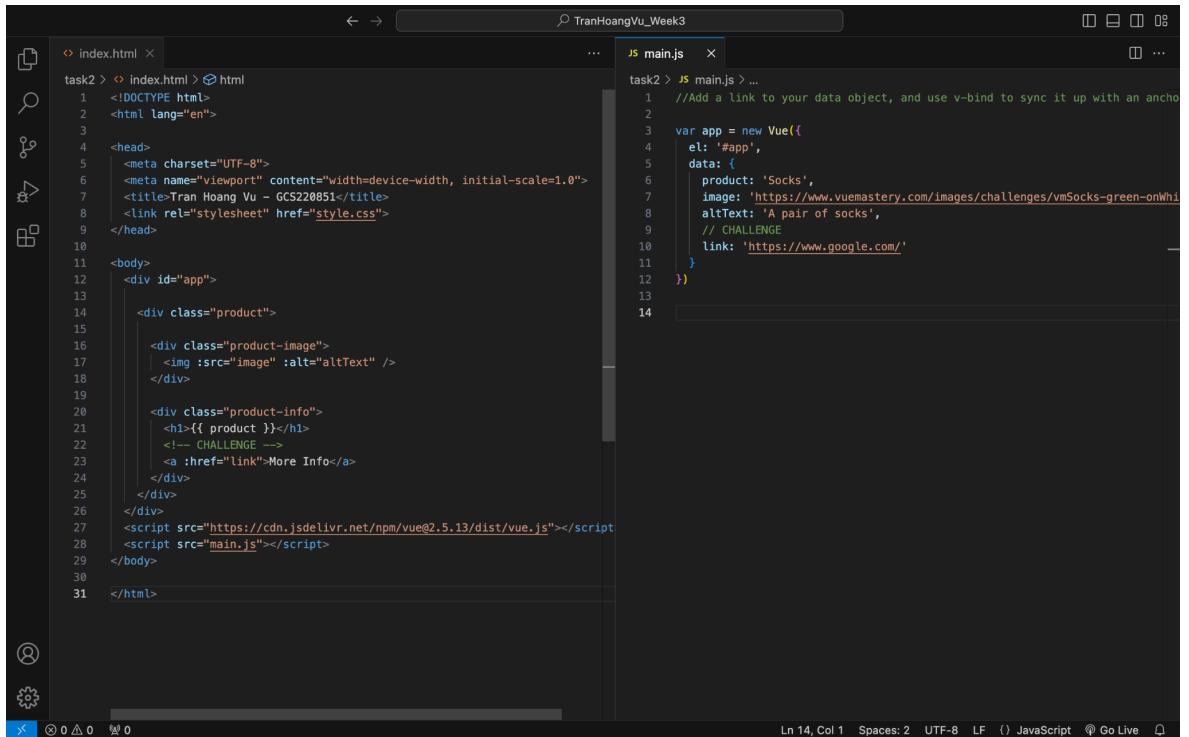
Figure 1: Lesson and Challenge of Task 1

Socks

A pair of warm, fuzzy socks

Figure 2: Result of lesson and challenge of task 1

Task 2



The screenshot shows a code editor with two files open: `index.html` and `main.js`. The `index.html` file contains the structure of a web page with a product card. The `main.js` file contains Vue.js code defining a component for a product card, specifically for socks.

```
index.html
task2 > index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Tran Hoang Vu - GCS220851</title>
8    <link rel="stylesheet" href="style.css">
9  </head>
10 <body>
11   <div id="app">
12
13     <div class="product">
14
15       <div class="product-image">
16         |   
17       </div>
18
19       <div class="product-info">
20         <h1>{{ product }}</h1>
21         <!-- CHALLENGE -->
22         <a :href="link">More Info</a>
23       </div>
24     </div>
25   </div>
26
27   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
28   <script src="main.js"></script>
29 </body>
30
31 </html>
```

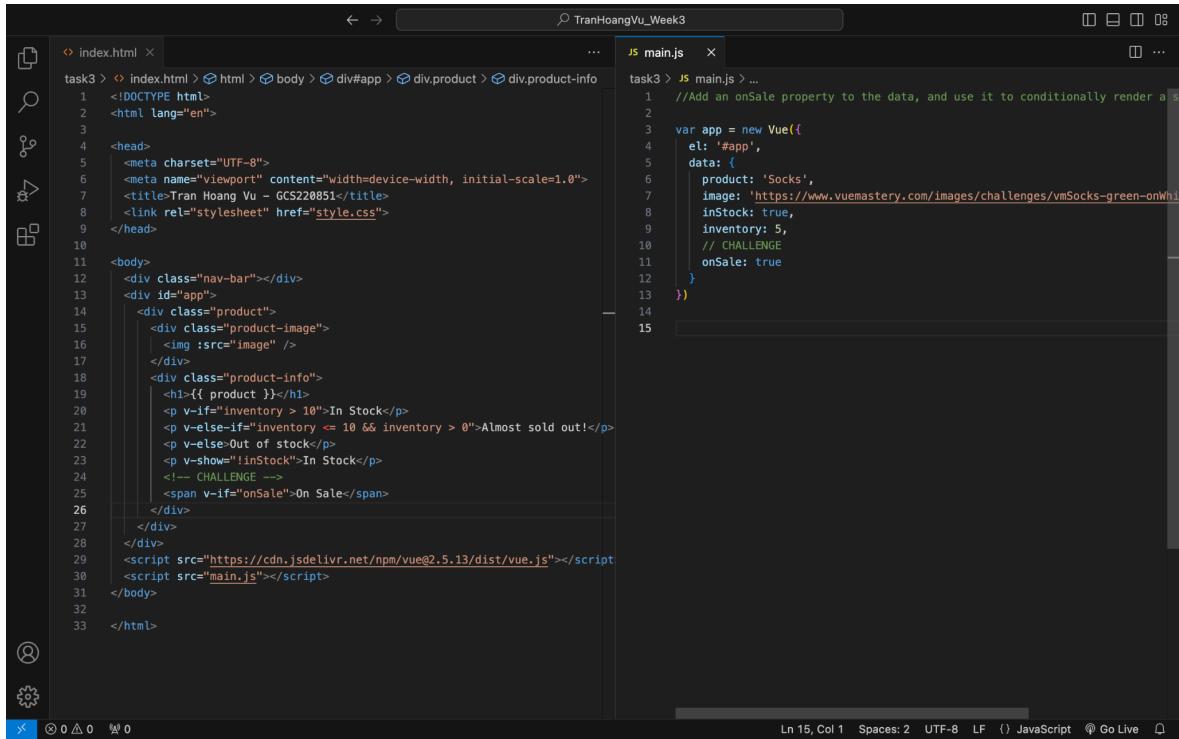
```
main.js
task2 > main.js > ...
1 //Add a link to your data object, and use v-bind to sync it up with an anchor
2
3 var app = new Vue({
4   el: '#app',
5   data: {
6     product: 'Socks',
7     image: 'https://www.vuemastery.com/images/challenges/vmSocks-green-onWhite.png',
8     altText: 'A pair of socks',
9     // CHALLENGE
10    link: 'https://www.google.com/'
11  }
12})
13
```

Figure 3: Lesson and Challenge of Task 2



Figure 4: Result of lesson and challenge of Task 2

Task 3



The screenshot shows a code editor with two files open: `index.html` and `main.js`. The `index.html` file contains the structure of a single product page with a navigation bar, an image, and a detailed product info section. The `main.js` file defines a Vue.js application with a single product object named 'Socks'.

```
task3 > index.html < TranHoangVu_Week3
task3 > main.js < ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Tran Hoang Vu - GCS220851</title>
7    <link rel="stylesheet" href="style.css">
8  </head>
9
10 <body>
11   <div class="nav-bar"></div>
12   <div id="app">
13     <div class="product">
14       <div class="product-image">
15         
16       </div>
17       <div class="product-info">
18         <h1>{{ product }}</h1>
19         <p v-if="inventory > 10">In Stock</p>
20         <p v-else-if="inventory <= 10 && inventory > 0">Almost sold out!</p>
21         <p v-else>Out of stock</p>
22         <p v-show="!inStock">In Stock</p>
23         <!-- CHALLENGE -->
24         <span v-if="onSale">On Sale</span>
25       </div>
26     </div>
27   </div>
28   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
29   <script src="main.js"></script>
30 </body>
31
32
33 </html>
```

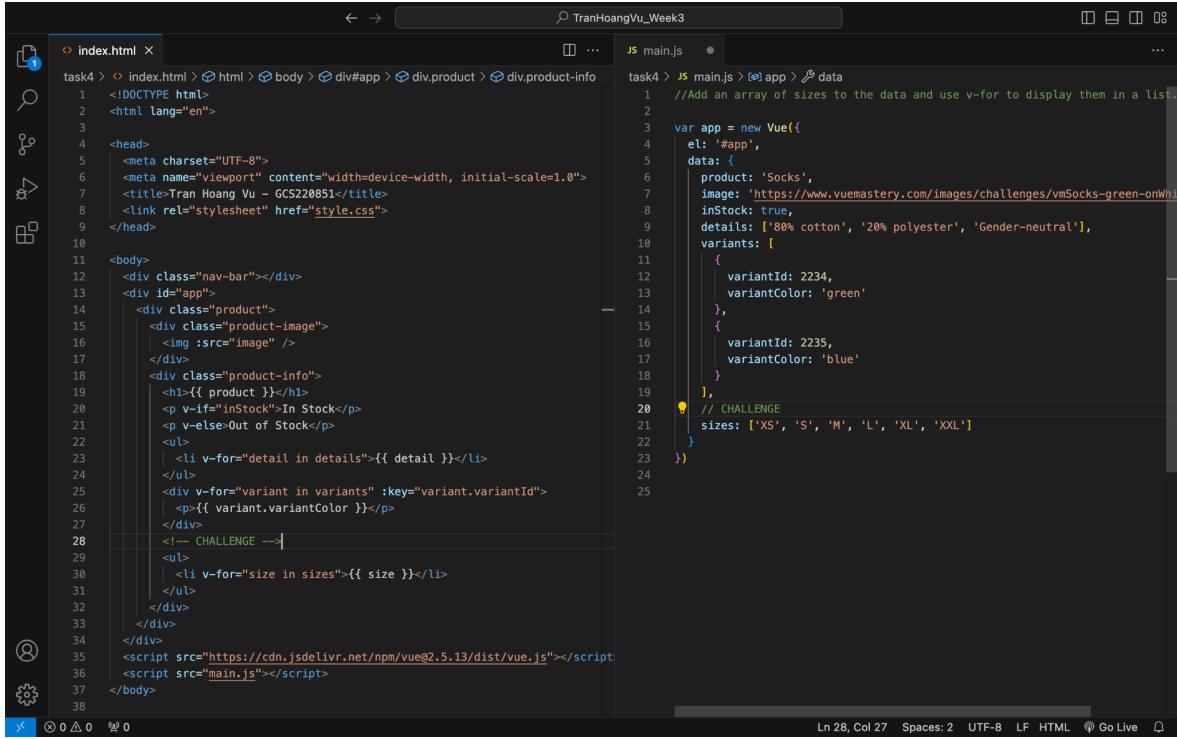
```
task3 > JS main.js > ...
1  //Add an onSale property to the data, and use it to conditionally render a ...
2
3  var app = new Vue({
4    el: '#app',
5    data: {
6      product: 'Socks',
7      image: 'https://www.vuemastery.com/images/challenges/vmSocks-green-onWhi...
8      inStock: true,
9      inventory: 5,
10     // CHALLENGE
11     onSale: true
12   }
13 })
14
15
```

Figure 5: Lesson and Challenge of Task 3



Figure 6: Result of lesson and challenge of Task 3

Task 4



The screenshot shows a code editor with two files open: `index.html` and `main.js`.

`index.html` contains the following code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tran Hoang Vu - GCS220851</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div class="nav-bar"></div>
    <div id="app">
      <div class="product">
        <div class="product-image">
          
        </div>
        <div class="product-info">
          <h1>{{ product }}</h1>
          <p v-if="inStock">In Stock</p>
          <p v-else>Out of Stock</p>
          <ul>
            <li v-for="detail in details">{{ detail }}</li>
          </ul>
          <div v-for="variant in variants" :key="variant.variantId">
            <p>{{ variant.variantColor }}</p>
          </div>
          <!-- CHALLENGE -->
          <ul>
            <li v-for="size in sizes">{{ size }}</li>
          </ul>
        </div>
      </div>
      <script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
      <script src="main.js"></script>
    </body>
```

`main.js` contains the following code:

```
var app = new Vue({
  el: '#app',
  data: {
    product: 'Socks',
    image: 'https://www.vuemastery.com/images/challenges/vmSocks-green-onWhite.png',
    inStock: true,
    details: ['80% cotton', '20% polyester', 'Gender-neutral'],
    variants: [
      {
        variantId: 2234,
        variantColor: 'green'
      },
      {
        variantId: 2235,
        variantColor: 'blue'
      }
    ],
    // CHALLENGE
    sizes: ['XS', 'S', 'M', 'L', 'XL', 'XXL']
})
```

Figure 7: Lesson and Challenge of Task 4



Figure 8: Result of lesson and challenge of Task 4

Appendix 1

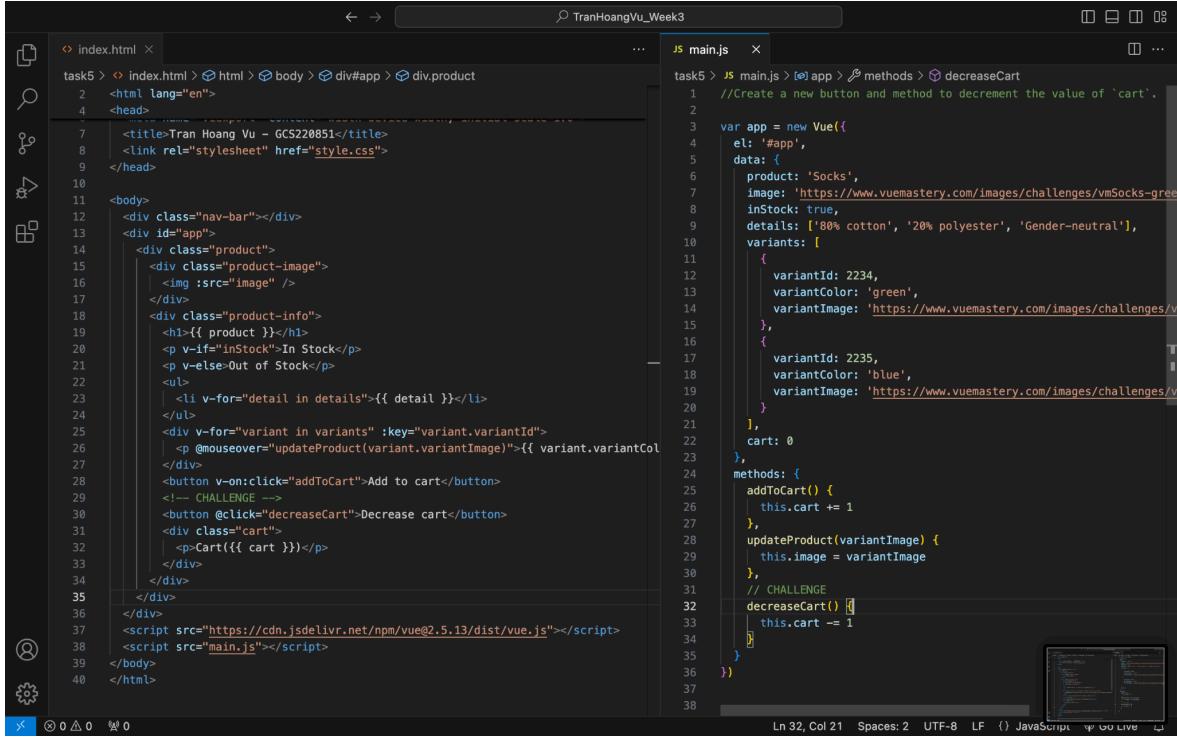
In this lab, I gained a solid understanding of building Vue.js applications, focusing on creating a Vue instance and leveraging its core features. Central to this is Vue's reactivity system, which allows data to automatically update the UI using simple syntax like `{} {}`. This data-binding capability is crucial for creating dynamic and responsive interfaces.

I also learned how to use Vue's directives, such as `v-bind`, which binds data to HTML attributes, enabling seamless interaction between the data model and the view. Additionally, conditional rendering with directives like `v-if` and `v-show` allows for efficient control over element visibility based on the application's state. This flexibility in rendering enhances user experience without unnecessarily altering the DOM structure.

Another key feature I explored was `v-for`, which enables looping through arrays to display data dynamically. Assigning unique keys to elements in these loops ensures Vue efficiently updates only the necessary parts of the interface, improving performance in larger applications.

Looking ahead, these skills will be invaluable for building scalable, reactive, and efficient web applications. The reactivity system will allow me to create user interfaces that respond seamlessly to changes, while Vue's directives will help me optimize performance and manage complex interactions in real time. These features will be essential for developing interactive, data-driven applications that provide a smooth user experience.

Task 5



The screenshot shows a code editor with two tabs: 'index.html' and 'main.js'. The 'index.html' tab displays the structure of an HTML page with a title, a navigation bar, and a product section containing a product image, product info (including a 'variant' section), and a cart summary. The 'main.js' tab shows a Vue.js application setup with a 'data' object defining a 'product' (Socks) and 'variants' (green and blue), and methods for 'addToCart' and 'decreaseCart'.

```
index.html
task5 > index.html > html > body > div#app > div.product
1 <html lang="en">
2   <head>
3     <title>Tran Hoang Vu - GCS220851</title>
4     <link rel="stylesheet" href="style.css">
5   </head>
6
7   <body>
8     <div class="nav-bar"></div>
9     <div id="app">
10       <div class="product">
11         <div class="product-image">
12           
13         </div>
14         <div class="product-info">
15           <h1>{{ product }}</h1>
16           <p v-if="inStock">In Stock</p>
17           <p v-else>Out of Stock</p>
18           <ul>
19             <li v-for="detail in details" :key="detail.variantId">{{ detail }}</li>
20           </ul>
21           <div v-for="variant in variants" :key="variant.variantId">
22             <p @mouseover="updateProduct(variant.variantImage)">{{ variant.variantColor }}</p>
23           </div>
24           <button v-on:click="addToCart">Add to cart</button>
25           <!-- CHALLENGE -->
26           <button @click="decreaseCart">Decrease cart</button>
27         <div class="cart">
28           <p>Cart({{ cart }})</p>
29         </div>
30       </div>
31     </div>
32   </body>
33 </html>
34
35 <script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
36 <script src="main.js"></script>
37
38 </body>
39 </html>
```

```
main.js
task5 > main.js > app > methods > decreaseCart
1 //Create a new button and method to decrement the value of 'cart'.
2
3 var app = new Vue({
4   el: '#app',
5   data: {
6     product: 'Socks',
7     image: 'https://www.vuemastery.com/images/challenges/vmSocks-green.png',
8     inStock: true,
9     details: ['80% cotton', '20% polyester', 'Gender-neutral'],
10    variants: [
11      {
12        variantId: 2234,
13        variantColor: 'green',
14        variantImage: 'https://www.vuemastery.com/images/challenges/vmSocks-green.png'
15      },
16      {
17        variantId: 2235,
18        variantColor: 'blue',
19        variantImage: 'https://www.vuemastery.com/images/challenges/vmSocks-blue.png'
20      }
21    ],
22    cart: 0
23  },
24  methods: {
25    addToCart() {
26      this.cart += 1
27    },
28    updateProduct(variantImage) {
29      this.image = variantImage
30    },
31    // CHALLENGE
32    decreaseCart() {
33      this.cart -= 1
34    }
35  }
})
```

Figure 9: Lesson and Challenge of Task 5

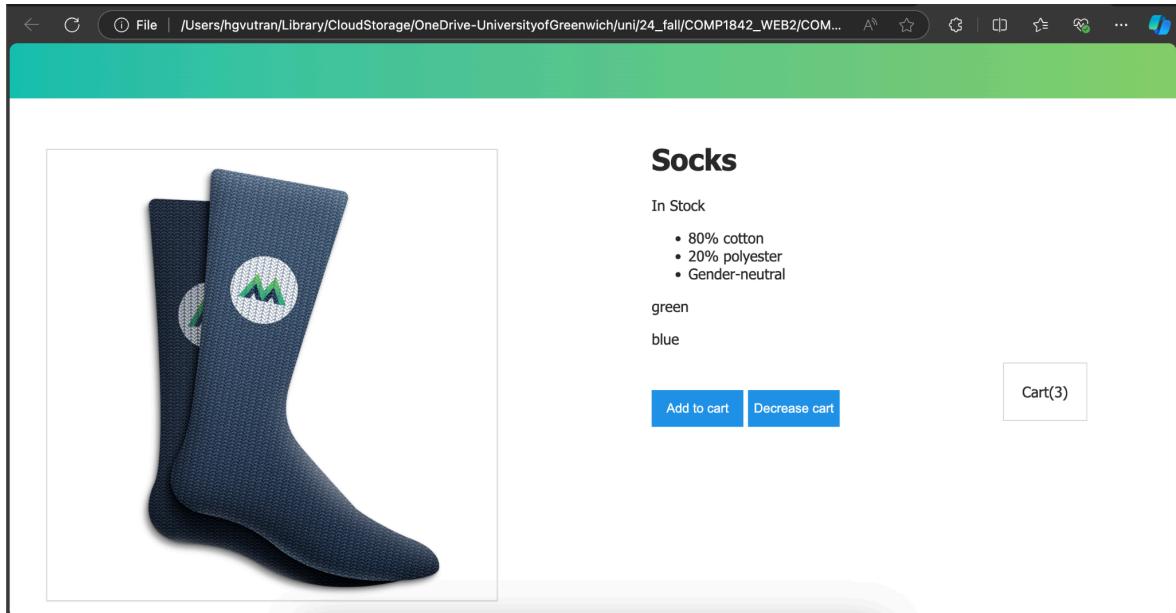
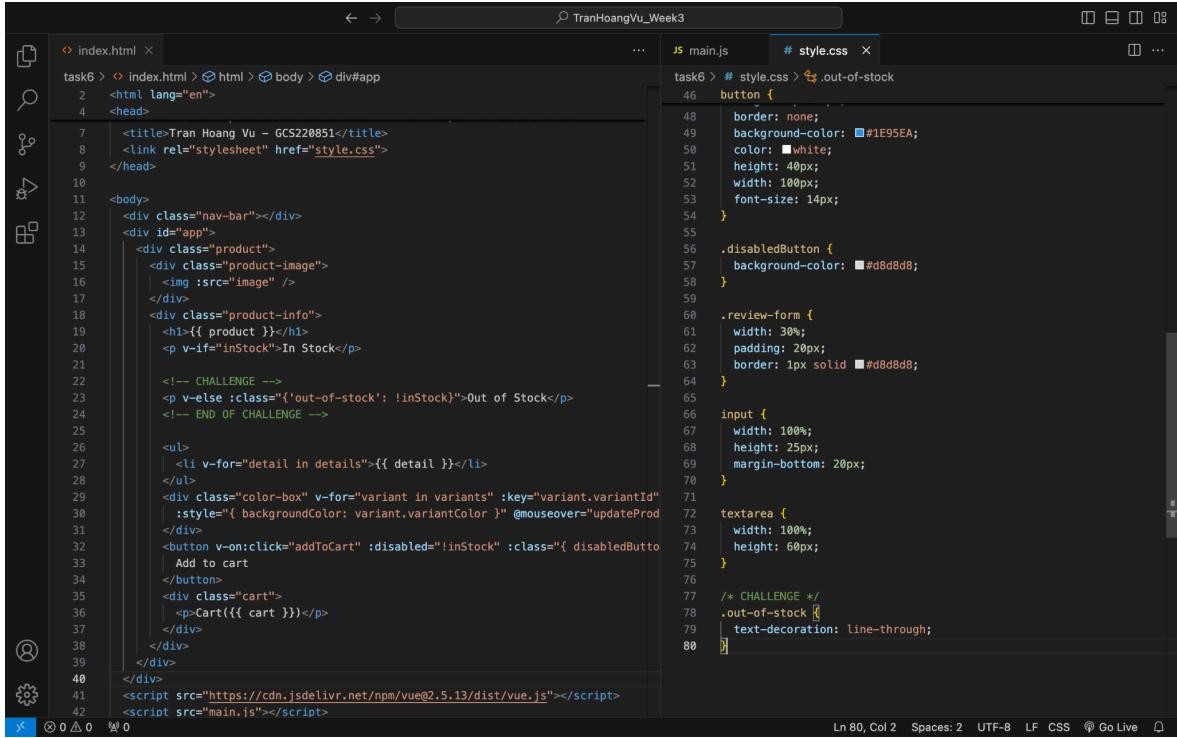


Figure 10: Result of lesson and challenge of Task 5

Task 6



The screenshot shows a code editor with two tabs: 'index.html' and 'style.css'. The 'index.html' tab displays the structure of a Vue.js application. The 'style.css' tab contains CSS rules for a button class and a disabled button class. The CSS includes styles like border none, background-color, color white, height 40px, width 100px, and font-size 14px. The 'disabledButton' class changes the background-color to #d8d8d8. The 'style.css' file also includes a review form section with input and textarea styles, and a challenge section with an out-of-stock class.

```
index.html
task6 > index.html > html > body > div#app
2 <html lang="en">
4 <head>
7   <title>Tran Hoang Vu - GCS220851</title>
8   <link rel="stylesheet" href="#style.css">
9 </head>
10 <body>
11   <div class="nav-bar"></div>
12   <div id="app">
13     <div class="product">
14       <div class="product-image">
15         | 
16       </div>
17       <div class="product-info">
18         | <h1>{{ product }}</h1>
19         | <p v-if="inStock">In Stock</p>
20
21         | <!-- CHALLENGE -->
22         | <p v-else :class="{'out-of-stock': !inStock}">Out of Stock</p>
23         | <!-- END OF CHALLENGE -->
24
25         | <ul>
26           | | <li v-for="detail in details">{{ detail }}</li>
27         </ul>
28       <div class="color-box" v-for="variant in variants" :key="variant.variantId"
29         | | :style="{ backgroundColor: variant.variantColor }" @mouseover="updateProd
30       </div>
31       <button v-on:click="addToCart" :disabled="!inStock" :class="{ disabledButto
32         | Add to cart
33       </button>
34       <div class="cart">
35         | | <p>Cart({{ cart }})</p>
36       </div>
37     </div>
38   </div>
39 </div>
40 </script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
41 <script src="main.js"></script>
```

```
style.css
task6 > # style.css > .out-of-stock
46 button {
47   border: none;
48   background-color: #1E95EA;
49   color: white;
50   height: 40px;
51   width: 100px;
52   font-size: 14px;
53 }
54
55 .disabledButton {
56   background-color: #d8d8d8;
57 }
58
59 .review-form {
60   width: 30%;
61   padding: 20px;
62   border: 1px solid #d8d8d8;
63 }
64
65 input {
66   width: 100%;
67   height: 25px;
68   margin-bottom: 20px;
69 }
70
71 textarea {
72   width: 100%;
73   height: 60px;
74 }
75
76 /* CHALLENGE */
77 .out-of-stock {
78   text-decoration: line-through;
79 }
```

Figure 11: Lesson and Challenge of Task 6

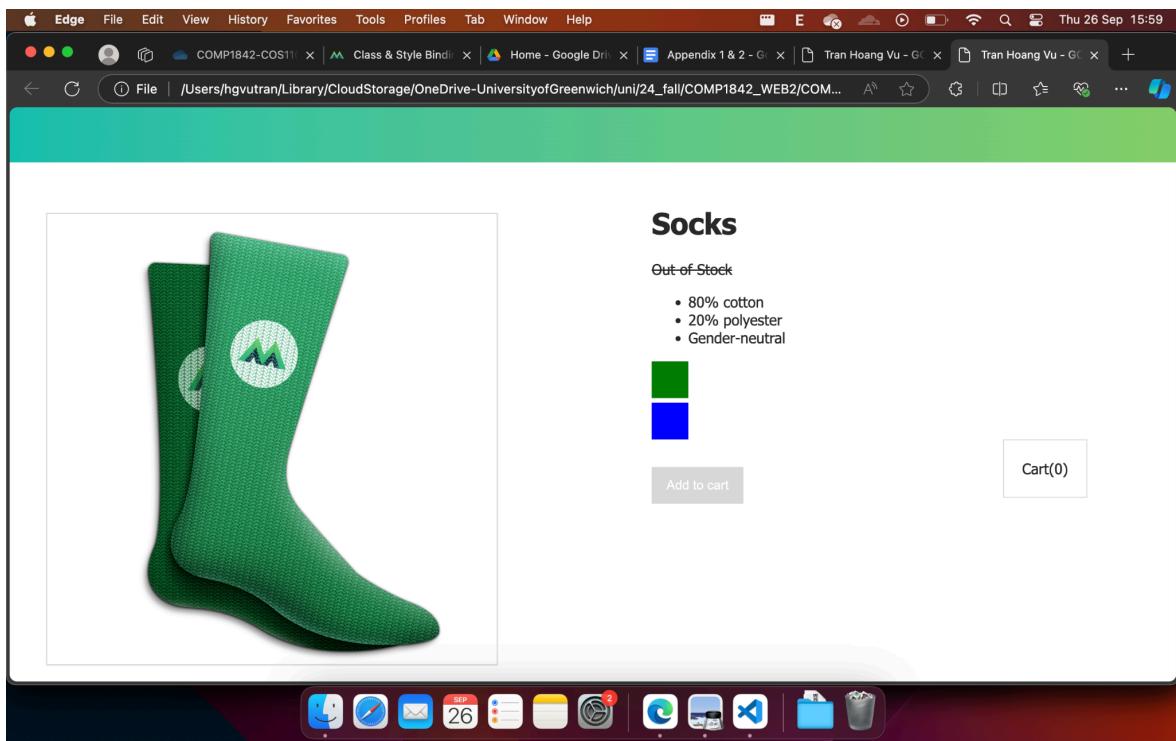


Figure 12: Result of lesson and challenge of Task 6

Task 7

The screenshot shows a code editor with two tabs: 'index.html' and 'main.js'. The 'index.html' tab displays the HTML structure for a product page, including a title, navigation bar, product image, product info (with a challenge section), and a cart summary. The 'main.js' tab shows the corresponding Vue.js script, defining a 'main.js' object with properties like 'app', 'data', 'variants', 'cart', and 'onSale', and methods for 'addToCart' and 'updateProduct'.

```
task7 > index.html < task7 > main.js <
1 <html lang="en">
2   <head>
3     <title>Tran Hoang Vu - GCS220851</title>
4     <link rel="stylesheet" href="style.css">
5   </head>
6   <body>
7     <div class="nav-bar"></div>
8     <div id="app">
9       <div class="product">
10         <div class="product-image">
11           
12         </div>
13         <div class="product-info">
14           <h1>{{ product }}</h1>
15           <p v-if="inStock">In Stock</p>
16           <p v-else>Out of Stock</p>
17
18           <!-- CHALLENGE -->
19           <span v-if="onSale">{{ onSaleComputed }}</span>
20           <!-- END OF CHALLENGE -->
21
22           <ul>
23             <li v-for="detail in details">{{ detail }}</li>
24           </ul>
25           <div class="color-box" v-for="(variant, index) in variants" :key="variant.v
26             <style>{ backgroundColor: variant.variantColor }</style>
27             @mouseover="updateProd
28           </div>
29           <button v-on:click="addToCart" :disabled="!inStock" :class="{ disabledButto
30             Add to cart
31           </button>
32           <div class="cart">
33             <p>Cart({{ cart }})</p>
34           </div>
35         </div>
36       </div>
37     </div>
38   </div>
39   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
40
41
42
43
44
45
46
47
48
49
50
51
52 >
```

Ln 20, Col 39 Spaces: 2 UTF-8 LF HTML ⌂ Go Live ⌂

Figure 13: Lesson and Challenge of Task 7

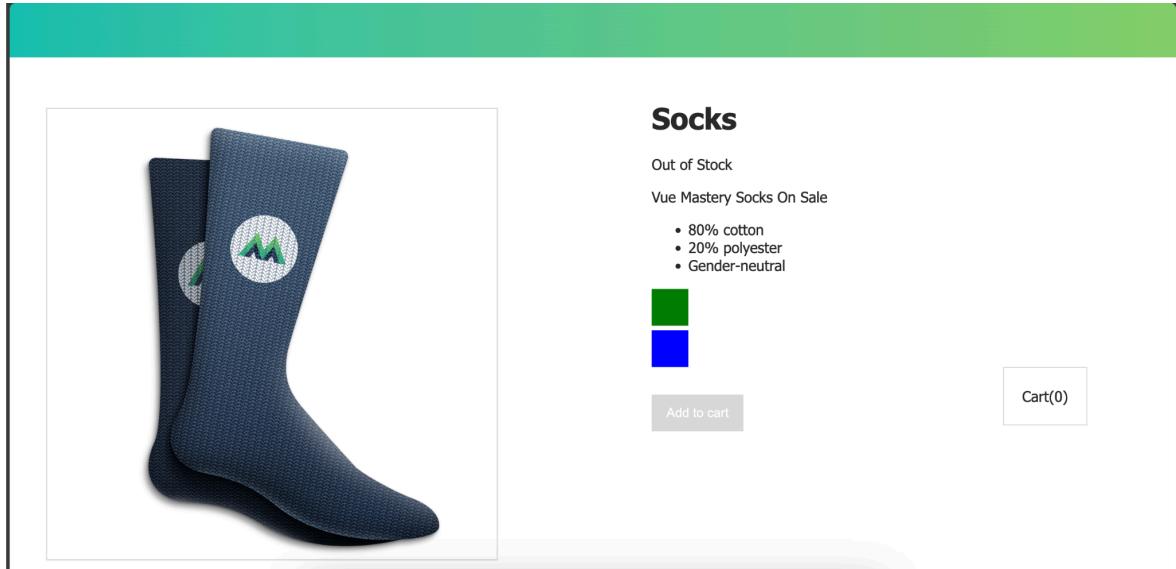


Figure 14: Result of lesson and challenge of Task 7

Appendix 2

In this lab, I delved into key advanced features of Vue.js that improve both interactivity and performance. I focused on event handling, dynamic styling, and using computed properties for efficient data management.

By learning how to use the v-on directive, I gained the ability to efficiently handle user-triggered events like clicks and keypresses, enhancing application responsiveness. Dynamic data binding to style and class attributes allowed me to control the visual appearance of elements based on real-time application state, making the interface more adaptable and user-friendly.

A significant highlight was computed properties, which dynamically calculate values based on dependencies and are only recalculated when necessary. This caching ability is particularly useful for operations that would otherwise be computationally expensive, making Vue.js applications more efficient.

Looking ahead, these skills will be essential for building interactive, scalable applications. Handling events effectively will improve user engagement, while dynamic styling ensures that the interface can adapt to different states seamlessly. Computed properties will be especially valuable in larger applications, where performance optimization is critical for complex data-driven features like filtering large datasets or managing real-time updates. Overall, these techniques will help me create responsive, dynamic, and performance-optimized Vue applications in the future.

Task 8

The screenshot shows the Visual Studio Code interface with two open files:

- index.html**:
A simple HTML file structure with a header containing meta tags and a title, followed by a body section with a navigation bar and a product list. It includes script tags for Vue.js and the main.js file.
- main.js**:
A JavaScript file containing Vue component definitions. The first component, `product-details`, has props for `details` (an array required to be truthy) and `template` (a string). The second component, `product`, also has props for `premium` (a boolean required to be truthy) and `template` (a string).

The status bar at the bottom indicates the file is 23 lines long, the code is 1024 PM on 10/3/2024, and the workspace has 2 spaces, 2 tabs, and is in UTF-8 encoding.

Figure 15: Lesson and Challenge of Task 8

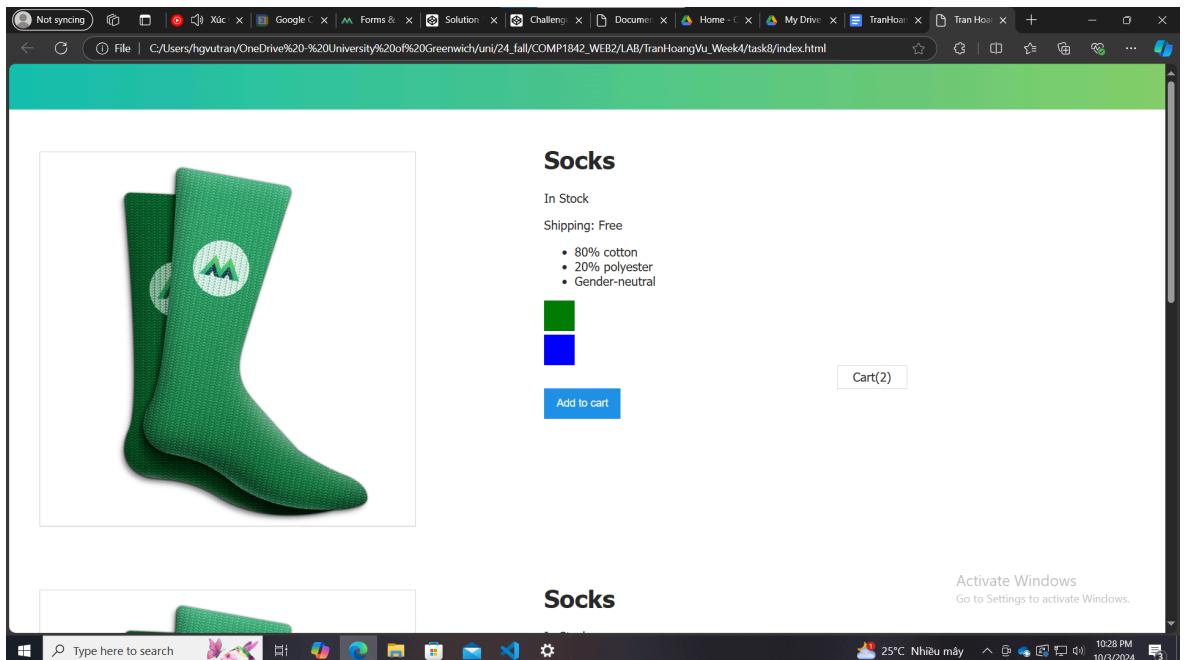


Figure 16: Result of lesson and challenge of Task 8

Task 9

The screenshot shows a code editor with two tabs open: `index.html` and `main.js`. The `index.html` file contains the structure of a web page with a navigation bar, a cart summary, and script tags for Vue.js and main.js. The `main.js` file is a Vue.js component definition with methods for calculating the total price based on the brand and product, handling cart operations like adding and removing items, and displaying shipping information.

```
task9 > index.html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Tran Hoang Vu - GCS20851</title>
7   <link rel="stylesheet" href="style.css">
8 </head>
9 <body>
10 <div class="nav-bar"></div>
11 <div id="app">
12   <div class="cart">
13     <p>Cart({{ cart.length }})</p>
14   </div>
15   <product :premium="premium" @add-to-cart="updateCart" @delete-from-cart="removeCart">
16     <div>
17       <script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
18       <script src="main.js"></script>
19     </div>
20   </product>
21 </div>
22 <script src="https://cdn.jsdelivr.net/npm/vuex@2.5.13/dist/vuex.js"></script>
23 <script src="main.js"></script>
24 </body>
25 </html>
```

```
task9 > main.js
65   methods: {
66     },
67     computed: {
68       title() {
69         return this.brand + ' ' + this.product
70     },
71     image() {
72       return this.variants[this.selectedVariant].variantImage
73     },
74     inStock(){
75       return this.variants[this.selectedVariant].variantQuantity
76     },
77     shipping() {
78       if (this.premium) {
79         return "Free"
80       } else {
81         return 2.99
82       }
83     }
84   }
85   var app = new Vue({
86     el: "#app",
87     data: {
88       premium: true,
89       cart: []
90     },
91     methods: {
92       updateCart(id){
93         this.cart.push(id)
94       },
95       // CHALLENGE
96       removeCart(id){
97         this.cart = this.cart.filter(item => item !== id);
98       }
99     }
100 })
101 
```

Figure 17: Lesson and Challenge of Task 9

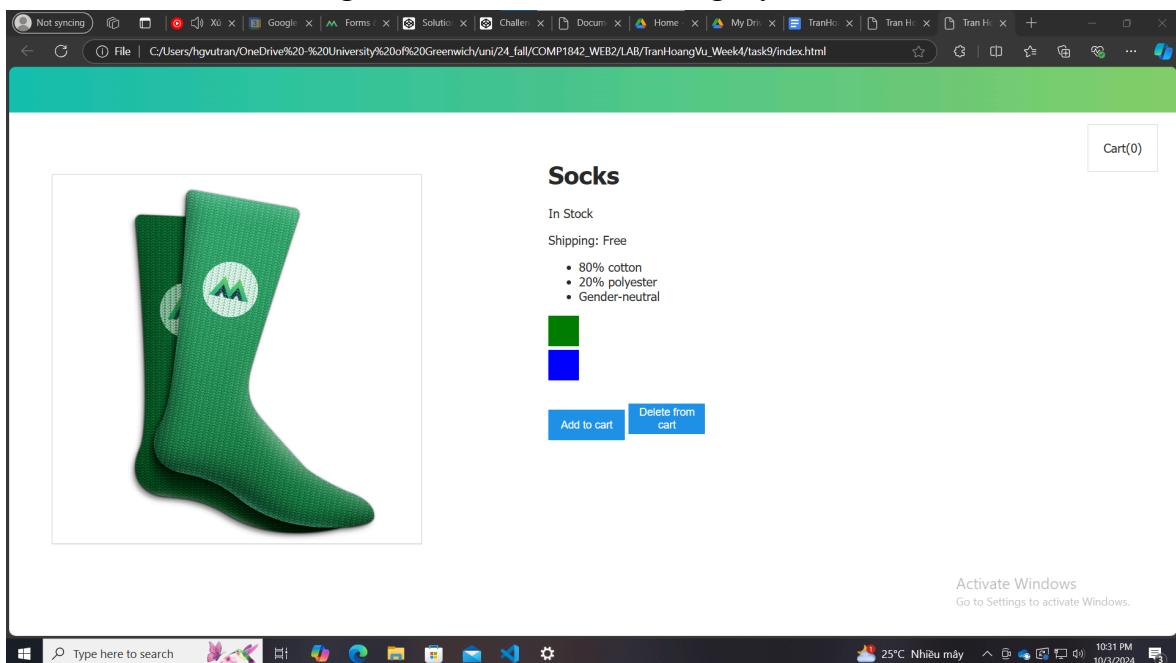


Figure 18: Result of lesson and challenge of Task 9

Task 10

The screenshot shows a code editor with two files open: `index.html` and `main.js`. The `index.html` file contains the structure of a Vue.js application, including a navigation bar, a cart section, and a product detail section. The `main.js` file contains the Vue component definition, which includes properties for name, review, rating, recommend, and errors, and methods for data retrieval and form submission.

```
index.html
task10 > index.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5 <meta charset="UTF-8">
6 <meta name="viewport" content="width=device-width, initial-scale=1.0">
7 <title>Tran Hoang Vu - GCS220851</title>
8 <link rel="stylesheet" href="style.css">
9
10 </head>
11
12 <body>
13 <div class="nav-bar"></div>
14 <div id="app">
15   <div class="cart">
16     <p>Cart({{ cart.length }})</p>
17   </div>
18   <product :premium="premium" @add-to-cart="updateCart" @delete-from-cart="r
19
20   </product>
21
22 </div>
23 <script src="https://cdn.jsdelivr.net/npm/@2.5.13/dist/vue.js"></script>
24 <script src="main.js"></script>
25
26
27 </html>
```

```
main.js
task10 > main.js > template
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
template: -
</p>
<p>Would you recommend this product?</p>
<label>
  Yes
<input type="radio" value="Yes" v-model="recommend"/>
</label>
<label>
  No
<input type="radio" value="No" v-model="recommend"/>
</label>
<p> <input type="submit" value="Submit">
</p>
<p v-if="errors.length">
  Please correct the following error(s):
<ul>
  <li v-for="error in errors">{{ error }}</li>
</ul>
</p>
</form>
<script>
  data() {
    return {
      name: null,
      review: null,
      rating: null,
      recommend: null,
      errors: []
    }
  },
  methods: {
    ...
  }
</script>
```

Figure 19: Lesson and Challenge of Task 10

The screenshot shows a web browser displaying a Vue.js application. The page title is "Reviews". It features a form for leaving a review, including fields for Name, Review, Rating (set to 4), and a "Would you recommend this product?" section with radio buttons for Yes and No. Below the form is a message about required fields: "Please correct the following error(s)" with a list: "Review required." and "Recommend required.".

Reviews

Name: han

Review:

Rating: 4

Would you recommend this product?

Yes

No

Submit

Please correct the following error(s):

- Review required.
- Recommend required.

Figure 20: Result of lesson and challenge of Task 10

Appendix 3

Through Appendix 3, I learned how to create components, which makes it easier to reuse code. In my view, this is similar to creating functions in other programming languages. This approach has been extremely helpful in managing code and fixing bugs. Additionally, I discovered how to pass data from parent tags to child tags using props, as well as how to send data back from child components to parent components. This concept of data flow between components has been particularly useful in improving my understanding of React's architecture.

Moreover, I learned how to create forms and handle custom validation, which is crucial in ensuring that the user input meets specific criteria. Understanding how to set up and manage forms efficiently can significantly reduce the likelihood of errors or invalid data submissions.

Looking forward, I can see how these skills will help me in larger projects. For instance, component reusability will streamline the development of scalable applications by reducing redundant code. Additionally, mastering data flow between components will allow me to build more interactive and dynamic user interfaces, where parent and child components can seamlessly communicate. The ability to handle forms with custom validation could be vital for building applications that require user authentication, payment processing, or any other data-sensitive operations, ensuring a smooth user experience.

Appendix 4

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with 'Open connections' (tv6950k) and 'My resources' sections. The main area is titled 'Quickstart > welsh_pubs > welsh_pubs'. It has a query builder at the top with fields for 'Run', 'Load query', 'Save query', 'Query history', 'Set default query', 'Copy', 'Paste', 'AI Helper', and 'Visual Query Builder'. Below the query builder is a table with columns: _id, fsa_id, name, address, location, local_authority, and contact. The table contains 3,122 documents, each with a unique ID and details about a pub in Wales. The table has a 'Table View' button at the bottom right.

Figure: Lesson 4 Exercise 1

This screenshot shows the same MongoDB Compass interface as the previous one, but with a different query. The query in the builder is: { "name": "/snooker/", "local_authority": { \$ne: "Caerphilly" } }. The projection is set to { "fsa_id": 1, "name": 1, "address.city": 1, "local_authority": 1 }. The resulting table shows 14 documents, all of which have a 'local_authority' field value of 'Caerphilly'. The table has a 'Table View' button at the bottom right.

Figure: Lesson 4 Exercise 3

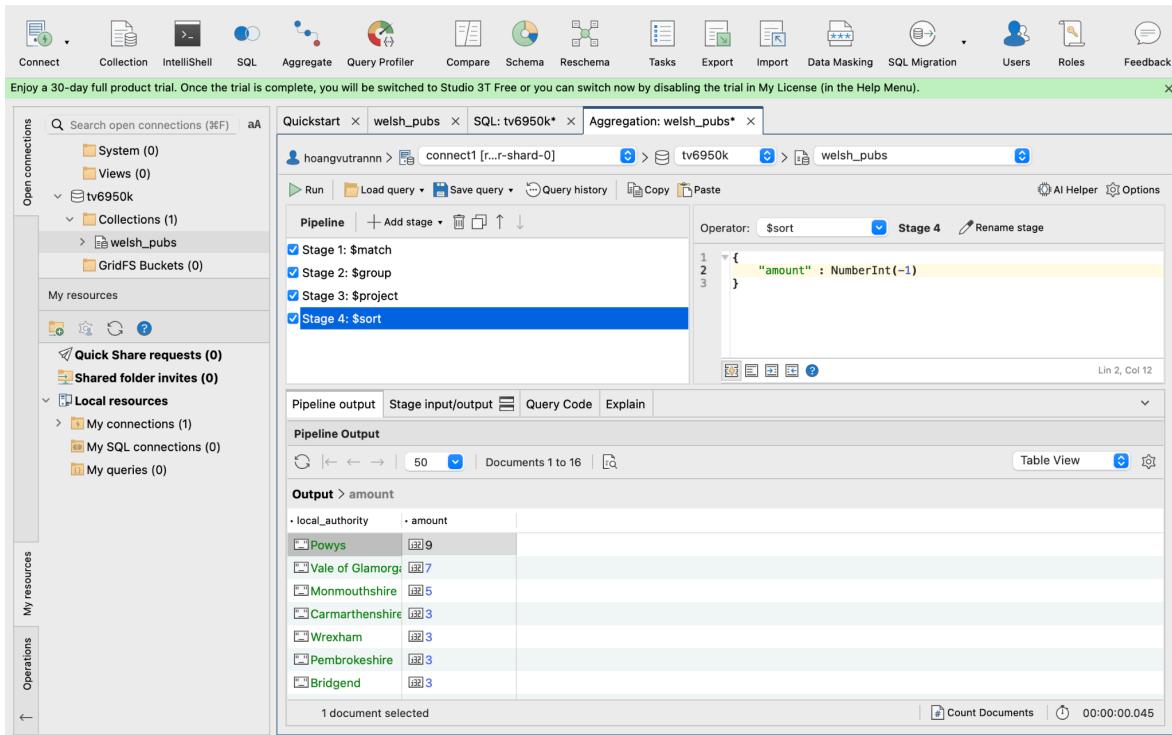


Figure: lesson 5 exercise 3

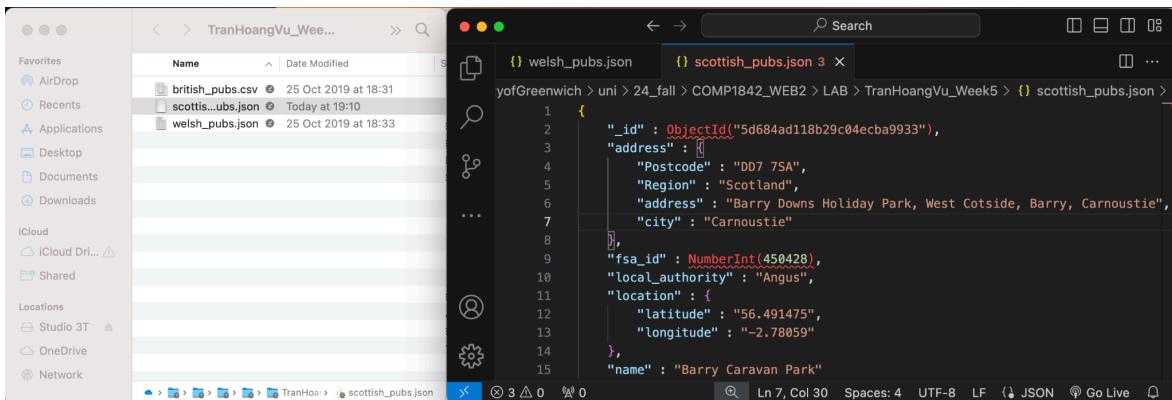


Figure: lesson 6 exercise 2

Understanding NoSQL and MongoDB

NoSQL, which stands for "Not Only SQL," refers to a category of database management systems that provide flexible schema design and scalable performance, making them suitable for handling large volumes of unstructured or semi-structured data (McCool, 2019). Unlike traditional relational databases, which use fixed schemas and SQL for data manipulation, NoSQL databases offer various models such as document, key-value, column-family, and graph databases (Chodorow, 2013). This flexibility enables organizations to adapt to changing data requirements and support high-velocity data ingestion.

MongoDB is a leading NoSQL database that utilizes a document-oriented data model (MongoDB Official Documentation, n.d.). In MongoDB, data is stored in JSON-like

format called BSON (Binary JSON), allowing for easy representation of complex data structures. Each document in a collection can have a different structure, accommodating diverse data types and providing the flexibility to evolve the data model over time. This makes MongoDB particularly well-suited for applications requiring quick iterations, such as content management systems and real-time analytics.

One of the primary advantages of MongoDB is its scalability. It supports sharding, which allows data to be distributed across multiple servers, ensuring high availability and improved performance (Chodorow, 2013). Additionally, its rich query capabilities and built-in support for indexing enhance data retrieval efficiency.

In summary, NoSQL databases, particularly MongoDB, empower organizations to manage vast amounts of data with agility and efficiency, making them essential for modern application development and data-driven decision-making (McCool, 2019).

References

Jahr, A. (2024). *Intro to Vue 2 - Intro to Vue 2 | Vue Mastery*. [online] Vue Mastery. Available at: <https://www.vuemastery.com/courses/intro-to-vue-js> [Accessed 26 Sep. 2024].

MongoDB: The Definitive Guide by Kristina Chodorow (2013).

R. McCool, "NoSQL Databases: A Survey and a Future Research Direction," Journal of Computer Science, vol. 10, no. 2, pp. 1-15, 2019.

MongoDB Official Documentation. (n.d.). Retrieved from <https://www.mongodb.com/docs> [Accessed 10 Oct. 2024].