Client Side JavaScript

In Class Assessment 8 Displaying JSON DATA.

Submission
Upload your finished files to Netlify and copy and paste the url to the Netlify site in Moodle

Introduction
For the assessment you're going to use the fetch api to load a JSON file into your code and display the data inDOM. As you may have noticed by now that presenting data has a certain flow in code. Depending on your use case an effective flow might be as follows:

1. Import the JSON data file using fetch(“path to data”).
2. Create a copy of the data and hold it in a storage array (store).
3. Create a template that will hold the data using html and css.
4. Use the template with the store data and seed the template with data.
5. Convert the template to an element node.
6. Add any events to the template if required.
7. Return an array if desired from your templating function.
8. Display the markup elements in the DOM.

Request Data Using Fetch
Create a load event and fetch the data.json file. Create an empty array called store to copy the data from the request so that you have a backup of the data.

```
window.addEventListener("load", function (e) {
 const rentalRequest = fetch("./js/data.json");
 let store = [];
});
```

Handling The Promise
The fetch method returns a promise and as discussed a promise is just one way in javascript to handle asynchronous code. Promises have two parts, a resolve() method and a reject method(). The resolve method is returned to .then() portion of the promise and the reject() method is sent to the catch() portion of your promise code.

Create a .then(cb) block with a callback function. The callback function will contain the response from the requested URL when you used fetch(“./js/data.json”). The response contains the data but not in a format that javascript can use. So we use the .json() method to extract the json data and convert it to an array of objects. This again is not a synchronous task. It takes time to parse the json data. So the .json() method returns another promise.
Again we can access the resolve() method of the promise with a .then(cb) block and this time the return value from the resolve method is the parsed data an array of objects.

```
window.addEventListener("load", function (e) {
 const rentalRequest = fetch("./js/data.json");
 let store = [];
 rentalRequest
   .then((response) => response.json())
   .then((data) => {
     store = [...data];
   })
   .catch((error) => console.warn(`Error: ${error}`));
});
```

Make sure to create the store array, this array is used to store the data in your application. It’s a good rule of thumb not to manipulate this data. If you make any mistakes you will have to re-fetch the data. Make a new copy of the data by spreading the data into the store array.

Templating
The next step is to create a template that will hold the name value pairs or property value pairs for each object in the array that you plan on displaying.

Create a function called createMarkup() and for this example let’s return an array of elements.

```
window.addEventListener("load", function (e) {
 const rentalRequest = fetch("./js/data.json");
 let store = [];
 rentalRequest
   .then((response) => response.json())
   .then((data) => {
     store = [...data];
     const rentals = createMarkup();
   })
```

```
        .catch((error) => console.warn(`Error: ${error}`));
     });
```

createMarkup()
Create markup is where you will loop through the objects in the store array and seed the template with data. In this case we are converting the template strings to element nodes and returning each new element to the markup array created by the map() of the array

```
const createMarkup = function () {

 const markup = store.map(function (rental) {
   const imagePath = `./img/thumbnails/${rental.thumbnail}`;
   const template = `

   <aside class="rental">
     <header>
       <img class="thumbnail"src="${imagePath}" width="290"height="168" alt="rental
accommodation"
       />
     </header>
     <ul class="details">
       <li class="content">
         <div>
           <p class="type">${rental.rentalType}</p>
           <h3 class="location">${rental.location}</h3>
         </div>
         <div class="price">
           <p><span>$${rental.dailyRate}</span><span>/${rental.currency}</span></p>
         </div>
       </li>
       <li class="rating">
         <img class="star-icon" src="img/icons/rating.svg"alt="star rating"
           width="16px"
           height="16px"
         />
         <span>${rental.rating}</span> <span>(${rental.reviews})</span>
       </li>
      </ul>
   </aside>
   `;
   return document.createRange().createContextualFragment(template).querySelector("aside");
 });
 return markup;
};
```

Displaying Rentals

Immediately after the createMarkup method we call the displayRentals function to add the elements to the DOM. Because the createMarkup() function returned an array of elements we will pass the array of elements to our display method.

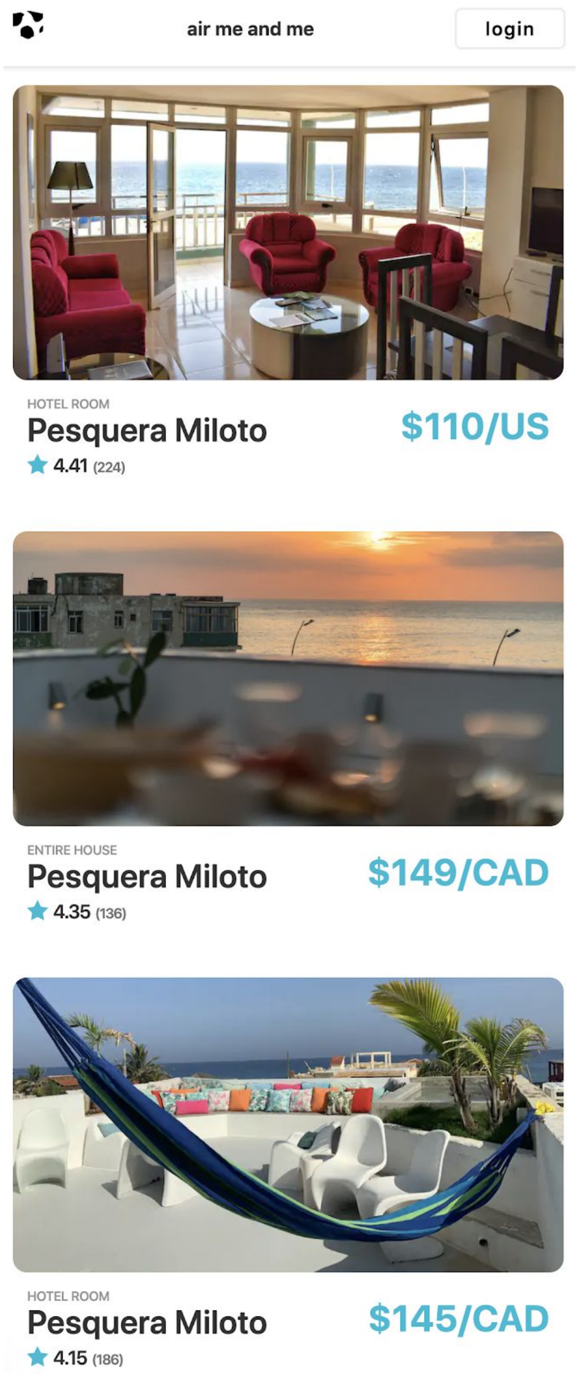
```
window.addEventListener("load", function (e) {
 const rentalRequest = fetch("./js/data.json");
 let store = [];
 rentalRequest
   .then((response) => response.json())
   .then((data) => {
     store = [...data];
     const rentals = createMarkup();
     displayRentals(rentals);
   })
   .catch((error) => console.warn(`Error: ${error}`));
});
```

The displayRentals() function takes a single parameter the array of elements to add to the display. A simple for each loop cycles through the elements in the array and adds each to the visual container in the DOM.

```
const displayRentals = function(elements){
   elements.forEach(function(rental){
       document.querySelector('.rentals').appendChild(rental)
   })
}
```

Checking The Display
When you have completed your code check the browser to see the layout of all 16 rentals. Your project should look similar to the image below.

Client Side JavaScript



HOTEL ROOM
Pesquera Miloto
★ 4.41 (224)
$110/US

ENTIRE HOUSE
Pesquera Miloto
★ 4.35 (136)
$149/CAD

HOTEL ROOM
Pesquera Miloto
★ 4.15 (186)
$145/CAD