

**Comp1842- Web Programming 2**  
**Vuong Quoc Anh-GCS220150**  
**Class: Cos1103**  
**Submission: Week 4**

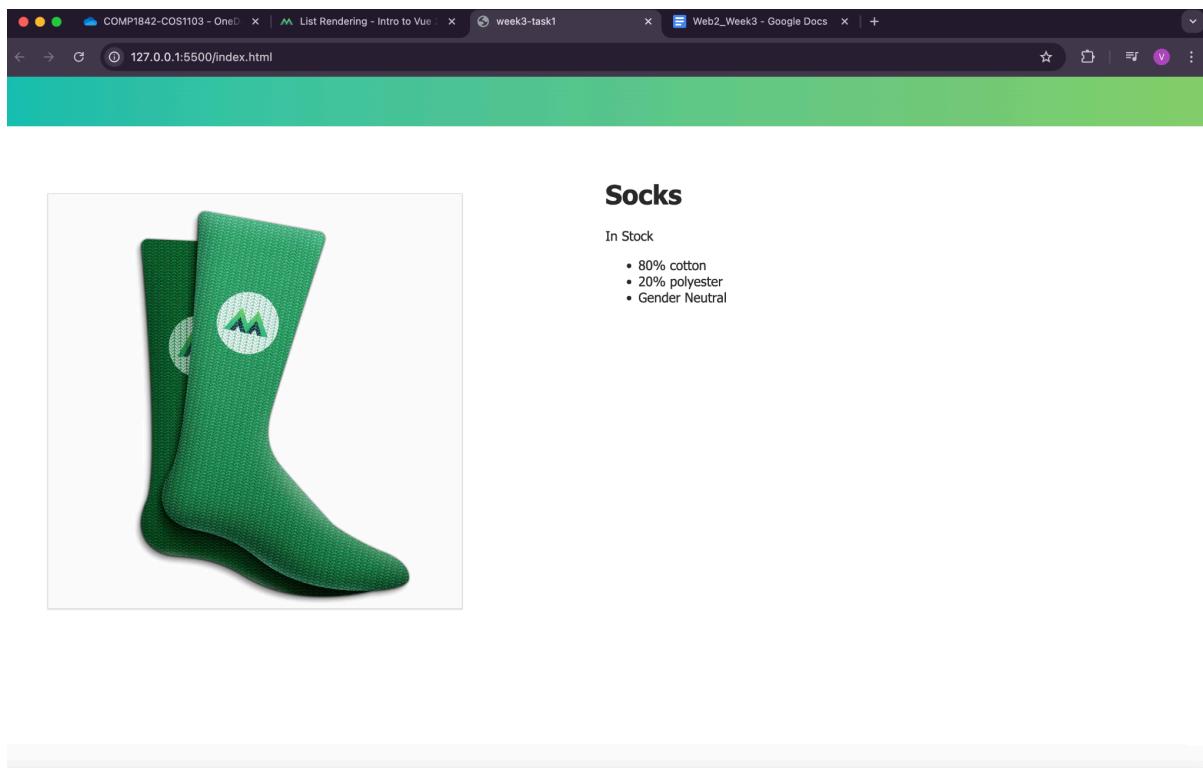
**Table of Contents:**

<b>Appendix1:</b> .....	<b>1</b>
1.1Task:.....	1
1.2Challenges:.....	3
<b>Appendix2:</b> .....	<b>6</b>
2.1Task:.....	6
2.2Challenges:.....	9
<b>Appendix3:</b> .....	<b>11</b>
3.1Task:.....	11
3.2Challenges:.....	14

**Appendix1:**

**1.1 Task:**

In this section, we learn how to create Vue instances and load data onto a website. Vue instances are the core of every Vue app, connecting elements to the DOM and making Vue reactive. We use v-bind: (or : for short) to bind data to HTML attributes. Vue has directives like v-if, v-else-if, v-else and v-show for conditional rendering. If the condition is true, the element shows v-show only toggles visibility without altering the DOM. The v-for directive lets us loop through arrays to display data, and it's best to use a unique key for each element.



**Fig 1:Browser**

```

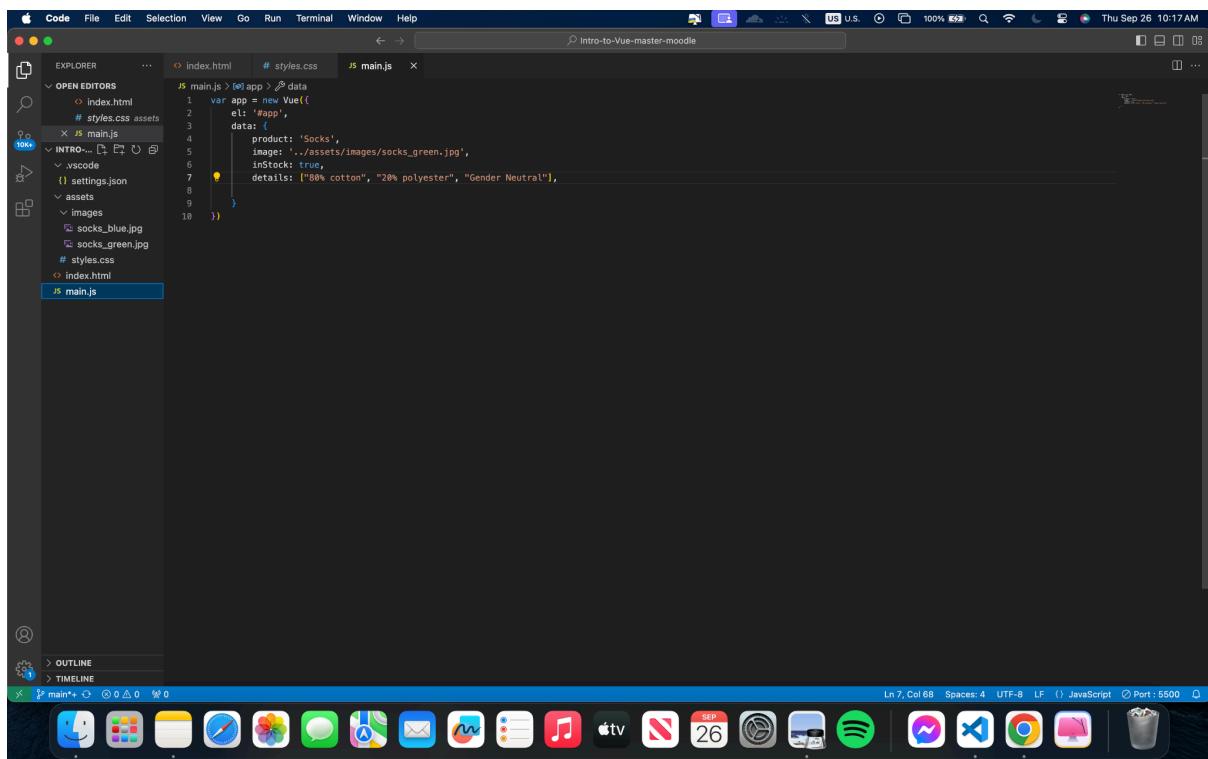
Code File Edit Selection View Go Run Terminal Window Help
Thu Sep 26 10:16 AM

EXPLORER ... OPEN EDITORS index.html # styles.css JS main.js
INTRO... index.html # styles.css assets
main.js
vscode settings.json
assets
images
socks_blue.jpg
socks_green.jpg
styles.css
index.html
main.js

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>week3-task1</title>
    <!-- Import Styles -->
    <link rel="stylesheet" href="assets/styles.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>
        <!-- Import Vue.js -->
        <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.js"></script>
        <!-- Import Js -->
        <script src="main.js"></script>
      </div>
    </div>
  </body>
</html>

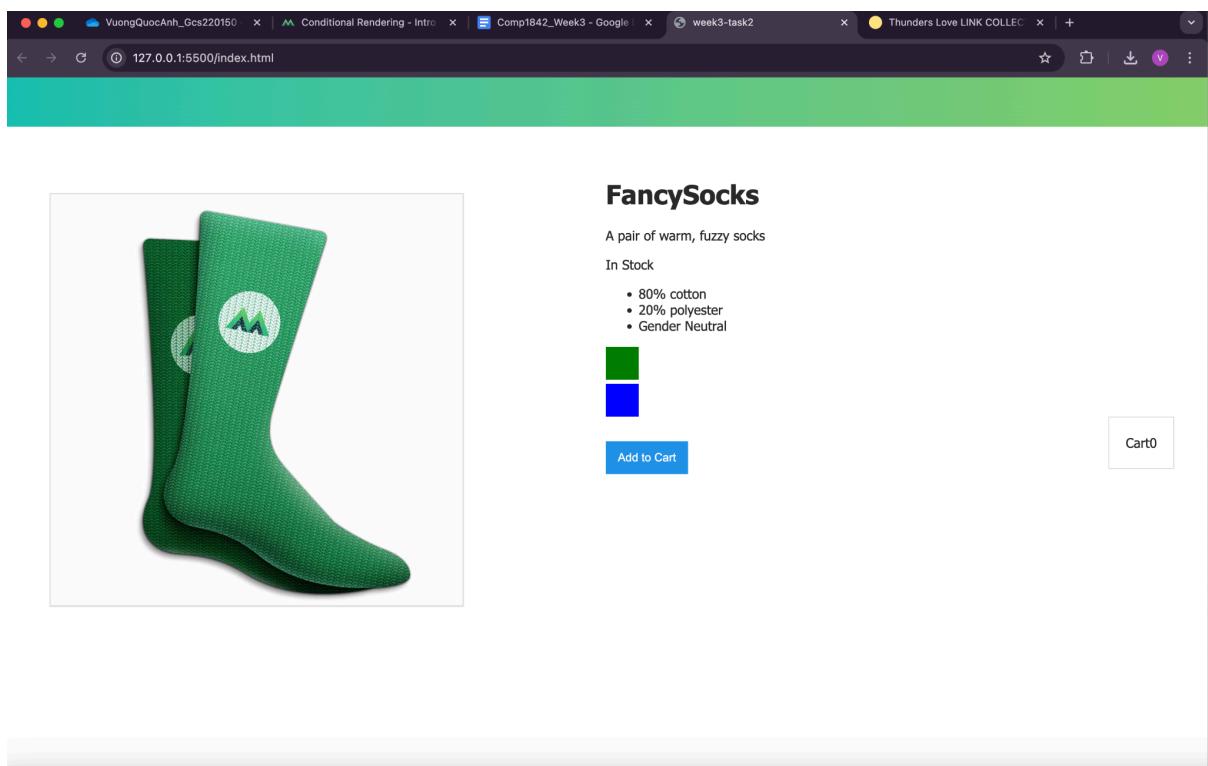
```

**Fig 2:HTML**



**Fig 3: JS**

## 1.2 Challenges:



**Fig4: Adding description**

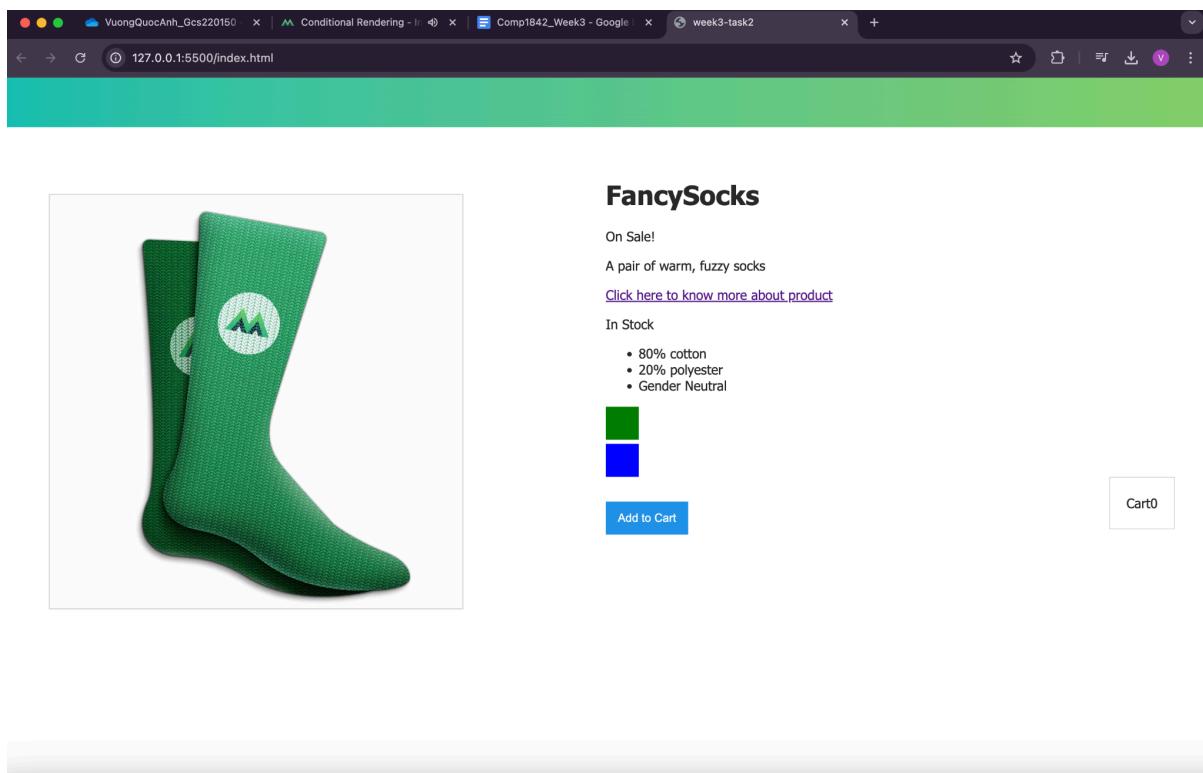


Fig5:Adding On Sale

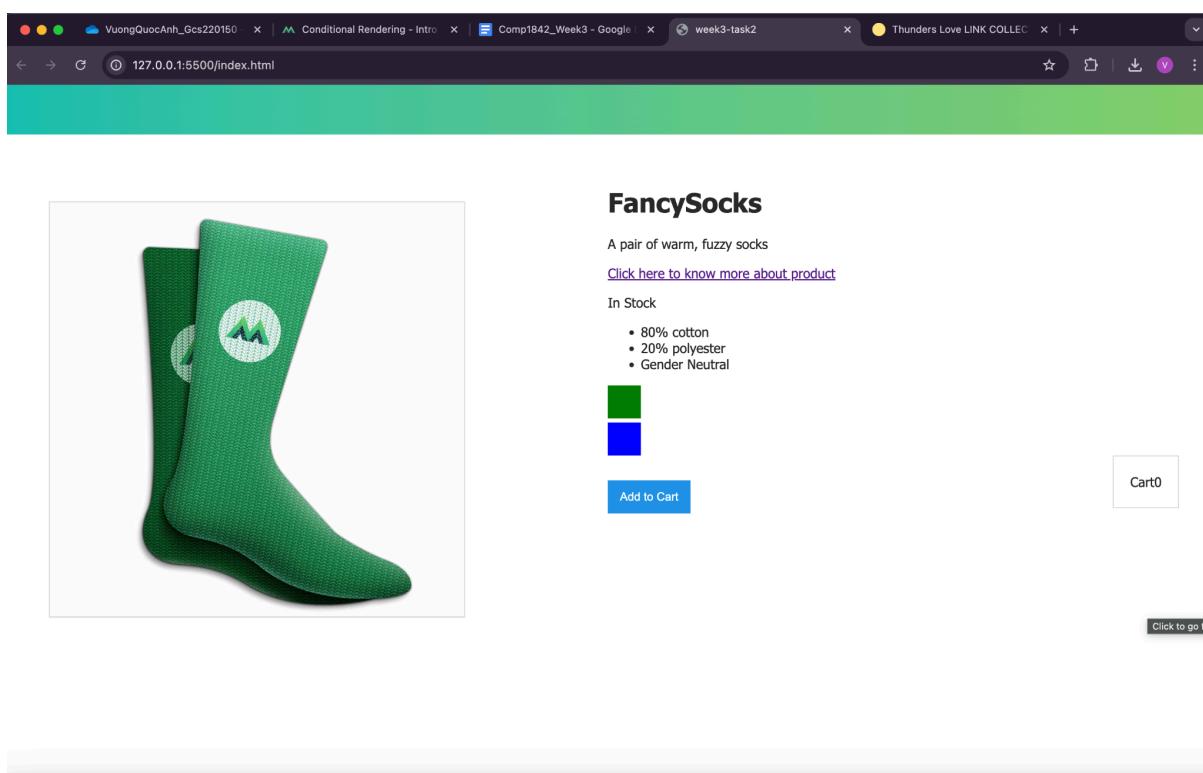
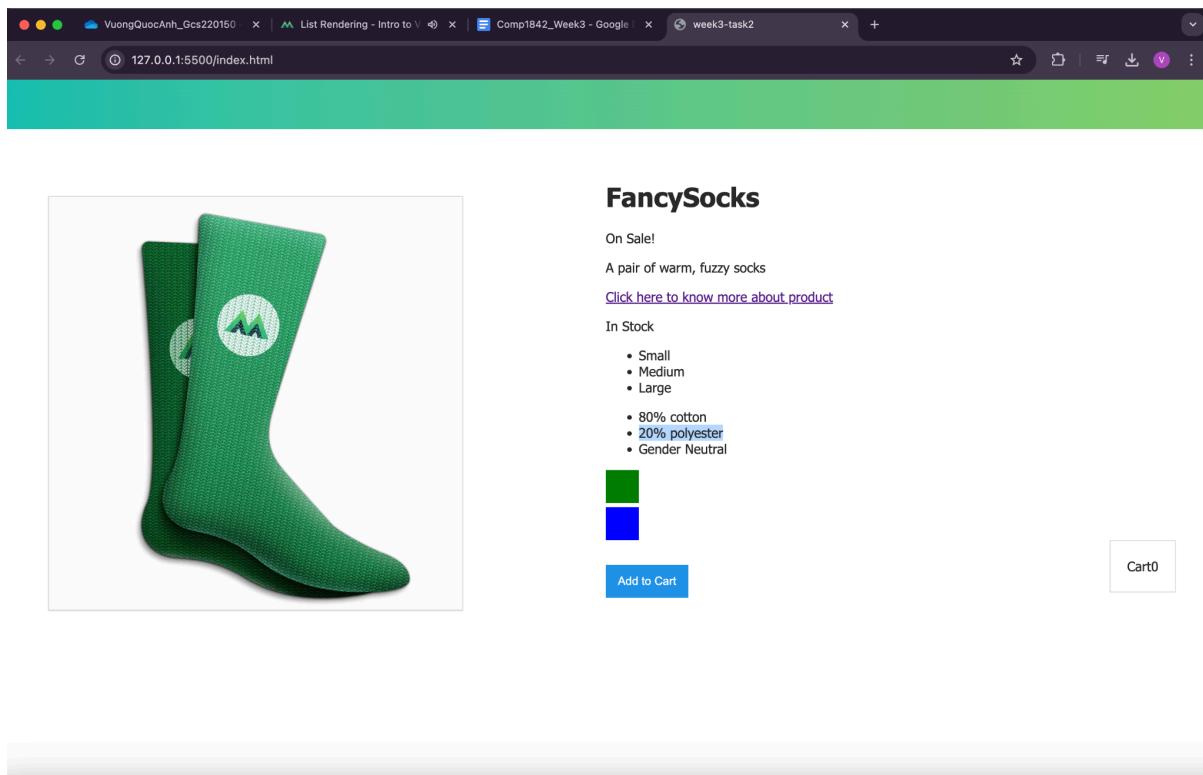


Fig6:Adding link



**Fig7:Adding Size**

```

<!-->
<div class="product-info">
  <h1>{{ title }}</h1>
  <span v-if="onSale">On Sale!</span>
  <p>{{ description }}</p>
  <a v-bind:href="link" target="_blank">Click here to know more about product</a>
  <p v-if="inStock">In Stock</p>
  <p v-else>Out of Stock</p>
  <ul>
    <li v-for="size in sizes">{{ size }}</li>
  </ul>
  <ul>
    <li v-for="detail in details">{{ detail }}</li>
  </ul>
  <div v-for="(variant, index) in variants"
        :key="variant.variantId"
        class="color-box"
        :style="{ backgroundColor: variant.variantColor}"
        @mouseover="updateProduct(index)">
    </div>
  <button v-on:click="addToCart"
          :disabled="!inStock"
          :class="{ disabledButton: !inStock }">Add to Cart</button>
</div>

```

**Fig8: HTML after doing challenges**

```

var app = new Vue({
  el: '#app',
  data: {
    brand: 'Fancy',
    description: 'A pair of warm, fuzzy socks',
    link: 'https://shop.kitchener.ca/thunders-love-link-collection-green-socks.html',
    product: 'Socks',
    selectedVariant: 0,
    onSelect: true,
    details: ["80% cotton", "20% polyester", "Gender Neutral"],
    sizes: ['Small', 'Medium', 'Large'],
    variants: [
      {
        variantId: 2234,
        variantColor: "green",
        variantImage: '../assets/images/socks_green.jpg',
        variantQuantity: 5
      },
      {
        variantId: 2235,
        variantColor: "blue",
        variantImage: '../assets/images/socks_blue.jpg',
        variantQuantity: 0
      }
    ],
    cart: 0
  },
  methods: {
    addToCart() {
      this.cart += 1
    },
    updateProduct(index) {
      this.selectedVariant = index
      console.log(index)
    }
  },
  computed: {
    title() {
      return this.brand + ' ' + this.product
    },
    image() {
      return this.variants[this.selectedVariant].variantImage
    },
    inStock() {
      return this.variants[this.selectedVariant].variantQuantity
    }
  }
})

```

**Fig9:JS after doing challenges**

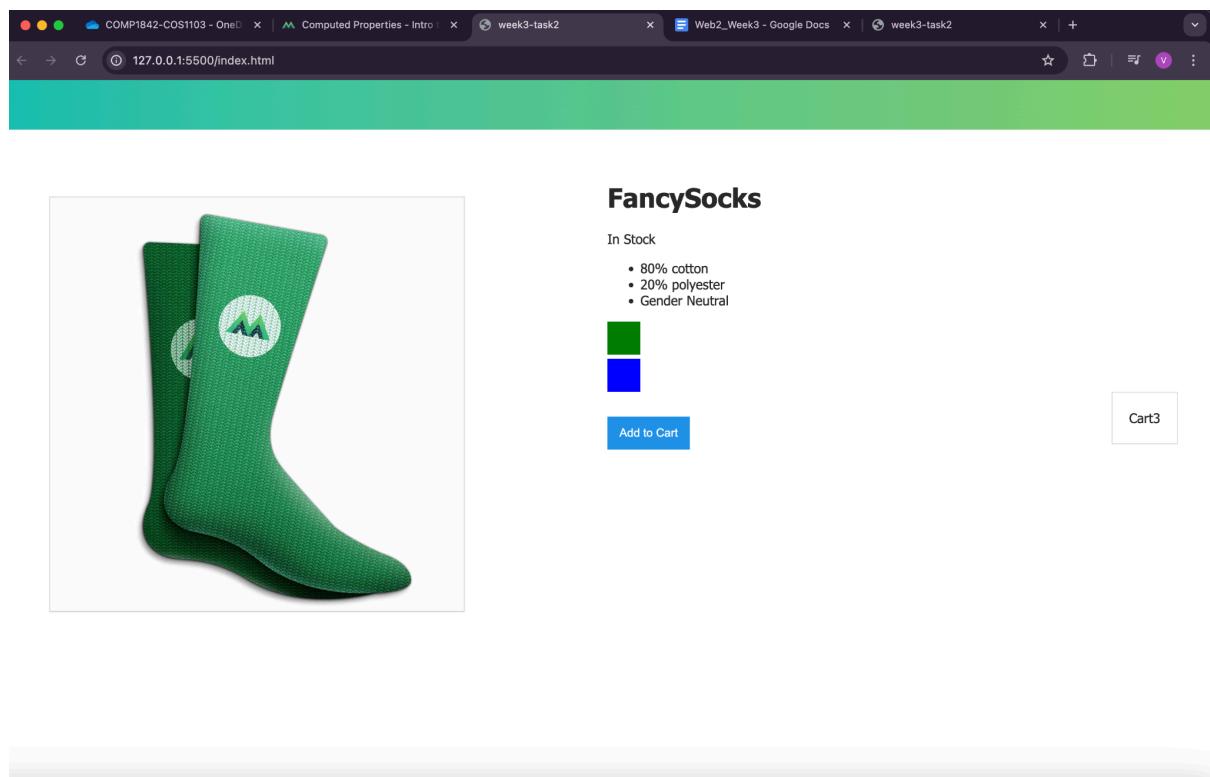
## Appendix2:

### 2.1 Task:

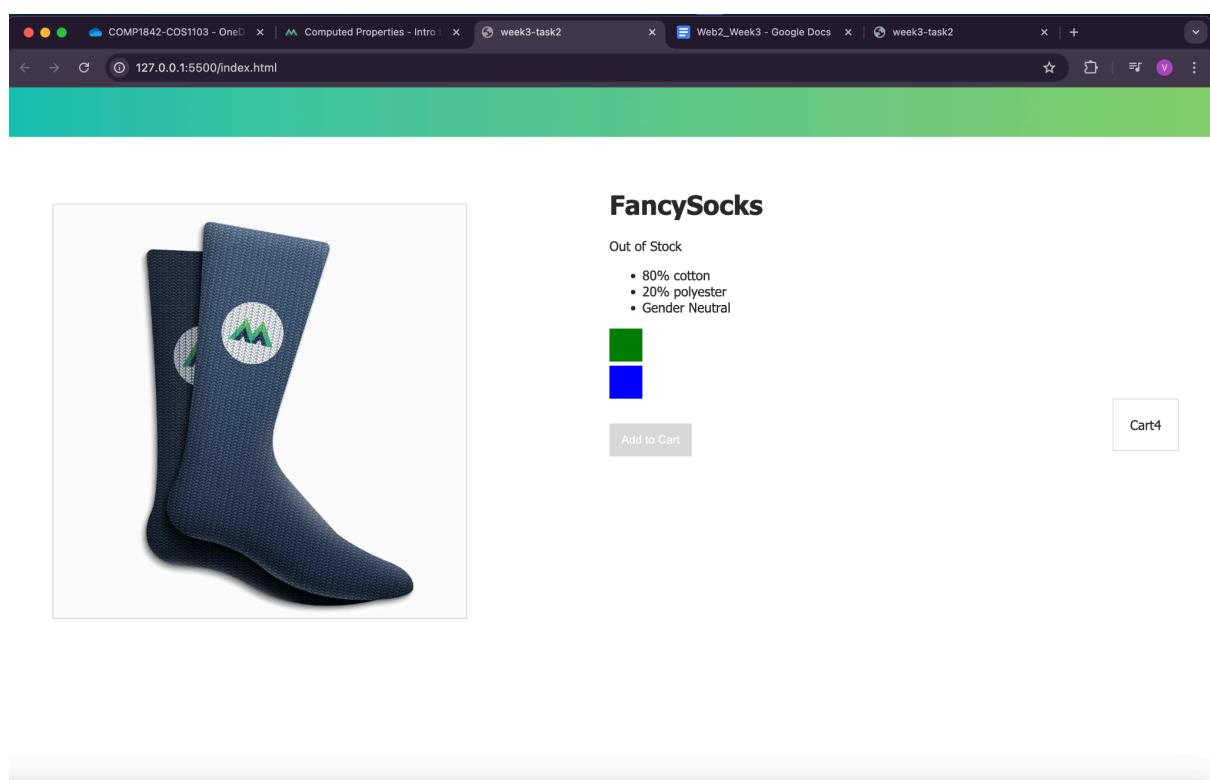
In this section, we use v-on to allow elements to listen for events. We know that the shorthand for v-on is @. V-on can directive trigger a method.

Data can be bound to an element's style and class. We can use expressions inside an element's class binding to evaluate whether a class should appear or not.

Computed properties calculate a value rather than store a value and use data from the application to calculate its values.



**Fig10:Browser 1**



**Fig11:Browser2**

The screenshot shows the VS Code interface on a Mac OS X desktop. The title bar reads "Intro-to-Vue-master-moodle". The Explorer sidebar shows files like index.html, main.js, and styles.css. The main editor area displays the HTML code for a product page. The code includes a navigation bar, a product container with an image, and a detailed info section with variants and a cart button. The status bar at the bottom shows "Ln 38, Col 51" and "Port: 5500".

```
<html lang="en">
  <head>
    <body>
      <div class="nav-bar"></div>
      <div id="app">
        <div class="product">
          <div class="product-image">
            
          </div>
          <div class="product-info">
            <h1>{{ title }}</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, index" in variants">
              <key>variant.variantId</key>
              <div class="color-box" :style="{ backgroundColor: variant.variantColor }"
                @mouseover="updateProduct(index)">
                </div>
              <button v-on:click="addToCart"
                :disabled="!inStock"
                :class="{'disabledButton': !inStock}">Add to Cart</button>
            </div>
            <div class="cart">
              <p>Cart {{ cart }}</p>
            </div>
          </div>
        </div>
      </div>
    </body>
  </html>
  <!-- Import Vue.js -->
```

Fig12:HTML

The screenshot shows the VS Code interface on a Mac OS X desktop. The title bar reads "Intro-to-Vue-master-moodle". The Explorer sidebar shows files like index.html, main.js, and styles.css. The main editor area displays the JavaScript code for the application. It defines a Vue instance with a computed property for the app object, methods for adding to cart and updating product, and computed properties for title, image, and in stock status. The status bar at the bottom shows "Ln 43, Col 6" and "Port: 5500".

```
var app = new Vue({
  el: '#app',
  data: {
    brand: 'Fancy',
    product: 'Socks',
    selectedVariant: 0,
    details: ["80% cotton", "20% polyester", "Gender Neutral"],
    variants: [
      {
        variantId: 224,
        variantColor: "green",
        variantImage: '../assets/images/socks_green.jpg',
        variantQuantity: 5
      },
      {
        variantId: 225,
        variantColor: "blue",
        variantImage: '../assets/images/socks_blue.jpg',
        variantQuantity: 0
      }
    ],
    cart: 0
  },
  methods: {
    addToCart() {
      this.cart += 1
    },
    updateProduct(index) {
      this.selectedVariant = index
      console.log(index)
    }
  },
  computed: {
    title() {
      return this.brand + ' ' + this.product
    },
    image() {
      return this.variants[this.selectedVariant].variantImage
    },
    inStock() {
      return this.variants[this.selectedVariant].variantQuantity
    }
  }
})
```

Fig13:JS

## 2.2 Challenges:

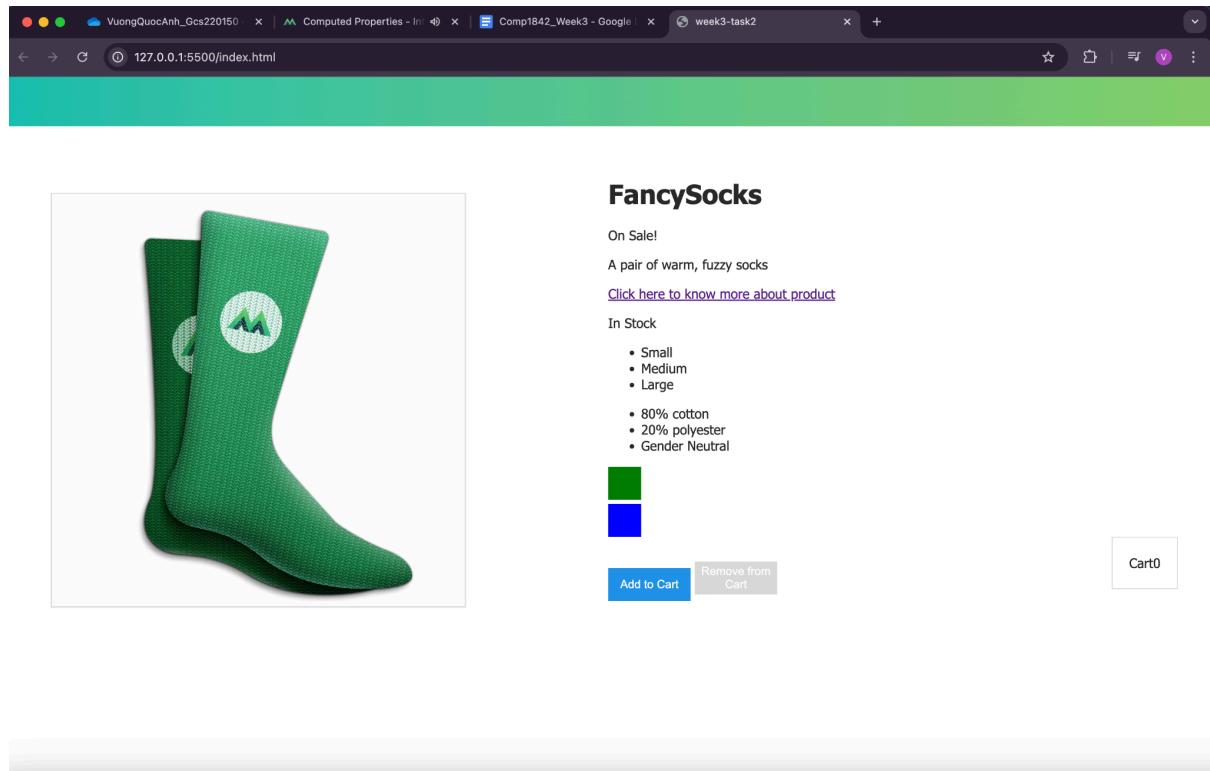


Fig14:Adding remove from cart button

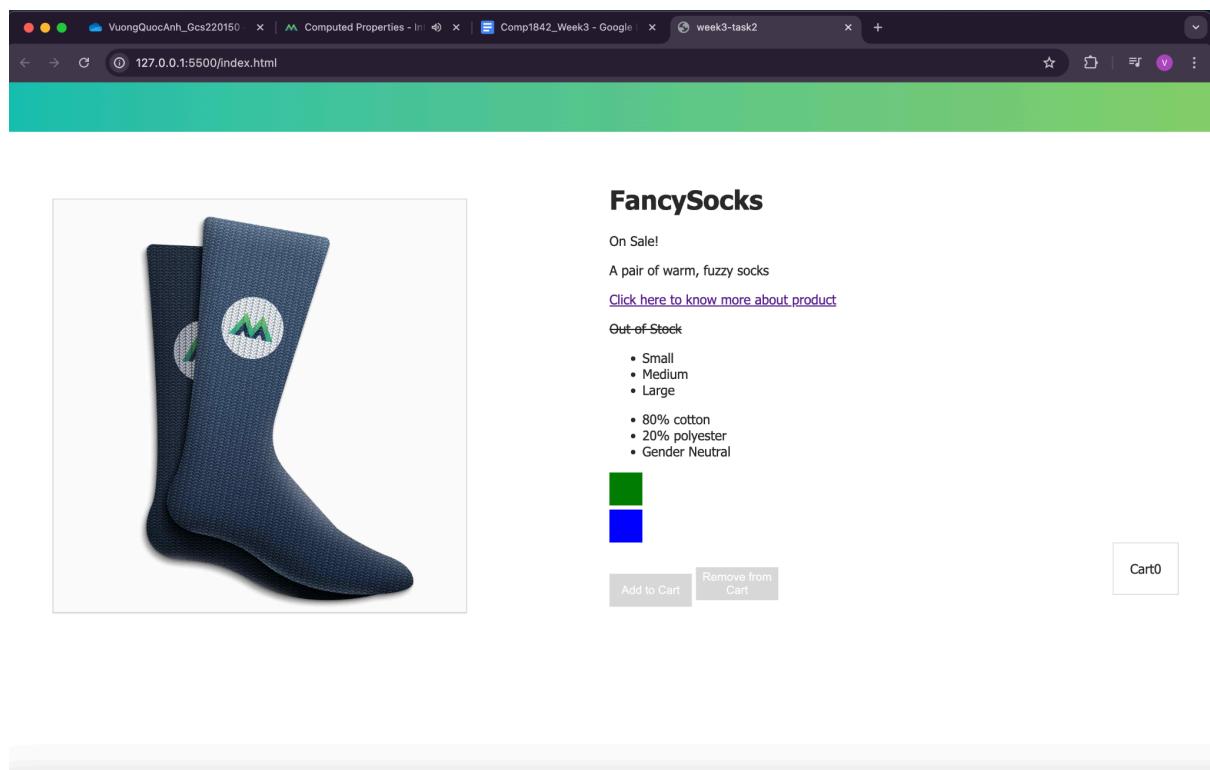


Fig15:Adding line-through

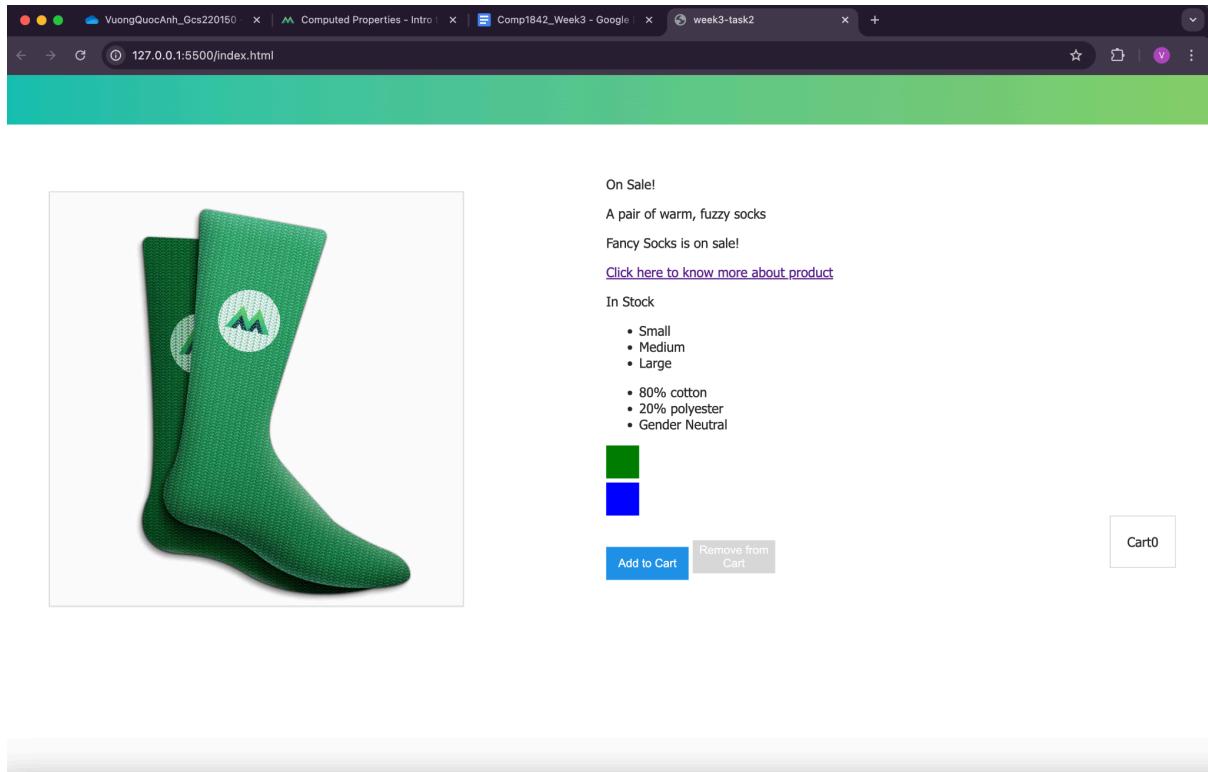


Fig16:Adding SaleMessage computed property

```

<html lang="en">
  <body>
    <div id="app">
      <div class="product">
        <div class="product-info">
          <p>{{ description }}</p>
          <p>{{ saleMessage }}</p>
          <a href="#" target="_blank">Click here to know more about product</a>
          <p>v-if="inStock">In Stock</p>
          <p>v-else v-bind:class="{'line-through': !inStock}">Out of Stock</p>
        </div>
        <ul>
          <li v-for="size in sizes">{{ size }}</li>
        </ul>
        <ul>
          <li v-for="detail in details">{{ detail }}</li>
        </ul>
        <div v-for="(variant, index) in variants"
            :key="variant.variantId"
            class="color-box"
            style="background-color: variant.variantColor"
            @mouseover="updateProduct(index)">
          </div>
        <button v-on:click="addToCart"
                :disabled="!inStock"
                :class="{'disabledButton': !inStock}">Add to Cart</button>
        <button v-on:click="removeFromCart"
                :disabled="cart <= 0"
                :class="{'disabledButton': cart <= 0}">Remove from Cart</button>
      </div>
      <div class="cart">
        <p>Cart {{ cart }}</p>
      </div>
    </div>
  </body>
</html>

```

Fig17:Html after doing challenges

```

    1 var app = new Vue({
2   data: {
3     variants: [
4       {
5         variantId: 2235,
6         variantColor: "blue",
7         variantImage: '../assets/images/socks_blue.jpg',
8         variantQuantity: 0
9       }
10      ],
11      cart: 0
12    },
13    methods: {
14      addCart() {
15        this.cart += 1;
16      },
17      removeFromCart() {
18        if (this.cart > 0) {
19          this.cart -= 1;
20        }
21      },
22      updateProduct(index) {
23        this.selectedVariant = index;
24        console.log(index);
25      }
26    },
27    computed: {
28      fullTitle() {
29        return this.brand + ' ' + this.product;
30      },
31      image() {
32        return this.variants[this.selectedVariant].variantImage;
33      },
34      inStock() {
35        return this.variants[this.selectedVariant].variantQuantity > 0;
36      },
37      saleMessage() {
38        if (this.onSale) {
39          return `${this.brand} ${this.product} is on sale!`;
40        }
41        return '';
42      }
43    }
44  });
45  
```

**Fig18:Js after doing challenges**

## Appendix3:

### 3.1 Task:

Vue components are like building blocks for an app. They help break up the app into smaller, reusable pieces, each with its own data and behaviour. This makes the app easier to manage and build.

In Vue, “props” are used to send data from a parent to a child component. Think of props as a way for the parent to give information to the child. You add props to a component through a special attribute, and you can set rules for the kind of data it can receive.

Components can also talk back to the parent. A child component can use `\\$emit` to send a message or data when something happens. The parent listens for this with `v-on` (or `@` for short) and runs a method when it hears the event.

For forms, Vue’s `v-model` creates a two-way data link between the input and the data. The `number` modifier can turn the input into a number, but watch out—there’s a known bug. Use `prevent` to stop the form from reloading the page when you submit. With Vue, you can add simple checks to make sure the form is filled out correctly.

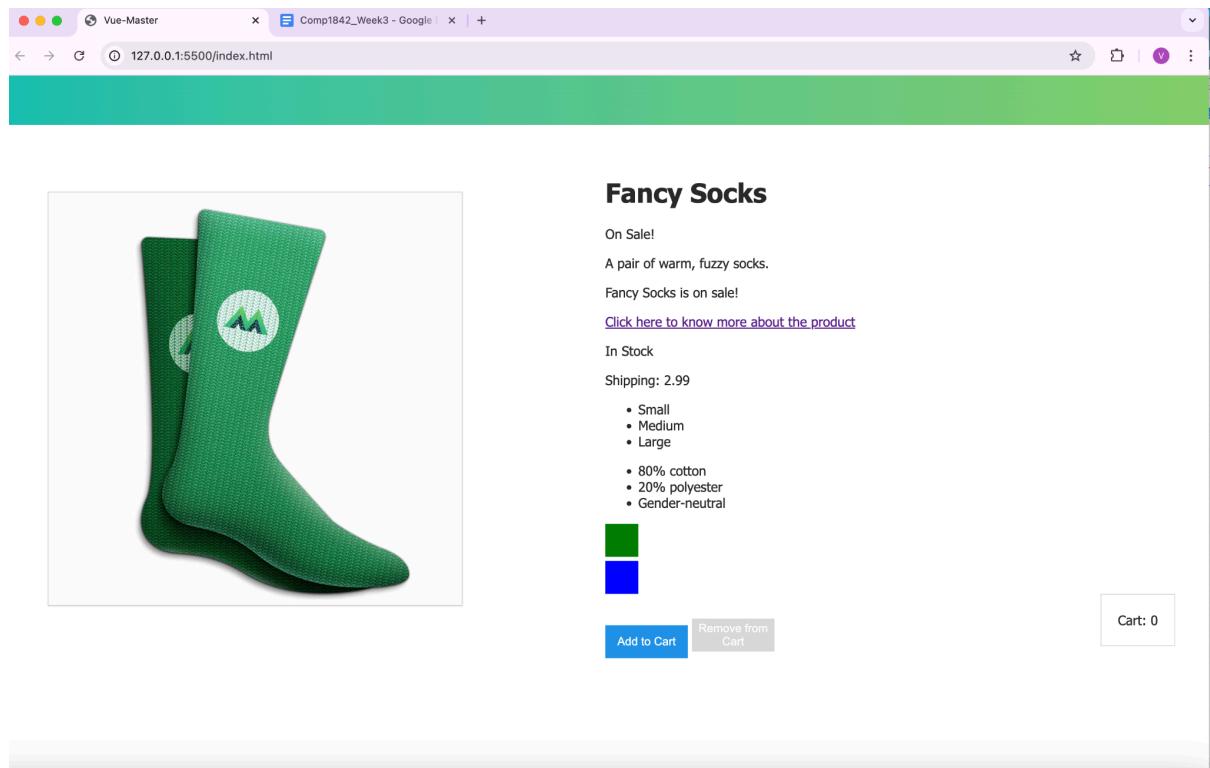


Fig19: Add components

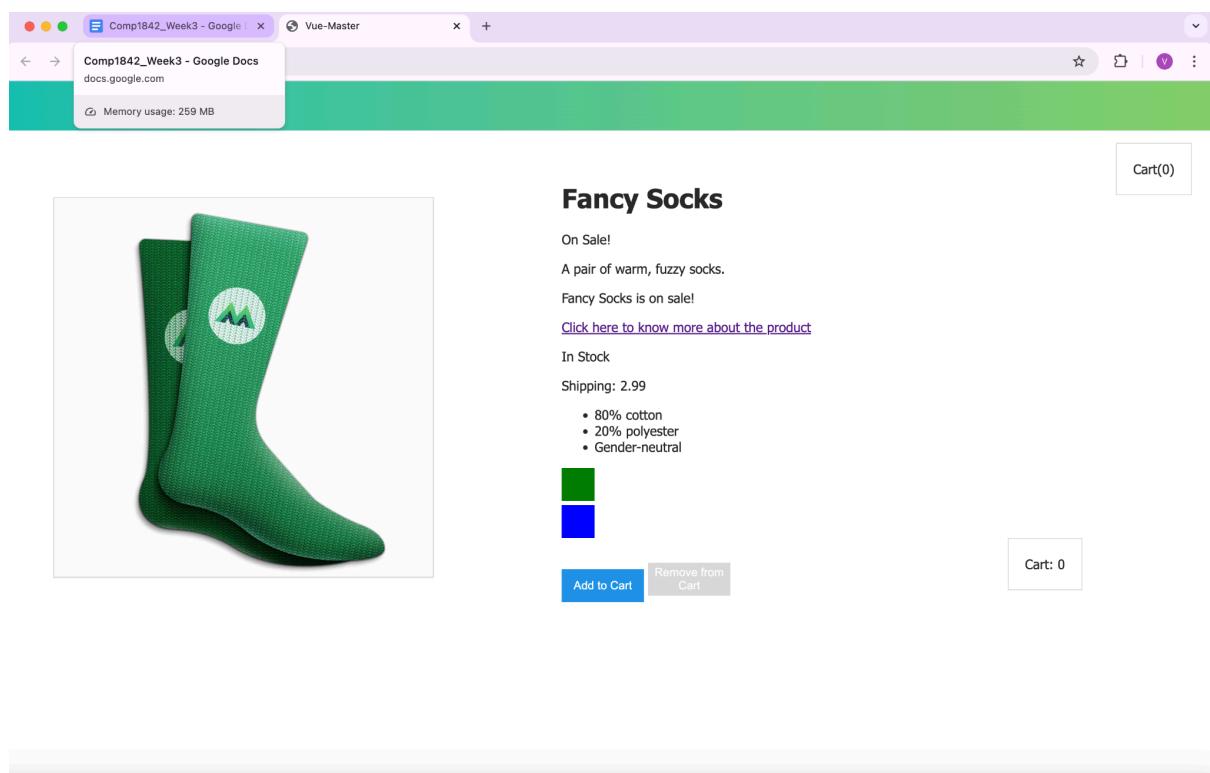


Fig20: Using emit

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Vue-Master</title>
    <!-- Import Styles -->
    <link rel="stylesheet" href="assets/styles.css" />
  </head>
  <body>
    <div class="nav-bar"></div>
    <div id="app">
      <div class="cart">
        <p>Cart({ cart.length })</p>
      </div>
      <product :premium="premium" @add-to-cart="updateCart"></product>
    </div>
    </div>
    <!-- Import Vue.js -->
    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.js"></script>
    <!-- Import Js -->
    <script src="main.js"></script>
  </body>
</html>
```

Fig21: html

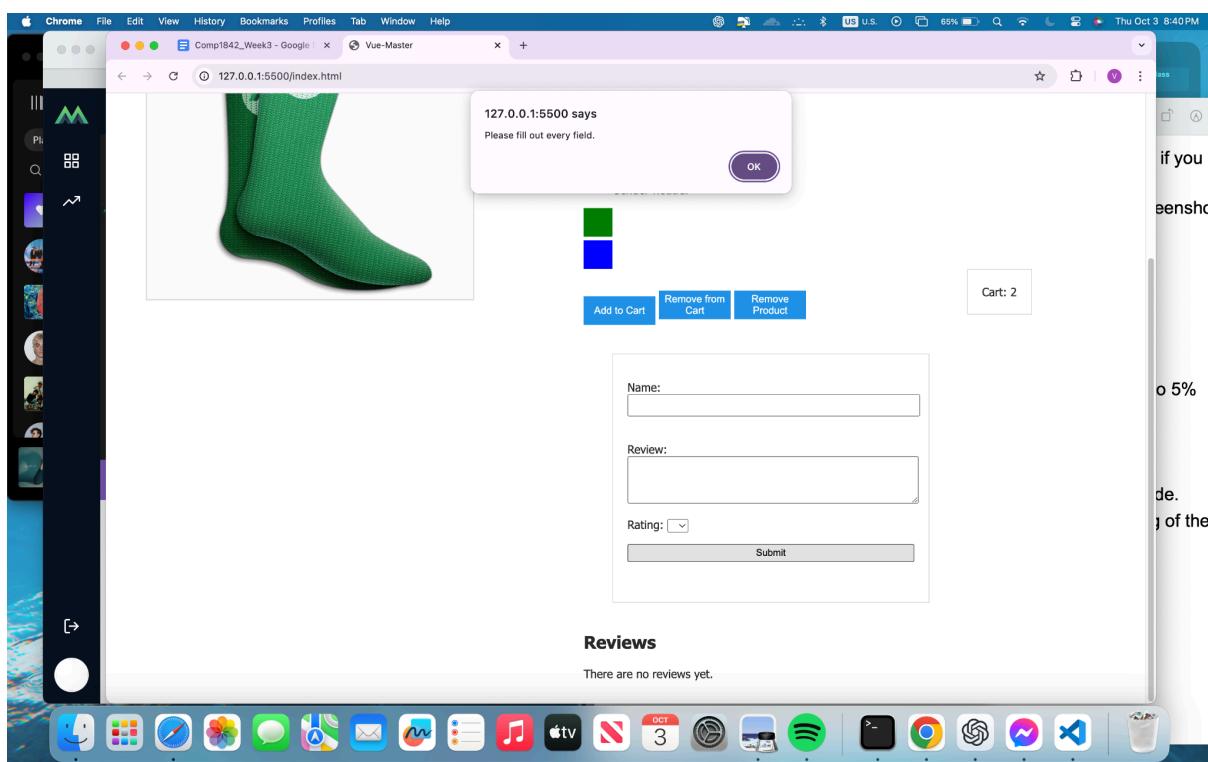
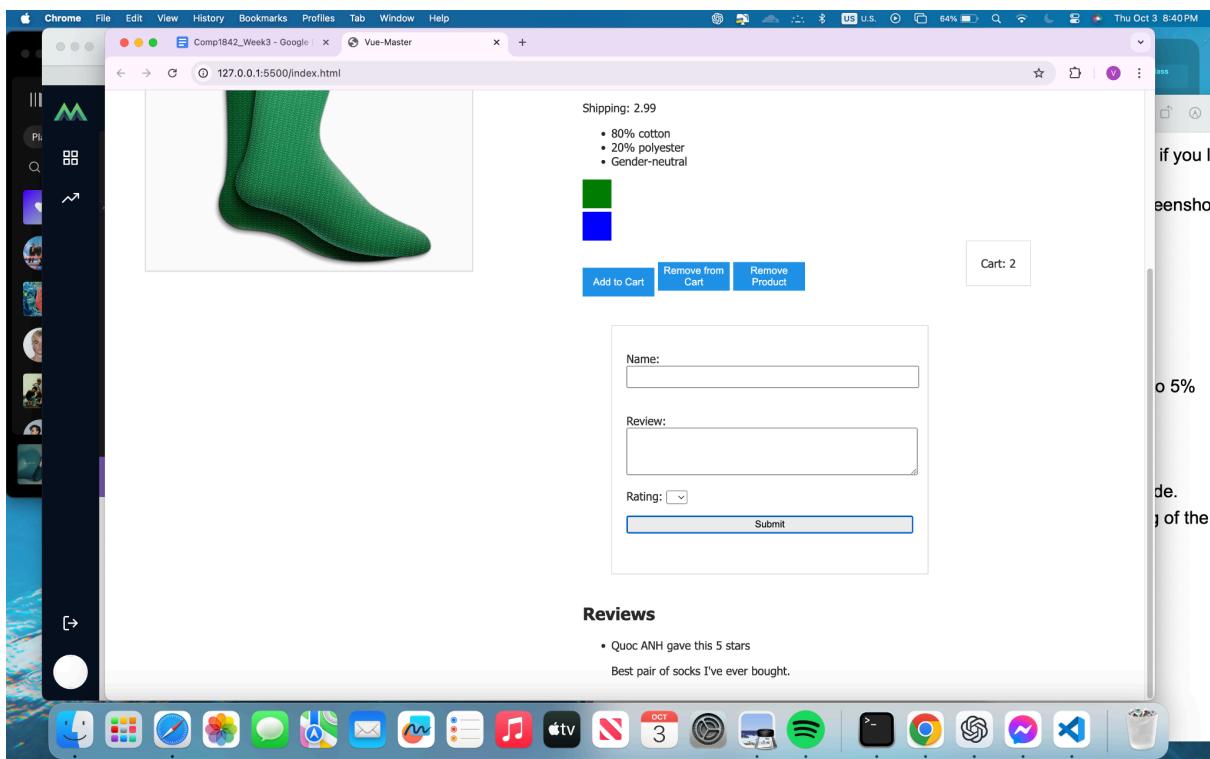
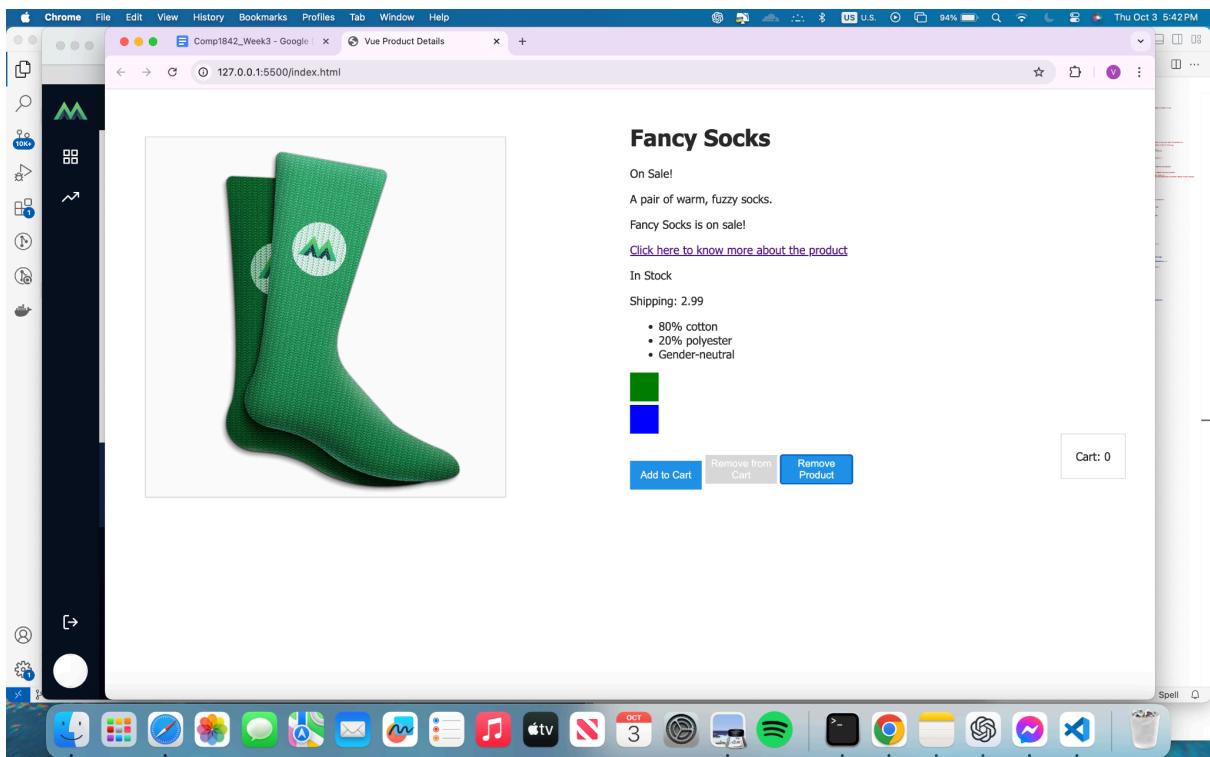


Fig22: Without filling any blank



**Fig23:** Add name and review

### 3.2 Challenges:



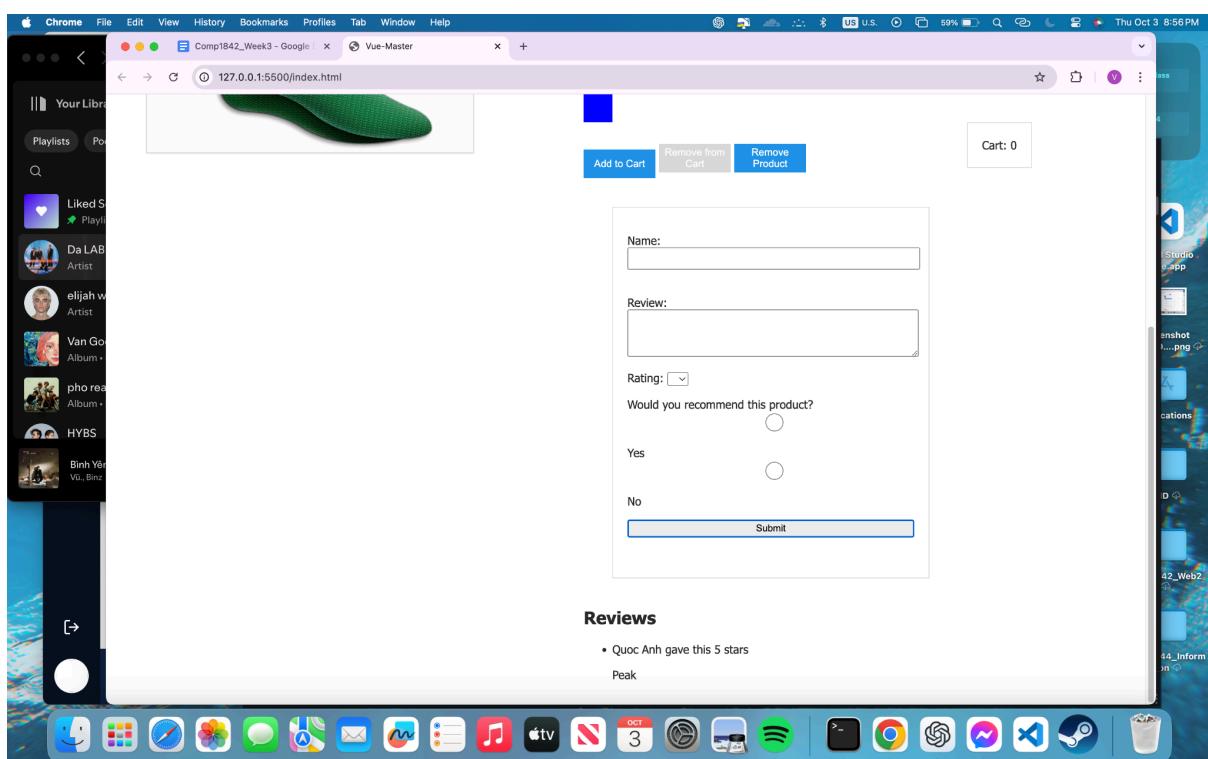
**Fig24:** Using emit to add remove product

Using following emit code to add remove product

```

        },
        emitRemoveProduct(productId) {
            // Emit the event to remove the product with its ID
            this.$emit('remove-product', productId);
        }
    },
},
methods: {
    removeProduct(productId) {
        // Remove the product from the cart by its ID
        this.cart = this.cart.filter(item => item !== productId);
    }
}
);

```



**Fig25:** Add recommendation with yes/no button

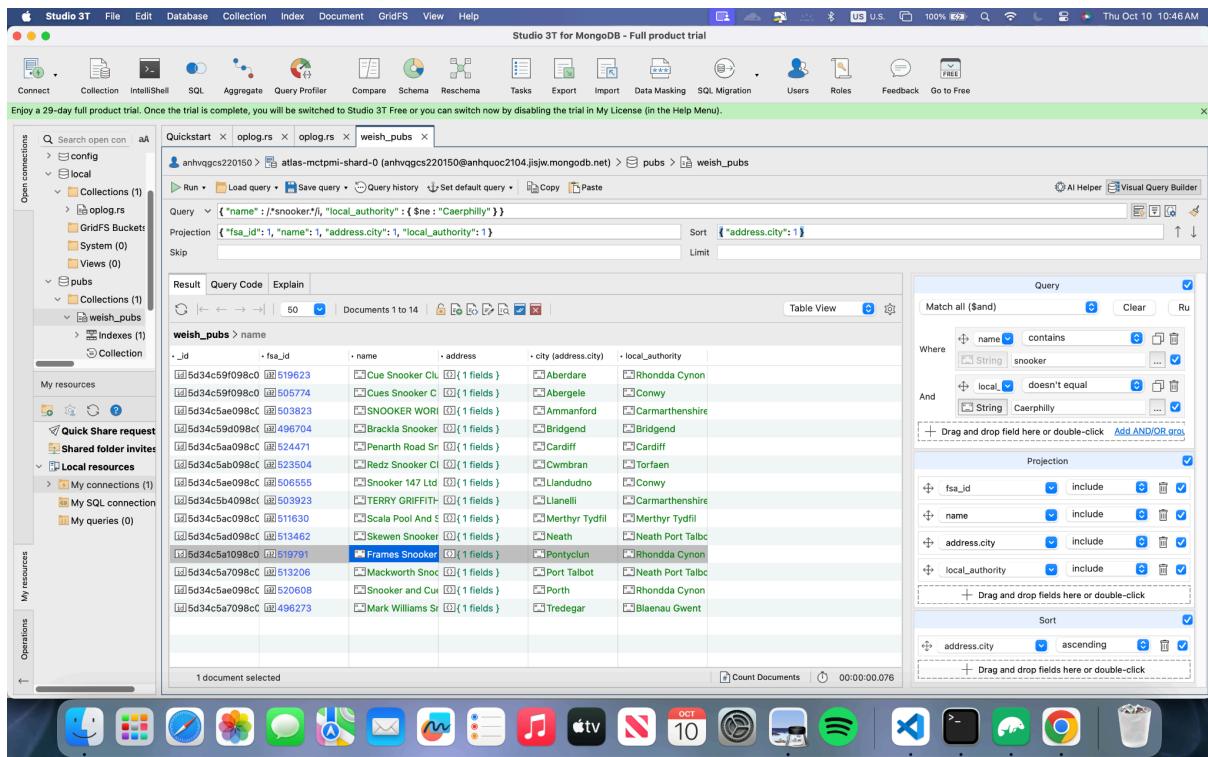
**Fig26: main.js after doing challenges**

## Appendix4:

In Lesson 4, I am learning how to manage collections and query data effectively. **Exercise 1** teaches me how to create a collection from a JSON file, a common format for data storage and transfer. In **Exercise 2**, I explore different views of the collection, which is vital for organising and analysing data efficiently. Finally, **Exercise 3** introduces me to using a Visual Query Builder to extract information from the dataset without writing code manually. These exercises help develop key skills in data management, ensuring I can handle collections and perform queries in various practical applications.

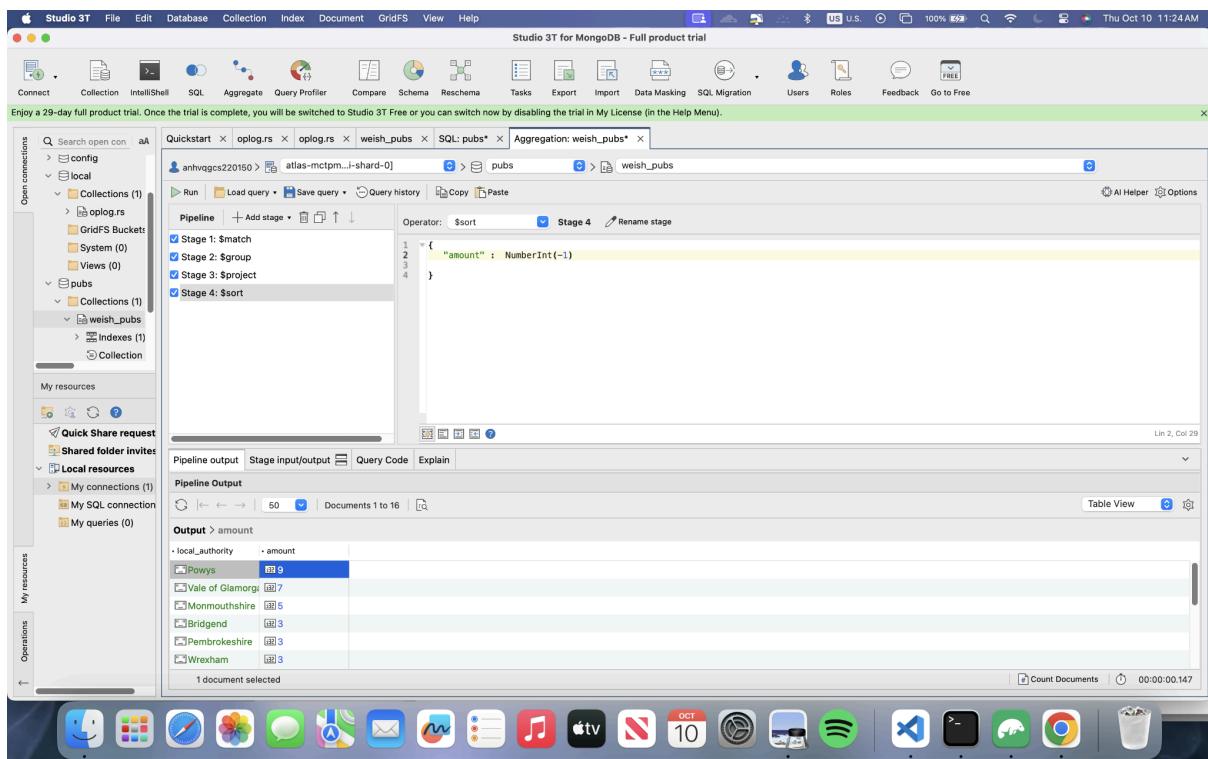
The screenshot shows the Studio 3T application interface for MongoDB. The title bar indicates it's running on a Mac with a 100% battery level and the date/time as Thu Oct 10 10:17AM. The main window has several tabs at the top: Quickstart, Tasks, Compare/Sync, IntelliShell, and others. The current tab is 'weish\_pubs'. The left sidebar shows 'Open connections' (Replica Set Members, admin, local, pufs), 'My resources' (Quick Share requests, Shared folder invites, Local resources: My connections, My SQL connections, My queries), and 'Operations'. The central area displays the 'weish\_pubs' collection with a table view showing documents like 'HOPE & ANCHO', 'Hope & Anchor', etc., with columns for \_id, fsa\_id, name, address, and contact. To the right is a 'Visual Query Builder' panel with sections for Match all (\$and), Projection, and Sort. The bottom of the screen shows the Mac OS X dock with various application icons.

**Fig27:Lesson4 exercise 1**



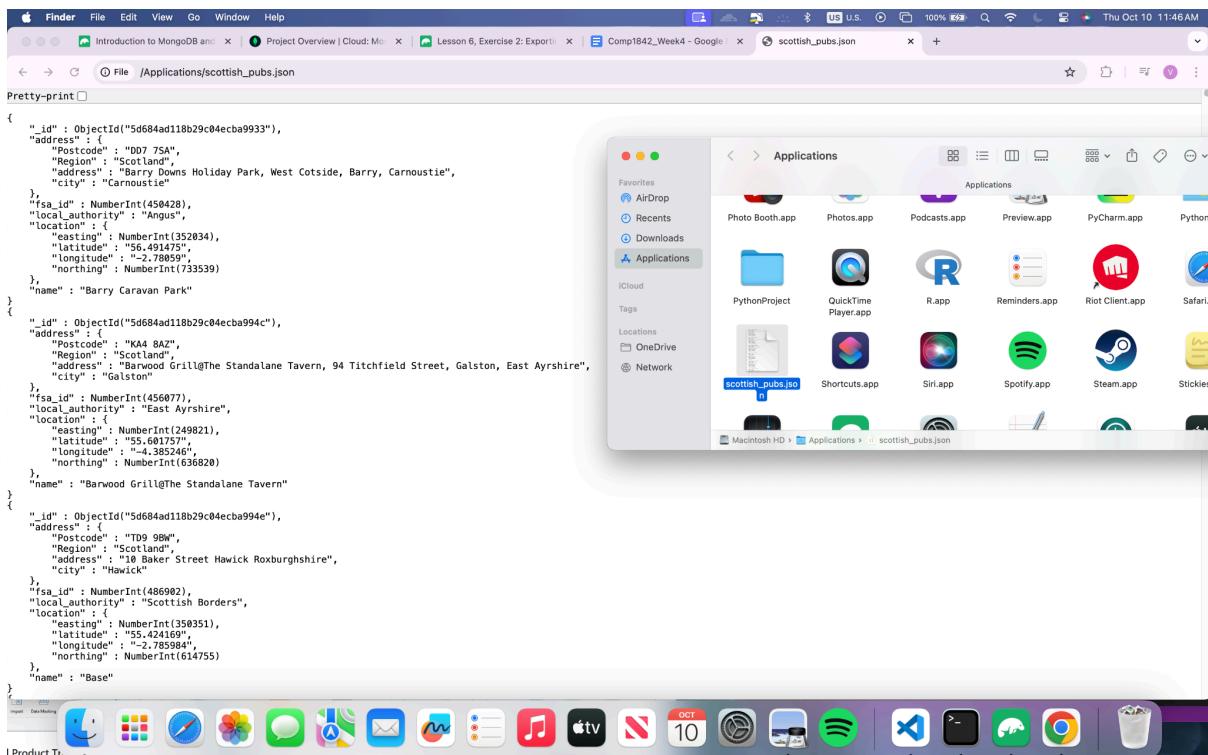
**Fig28:Lesson4 exercise 3**

In Lesson 5, I am developing proficiency in executing and refining SQL aggregate queries within MongoDB, a key skill for working with large datasets. Exercise 1 introduces the process of running a SQL aggregate query in MongoDB, demonstrating how SQL-based logic can be adapted to a NoSQL environment. Exercise 2 emphasises exporting SQL queries into the Aggregation Editor, providing a seamless method to transition between SQL and MongoDB's native aggregation framework. Finally, Exercise 3 covers editing queries in the Aggregation Editor, enhancing my ability to modify and optimise queries for more efficient data processing and analysis within MongoDB.



**Fig29:Lesson5 exercise 3**

In Lesson 6, I am learning how to manage document data through import and export processes. Exercise 1 focuses on importing data from a CSV file, a crucial skill for handling structured datasets. Exercise 2 teaches how to export document data to a JSON file, enabling data transfer and storage.



**Fig30: Lesson6 exercise 2**

**Git Code: <https://github.com/AnhQuoc1234/lab>**