Implementing the API Integration:

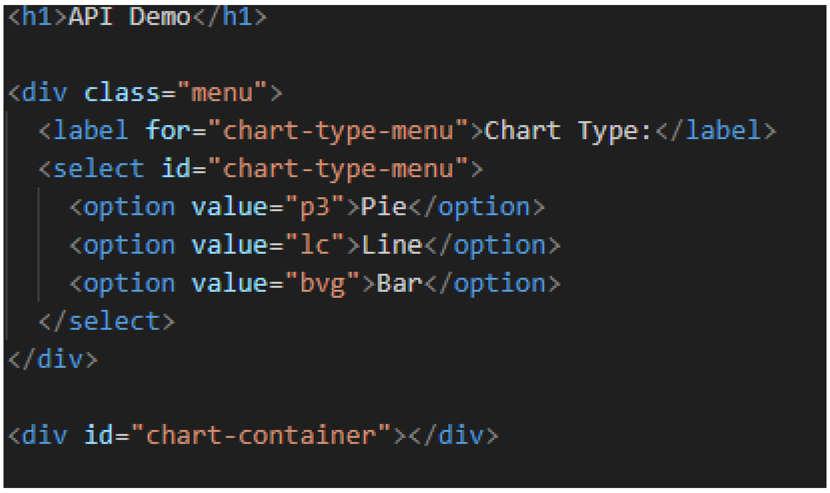
1 Adding HTML Structure:

In index.html in VS Code.

I added the HTML boilerplate code to set up the basic structure of your web application.

Inside the <body> element, I created a container <div> with an id for the chart. <div id="chart-container"></div>

I also added a div containing a chart-type selection which shows different chart menu selections



```html
<h1>API Demo</h1>

<div class="menu">
  <label for="chart-type-menu">Chart Type:</label>
  <select id="chart-type-menu">
    <option value="p3">Pie</option>
    <option value="lc">Line</option>
    <option value="bvg">Bar</option>
  </select>
</div>

<div id="chart-container"></div>
```

2 Styling the Web Application:

Open styles.css in VS Code.

I customized the CSS to style the web application.

3 Fetching and Displaying the Chart:

In script.js in VS Code.

Inside the DOMContentLoaded event listener, I used the fetch() function to make a request to the Image-Charts API URL.

I handled the response by converting it to a blob using the .blob() method.

I created an image element dynamically using JavaScript, set its source as the URL of the image blob, and append it to the chart container in the HTML page.

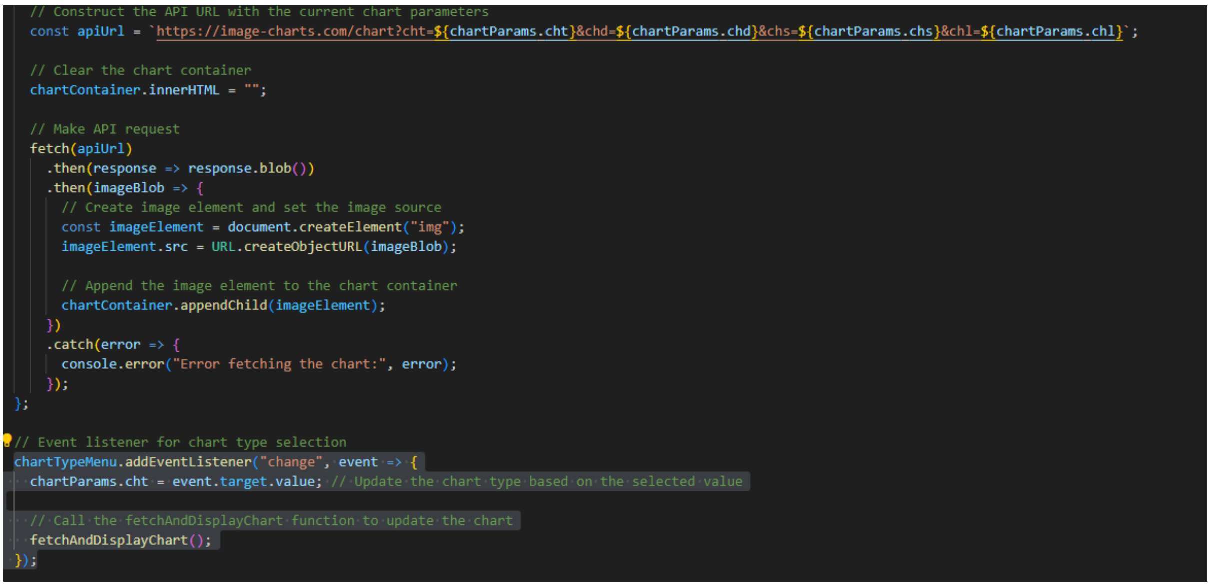
I also set up an event listener for a change event on an HTML element referred to as **chartTypeMenu**. When the value of the **chartTypeMenu** changes, the code inside the event listener function will be executed.

The event listener function takes an **event** object as a parameter. The **event** object contains information about the event that triggered the listener, including the **target** property, which represents the HTML element that triggered the event (in this case, the **chartTypeMenu** element).

Inside the event listener function, the code updates the **cht** property of an object called **chartParams**. It assigns the selected value of the **chartTypeMenu** element, which is accessed through **event.target.value**, to the **cht** property. This allows for dynamically updating the chart type based on the selected value.

After updating the **chartParams** object, the code calls a function named **fetchAndDisplayChart()**. This function is responsible for fetching data and displaying the chart based on the updated parameters.

In summary, this code sets up an event listener that tracks changes in a chart type menu. When the menu value changes, it updates the chart type parameter and triggers the function responsible for fetching and displaying the corresponding chart.



```
  // Construct the API URL with the current chart parameters
  const apiUrl = `https://image-charts.com/chart?cht=${chartParams.cht}&chd=${chartParams.chd}&chs=${chartParams.chs}&chl=${chartParams.chl}`;

  // Clear the chart container
  chartContainer.innerHTML = "";

  // Make API request
  fetch(apiUrl)
    .then(response => response.blob())
    .then(imageBlob => {
      // Create image element and set the image source
      const imageElement = document.createElement("img");
      imageElement.src = URL.createObjectURL(imageBlob);

      // Append the image element to the chart container
      chartContainer.appendChild(imageElement);
    })
    .catch(error => {
      console.error("Error fetching the chart:", error);
    });
};

// Event listener for chart type selection
chartTypeMenu.addEventListener("change", event => {
  chartParams.cht = event.target.value; // Update the chart type based on the selected value

  // Call the fetchAndDisplayChart function to update the chart
  fetchAndDisplayChart();
});
```

I understand i was supposed to use this API in the laravel team project, but as a team we decided not to because it did not fit what we wanted.