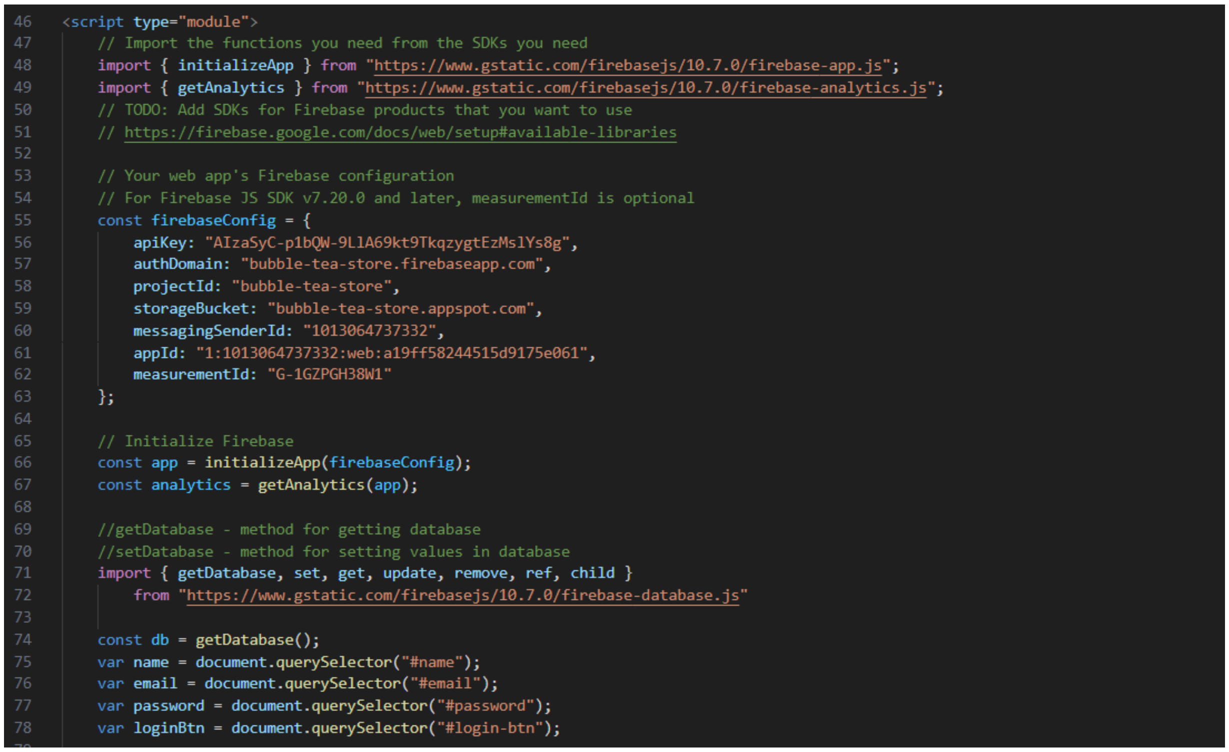
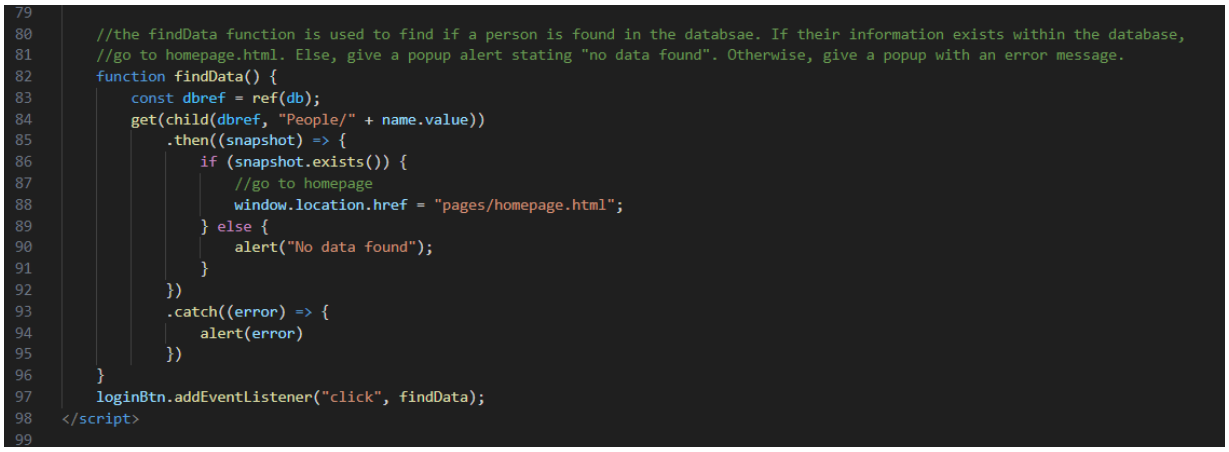
Project 2: Gray Paper

Sammi Wong
Matthew Fried
CSCI355: Web and Internet Technology
December 10, 2023

1. index.html



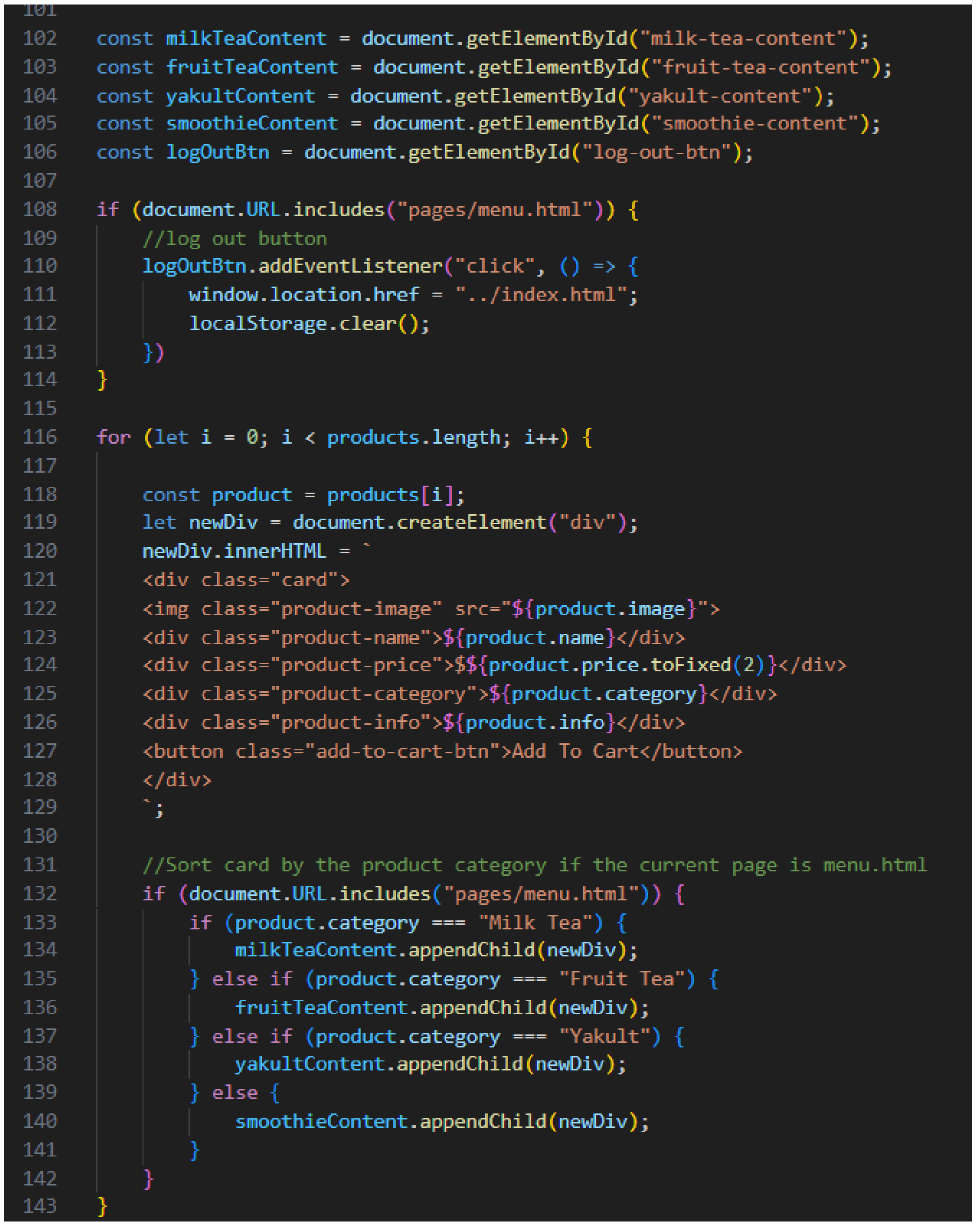
```
46  <script type="module">
47      // Import the functions you need from the SDKs you need
48      import { initializeApp } from "https://www.gstatic.com/firebasejs/10.7.0/firebase-app.js";
49      import { getAnalytics } from "https://www.gstatic.com/firebasejs/10.7.0/firebase-analytics.js";
50      // TODO: Add SDKs for Firebase products that you want to use
51      // https://firebase.google.com/docs/web/setup#available-libraries
52
53      // Your web app's Firebase configuration
54      // For Firebase JS SDK v7.20.0 and later, measurementId is optional
55      const firebaseConfig = {
56          apiKey: "AIzaSyC-p1bQW-9LlA69kt9TkqzygtEzMslYs8g",
57          authDomain: "bubble-tea-store.firebaseapp.com",
58          projectId: "bubble-tea-store",
59          storageBucket: "bubble-tea-store.appspot.com",
60          messagingSenderId: "1013064737332",
61          appId: "1:1013064737332:web:a19ff58244515d9175e061",
62          measurementId: "G-1GZPGH38W1"
63      };
64
65      // Initialize Firebase
66      const app = initializeApp(firebaseConfig);
67      const analytics = getAnalytics(app);
68
69      //getDatabase - method for getting database
70      //setDatabase - method for setting values in database
71      import { getDatabase, set, get, update, remove, ref, child }
72          from "https://www.gstatic.com/firebasejs/10.7.0/firebase-database.js"
73
74      const db = getDatabase();
75      var name = document.querySelector("#name");
76      var email = document.querySelector("#email");
77      var password = document.querySelector("#password");
78      var loginBtn = document.querySelector("#login-btn");
```



```
79
80      //the findData function is used to find if a person is found in the databsae. If their information exists within the database,
81      //go to homepage.html. Else, give a popup alert stating "no data found". Otherwise, give a popup with an error message.
82      function findData() {
83          const dbref = ref(db);
84          get(child(dbref, "People/" + name.value))
85              .then((snapshot) => {
86                  if (snapshot.exists()) {
87                      //go to homepage
88                      window.location.href = "pages/homepage.html";
89                  } else {
90                      alert("No data found");
91                  }
92              })
93              .catch((error) => {
94                  alert(error)
95              })
96      }
97      loginBtn.addEventListener("click", findData);
98  </script>
99
```

I used the script tag to create a database in the index.html. The variable firebaseConfig is implemented through FireBase. The app variable is used to initialize the firebaseConfig variable. I also imported methods including getDatabase, set, get, update, remove, ref, child which would primarily be used to get and set the login and register information in the database. Line 74 to 78 are used to initialize values that are set in the DOM. Subsequently, the function findData() is used to find if a person is found in the database. If their information exists within the database, users would be led to the homepage.html. Else, give a popup alert stating "no data found". Otherwise, give a popup with an error message.

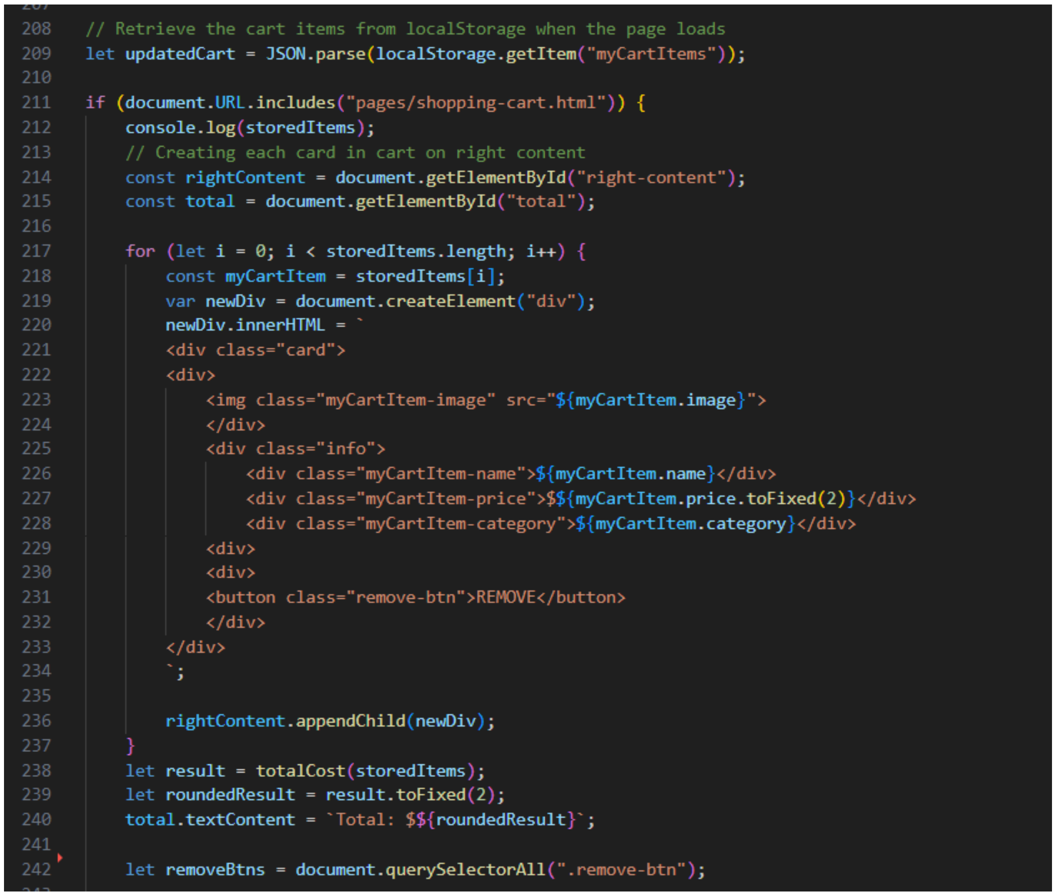
2. menu.js



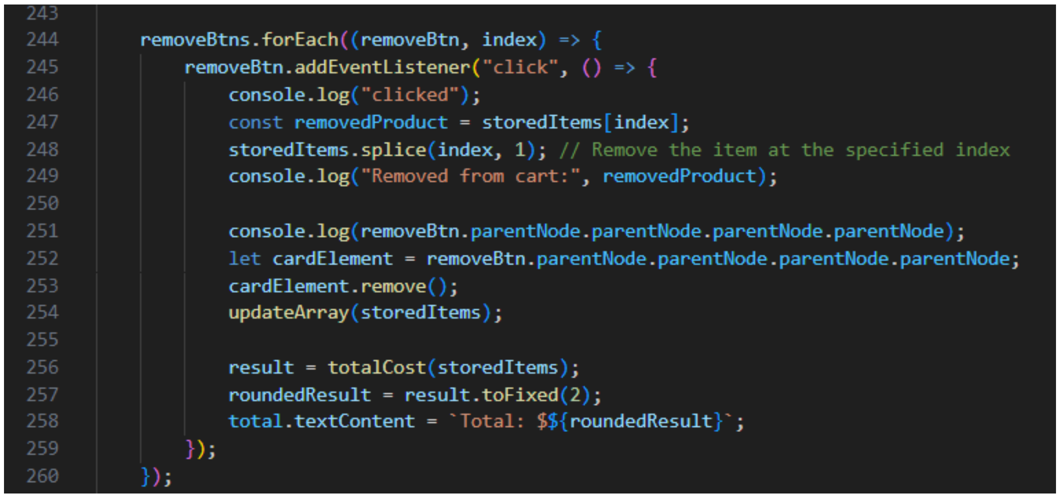
```
102  const milkTeaContent = document.getElementById("milk-tea-content");
103  const fruitTeaContent = document.getElementById("fruit-tea-content");
104  const yakultContent = document.getElementById("yakult-content");
105  const smoothieContent = document.getElementById("smoothie-content");
106  const logOutBtn = document.getElementById("log-out-btn");
107
108  if (document.URL.includes("pages/menu.html")) {
109      //log out button
110      logOutBtn.addEventListener("click", () => {
111          window.location.href = "../index.html";
112          localStorage.clear();
113      })
114  }
115
116  for (let i = 0; i < products.length; i++) {
117
118      const product = products[i];
119      let newDiv = document.createElement("div");
120      newDiv.innerHTML = `
121      <div class="card">
122      <img class="product-image" src="${product.image}">
123      <div class="product-name">${product.name}</div>
124      <div class="product-price">$${product.price.toFixed(2)}</div>
125      <div class="product-category">${product.category}</div>
126      <div class="product-info">${product.info}</div>
127      <button class="add-to-cart-btn">Add To Cart</button>
128      </div>
129      `;
130
131      //Sort card by the product category if the current page is menu.html
132      if (document.URL.includes("pages/menu.html")) {
133          if (product.category === "Milk Tea") {
134              milkTeaContent.appendChild(newDiv);
135          } else if (product.category === "Fruit Tea") {
136              fruitTeaContent.appendChild(newDiv);
137          } else if (product.category === "Yakult") {
138              yakultContent.appendChild(newDiv);
139          } else {
140              smoothieContent.appendChild(newDiv);
141          }
142      }
143  }
```

Line 108 to 114 is an if statement. If the current document URL is on the index.html page, then add an event listener for the logOutBtn. The if statement is necessary to prevent an error as the code is written in the menu.js which is also used for other html pages as well. Line 116 to 143 is used for each product through a for loop. For each element, create a div with innerHTML dynamically. The innerHTML creates an image, a div for the product's name, price, category, information, and a button to add the item to cart. Subsequently, line 132 to 143 is an if statement that organizes the products in accordance to the product's category.

3. menu.js



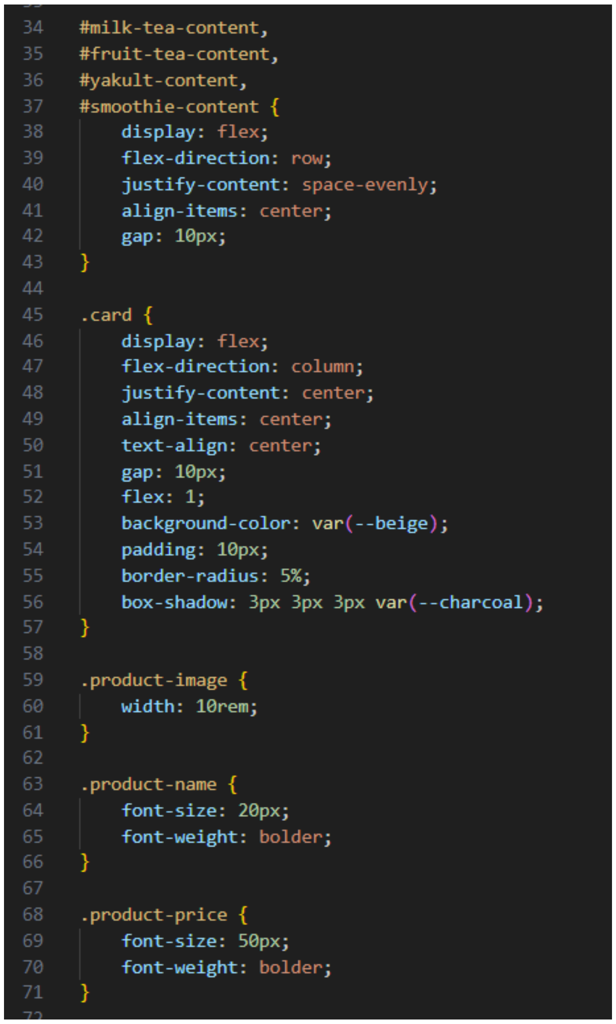
```javascript
// Retrieve the cart items from localStorage when the page loads
let updatedCart = JSON.parse(localStorage.getItem("myCartItems"));

if (document.URL.includes("pages/shopping-cart.html")) {
    console.log(storedItems);
    // Creating each card in cart on right content
    const rightContent = document.getElementById("right-content");
    const total = document.getElementById("total");

    for (let i = 0; i < storedItems.length; i++) {
        const myCartItem = storedItems[i];
        var newDiv = document.createElement("div");
        newDiv.innerHTML = `
        <div class="card">
        <div>
            <img class="myCartItem-image" src="${myCartItem.image}">
            </div>
            <div class="info">
                <div class="myCartItem-name">${myCartItem.name}</div>
                <div class="myCartItem-price">$${myCartItem.price.toFixed(2)}</div>
                <div class="myCartItem-category">${myCartItem.category}</div>
            <div>
            <div>
            <button class="remove-btn">REMOVE</button>
            </div>
        </div>
        `;

        rightContent.appendChild(newDiv);
    }
    let result = totalCost(storedItems);
    let roundedResult = result.toFixed(2);
    total.textContent = `Total: $${roundedResult}`;

    let removeBtns = document.querySelectorAll(".remove-btn");
```



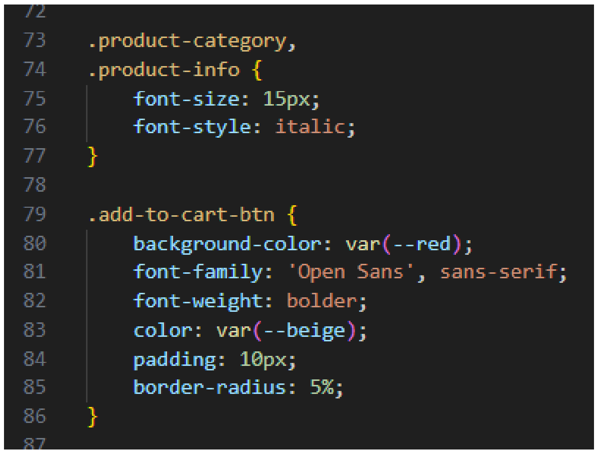
```javascript

    removeBtns.forEach((removeBtn, index) => {
        removeBtn.addEventListener("click", () => {
            console.log("clicked");
            const removedProduct = storedItems[index];
            storedItems.splice(index, 1); // Remove the item at the specified index
            console.log("Removed from cart:", removedProduct);

            console.log(removeBtn.parentNode.parentNode.parentNode.parentNode);
            let cardElement = removeBtn.parentNode.parentNode.parentNode.parentNode;
            cardElement.remove();
            updateArray(storedItems);

            result = totalCost(storedItems);
            roundedResult = result.toFixed(2);
            total.textContent = `Total: $${roundedResult}`;
        });
    });
```

If the current page is on the shopping cart page, create a container which displays the products added to cart when the add to cart button is clicked on the menu page. For each item added to cart, create a new div which contains the product's image, name, price, category, and remove button. Next, the div will be appended to the content on the right side of the page. For each removeBtns, add an event listener that removes the product from storedItems.

4. menu.css



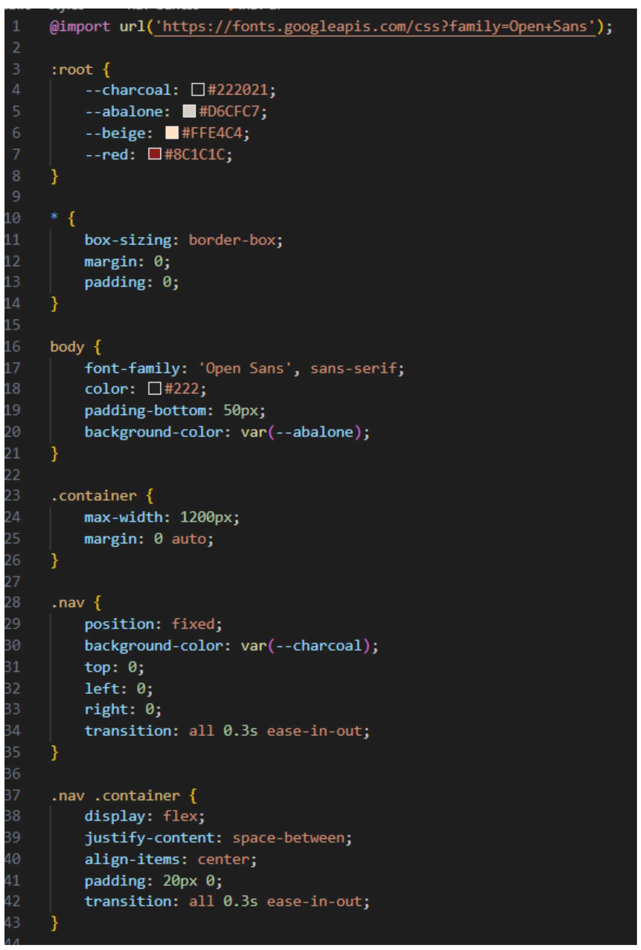
```
#milk-tea-content,
#fruit-tea-content,
#yakult-content,
#smoothie-content {
    display: flex;
    flex-direction: row;
    justify-content: space-evenly;
    align-items: center;
    gap: 10px;
}

.card {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    text-align: center;
    gap: 10px;
    flex: 1;
    background-color: var(--beige);
    padding: 10px;
    border-radius: 5%;
    box-shadow: 3px 3px 3px var(--charcoal);
}

.product-image {
    width: 10rem;
}

.product-name {
    font-size: 20px;
    font-weight: bolder;
}

.product-price {
    font-size: 50px;
    font-weight: bolder;
}
```



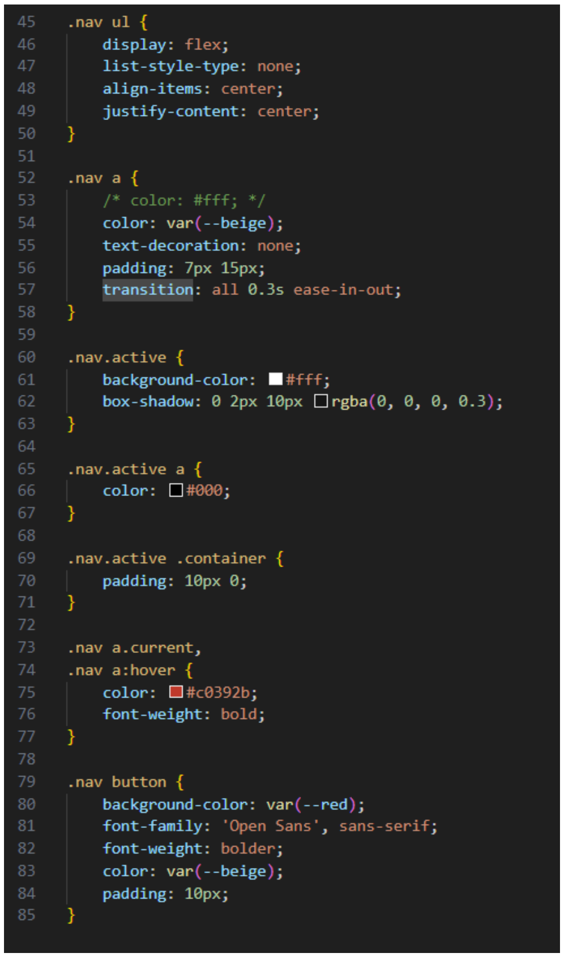
```
.product-category,
.product-info {
    font-size: 15px;
    font-style: italic;
}

.add-to-cart-btn {
    background-color: var(--red);
    font-family: 'Open Sans', sans-serif;
    font-weight: bolder;
    color: var(--beige);
    padding: 10px;
    border-radius: 5%;
}
```

This is the css code used for the menu page. For each category content, the display is set to row flex so the product would be displayed horizontally. The card class refers to each individual product. Similarly, it uses flex to display the product’s content centered and aligned vertically. The description for each product also have width, font-size, and font-weight properties to show consistency across all text information on the website.

5. nav-bar.css



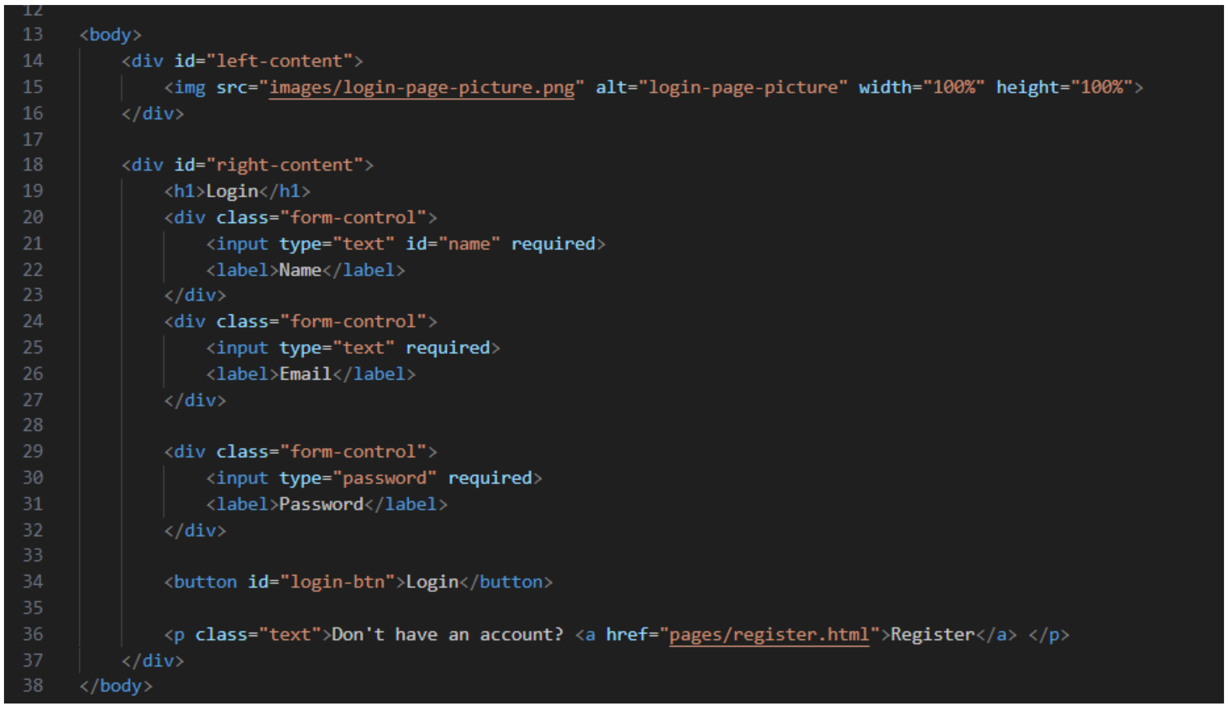
```css
@import url('https://fonts.googleapis.com/css?family=Open+Sans');

:root {
    --charcoal: #222021;
    --abalone: #D6CFC7;
    --beige: #FFE4C4;
    --red: #8C1C1C;
}

* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: 'Open Sans', sans-serif;
    color: #222;
    padding-bottom: 50px;
    background-color: var(--abalone);
}

.container {
    max-width: 1200px;
    margin: 0 auto;
}

.nav {
    position: fixed;
    background-color: var(--charcoal);
    top: 0;
    left: 0;
    right: 0;
    transition: all 0.3s ease-in-out;
}

.nav .container {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 20px 0;
    transition: all 0.3s ease-in-out;
}
```



```css
.nav ul {
    display: flex;
    list-style-type: none;
    align-items: center;
    justify-content: center;
}

.nav a {
    /* color: #fff; */
    color: var(--beige);
    text-decoration: none;
    padding: 7px 15px;
    transition: all 0.3s ease-in-out;
}

.nav.active {
    background-color: #fff;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);
}

.nav.active a {
    color: #000;
}

.nav.active .container {
    padding: 10px 0;
}

.nav a.current,
.nav a:hover {
    color: #c0392b;
    font-weight: bold;
}

.nav button {
    background-color: var(--red);
    font-family: 'Open Sans', sans-serif;
    font-weight: bolder;
    color: var(--beige);
    padding: 10px;
}
```

This is the navigation bar styling sheet used for all the html pages that displays the navigation bar. The position of the navigation bar is fixed so it would be stuck to the top of the page. Within the navigation bar, there are links that are displayed with flex. When the user hovers over the link, it will change the color to red. The button within the nav tag (which is the logout button) have properties including background-color, font-family, font-weight, color, and padding properties which are used for all buttons on the website.

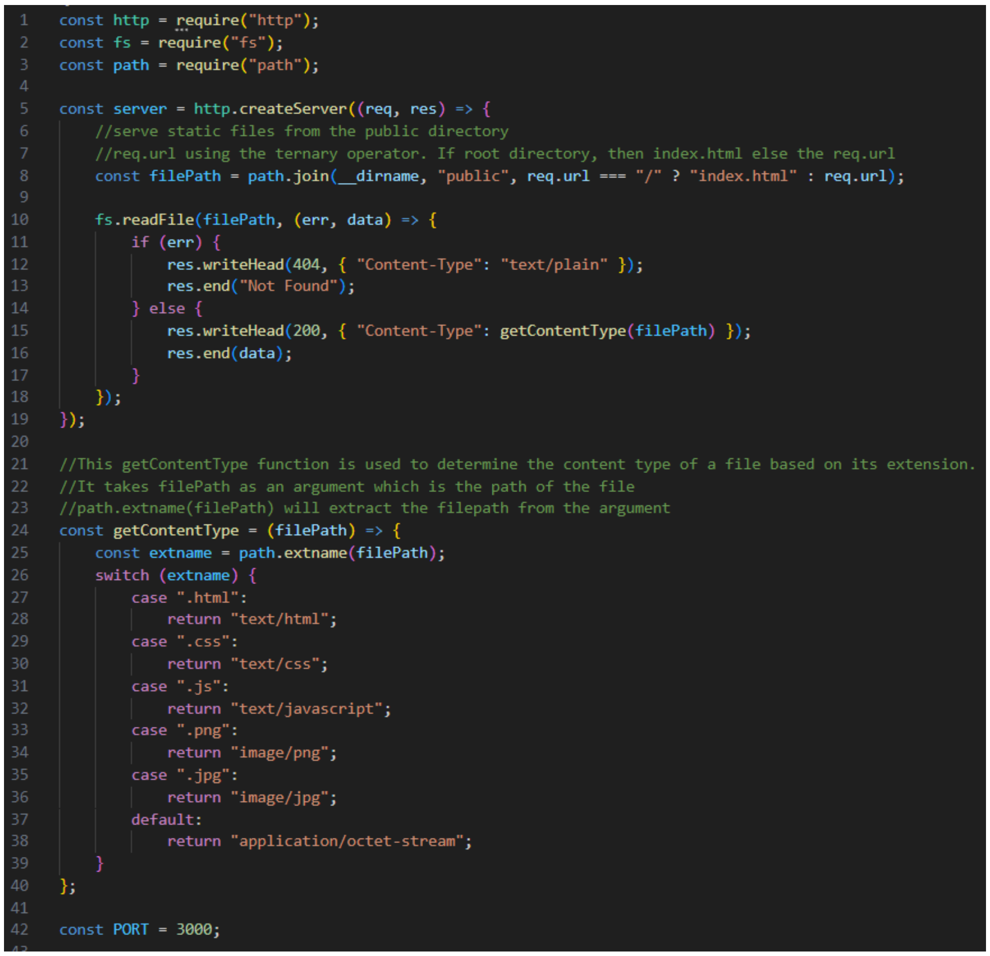
6. index.html



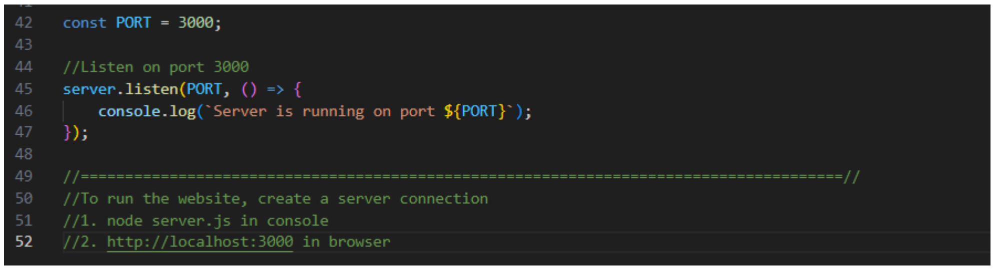
```html
<body>
    <div id="left-content">
        <img src="images/login-page-picture.png" alt="login-page-picture" width="100%" height="100%">
    </div>

    <div id="right-content">
        <h1>Login</h1>
        <div class="form-control">
            <input type="text" id="name" required>
            <label>Name</label>
        </div>
        <div class="form-control">
            <input type="text" required>
            <label>Email</label>
        </div>

        <div class="form-control">
            <input type="password" required>
            <label>Password</label>
        </div>

        <button id="login-btn">Login</button>

        <p class="text">Don't have an account? <a href="pages/register.html">Register</a> </p>
    </div>
</body>
```

This is the index.html page which is the login page as well. I used form control and tags with attributes such as text and password so the user can input their information. At the bottom of the page, there is a link tag that directs the user to the register page, which was setted up in a similar fashion.

7. server.js

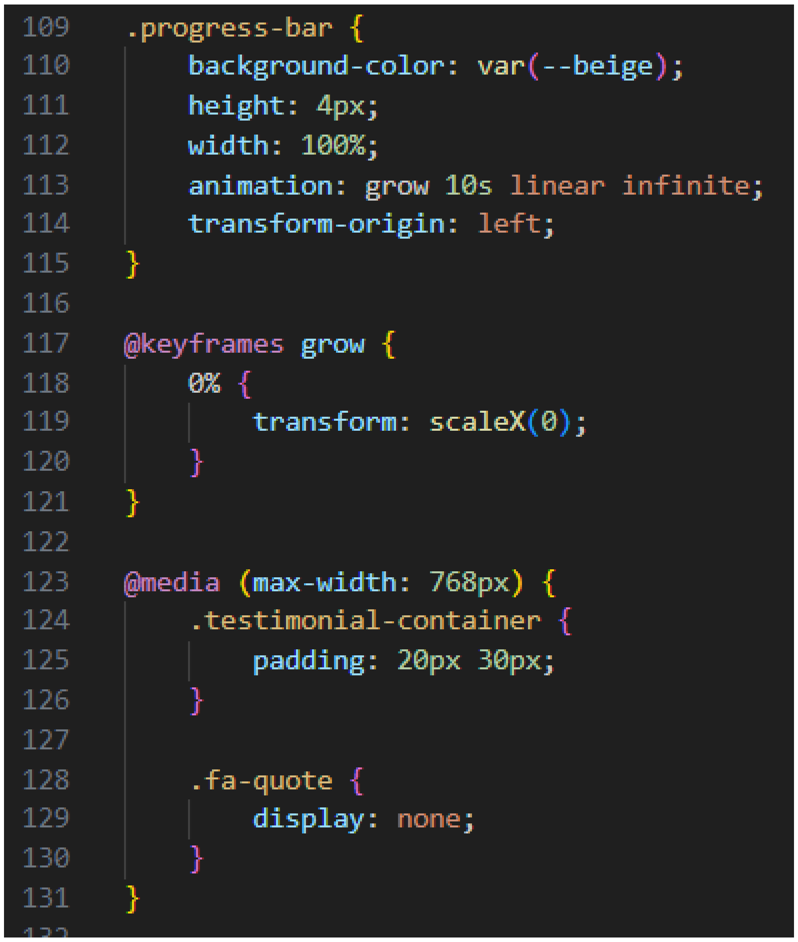


```
1   const http = require("http");
2   const fs = require("fs");
3   const path = require("path");
4
5   const server = http.createServer((req, res) => {
6       //serve static files from the public directory
7       //req.url using the ternary operator. If root directory, then index.html else the req.url
8       const filePath = path.join(__dirname, "public", req.url === "/" ? "index.html" : req.url);
9
10      fs.readFile(filePath, (err, data) => {
11          if (err) {
12              res.writeHead(404, { "Content-Type": "text/plain" });
13              res.end("Not Found");
14          } else {
15              res.writeHead(200, { "Content-Type": getContentType(filePath) });
16              res.end(data);
17          }
18      });
19  });
20
21  //This getContentType function is used to determine the content type of a file based on its extension.
22  //It takes filePath as an argument which is the path of the file
23  //path.extname(filePath) will extract the filepath from the argument
24  const getContentType = (filePath) => {
25      const extname = path.extname(filePath);
26      switch (extname) {
27          case ".html":
28              return "text/html";
29          case ".css":
30              return "text/css";
31          case ".js":
32              return "text/javascript";
33          case ".png":
34              return "image/png";
35          case ".jpg":
36              return "image/jpg";
37          default:
38              return "application/octet-stream";
39      }
40  };
41
42  const PORT = 3000;
```



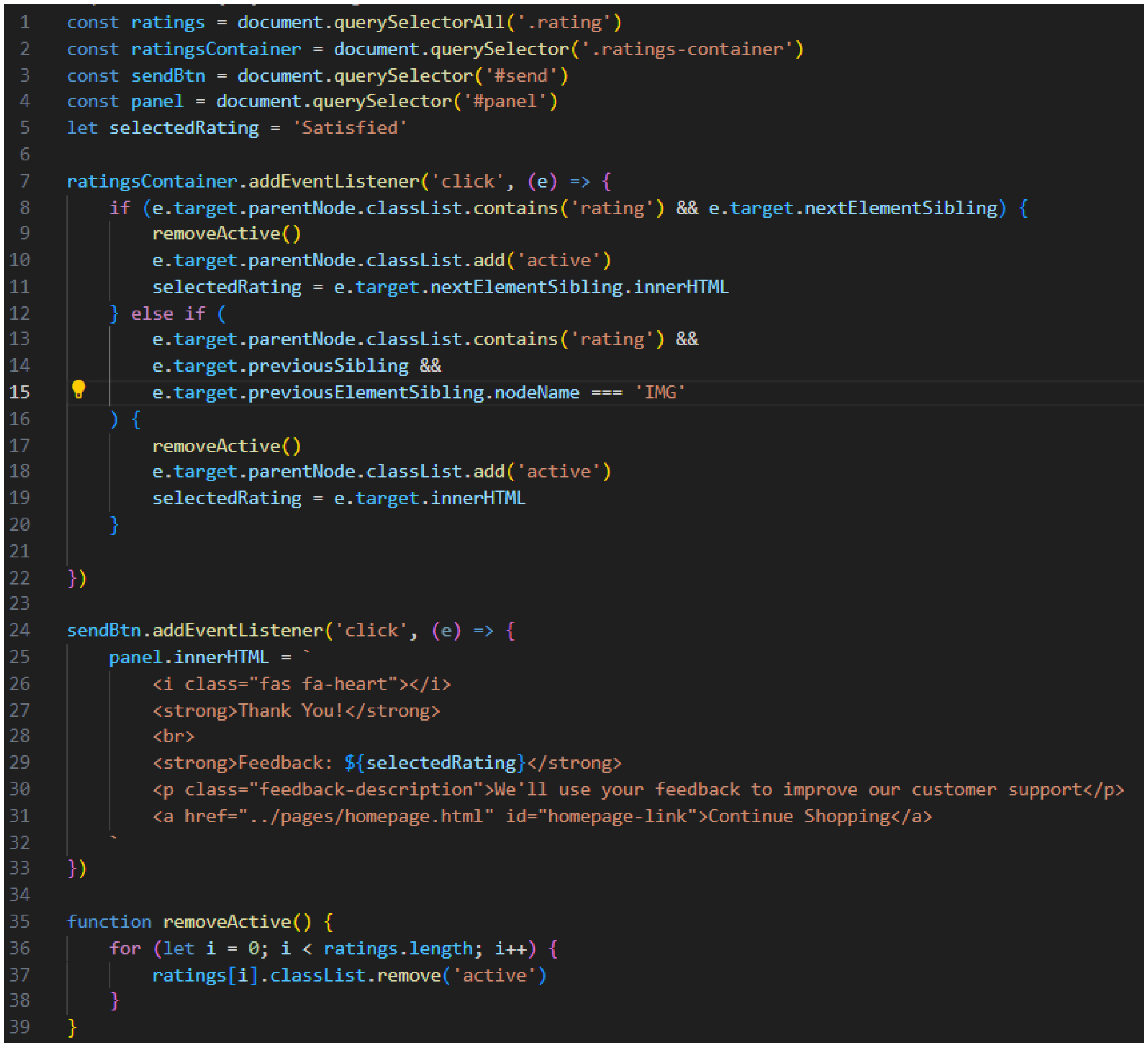
```
42  const PORT = 3000;
43
44  //Listen on port 3000
45  server.listen(PORT, () => {
46      console.log(`Server is running on port ${PORT}`);
47  });
48
49  //================================================================================//
50  //To run the website, create a server connection
51  //1. node server.js in console
52  //2. http://localhost:3000 in browser
```

This is the server.js file which is used to set up node server connection. The server variable is used to create a server connection. The filePath variable has a ternary operator. If the requested server is the root directory, then the user would be directed to the index.html. Otherwise, they would be directed to the requested url. Line 10 to 19 reads the filePath. If there is an error, send a 404 error message and display "Not Found" within the browser. Otherwise, returns a 200 response along with the filePath. Line 24 to 40 is a getContentType function which is used to determine the content type of the file based on its extension. It takes filePath as an argument which is the path of the file. path.extname(filePath) will extract the filePath from the argument. I use a switch case to determine its extension.

8. homepage.css

This is the styling sheet used for the homepage. The progress-bar class refers to the bar used in the testimonial. It has an animation named grow which starts with transform: scaleX(0), meaning that the bar is initially invisible.

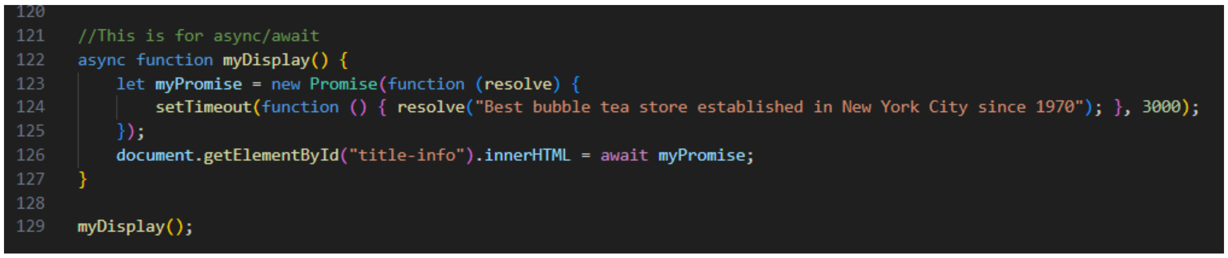
```css
.progress-bar {
    background-color: var(--beige);
    height: 4px;
    width: 100%;
    animation: grow 10s linear infinite;
    transform-origin: left;
}

@keyframes grow {
    0% {
        transform: scaleX(0);
    }
}

@media (max-width: 768px) {
    .testimonial-container {
        padding: 20px 30px;
    }

    .fa-quote {
        display: none;
    }
}
```

9. thank-you.js



```
const ratings = document.querySelectorAll('.rating')
const ratingsContainer = document.querySelector('.ratings-container')
const sendBtn = document.querySelector('#send')
const panel = document.querySelector('#panel')
let selectedRating = 'Satisfied'

ratingsContainer.addEventListener('click', (e) => {
    if (e.target.parentNode.classList.contains('rating') && e.target.nextElementSibling) {
        removeActive()
        e.target.parentNode.classList.add('active')
        selectedRating = e.target.nextElementSibling.innerHTML
    } else if (
        e.target.parentNode.classList.contains('rating') &&
        e.target.previousSibling &&
        e.target.previousElementSibling.nodeName === 'IMG'
    ) {
        removeActive()
        e.target.parentNode.classList.add('active')
        selectedRating = e.target.innerHTML
    }

})

sendBtn.addEventListener('click', (e) => {
    panel.innerHTML = `
        <i class="fas fa-heart"></i>
        <strong>Thank You!</strong>
        <br>
        <strong>Feedback: ${selectedRating}</strong>
        <p class="feedback-description">We'll use your feedback to improve our customer support</p>
        <a href="../pages/homepage.html" id="homepage-link">Continue Shopping</a>
    `
})

function removeActive() {
    for (let i = 0; i < ratings.length; i++) {
        ratings[i].classList.remove('active')
    }
}
```

This is the thank-you.js script file. There is an event listener on the ratingsContainer that is fired off when the user clicks on a rating. It will remove all previously active classes and set the clicked container to active. The sendBtn add event listener is also set off when the user clicks on the button which changes the panel's content through innerHTML, displaying a thank you message along with the rating selected.

10. homepage.js

This is the myDisplay function used along with the async and await. It makes a promise that the text “Best bubble tea store established in New York City since 1970) would be displayed after 3 seconds. Once it is revealed, the text would be changed for the title-info tag.

```javascript

//This is for async/await
async function myDisplay() {
    let myPromise = new Promise(function (resolve) {
        setTimeout(function () { resolve("Best bubble tea store established in New York City since 1970"); }, 3000);
    });
    document.getElementById("title-info").innerHTML = await myPromise;
}

myDisplay();
```

Challenges

A challenge that I encountered during the project is creating a database that tracks the user’s login information. Originally, I tried to set up the database using Node and mySql. I created the database in SQL Workbench but I couldn’t insert the user’s information unless it was done manually on SQL Workbench. I solved the problem by creating the database through Firebase by importing the necessary methods from the Firebase site. This includes get and set data methods which allows me to save the inputted values in javascript.