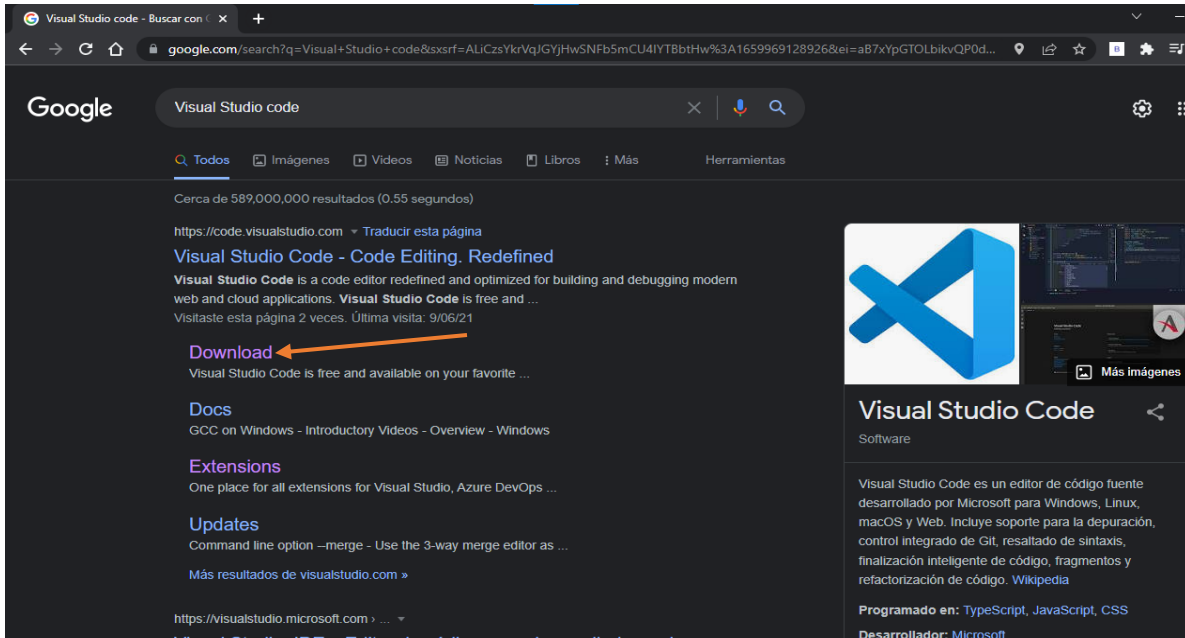
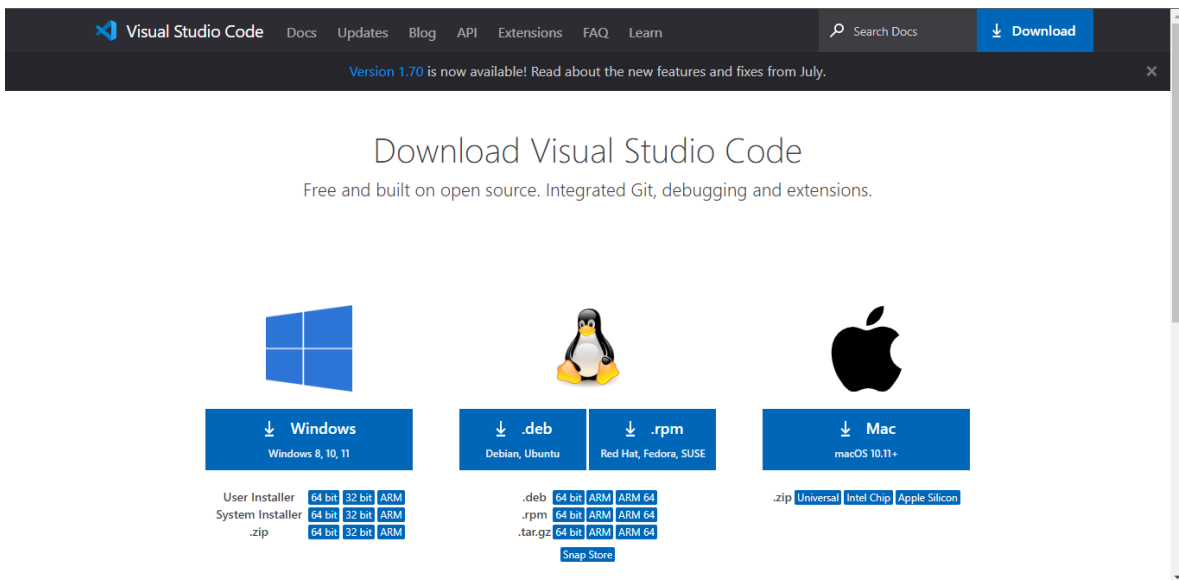


Proceso instalación Visual Studio Code

Paso 1: Escribimos en Google Visual Studio Code y seleccionamos donde dice “Download”.



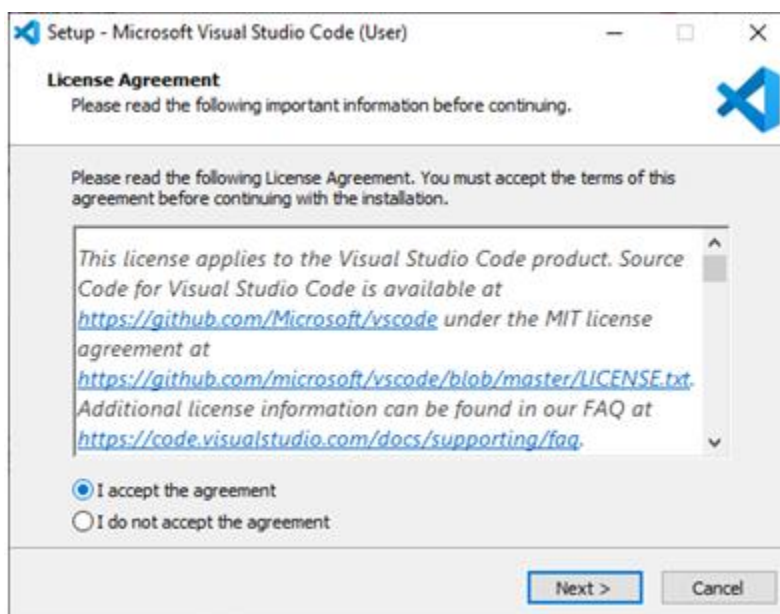
Paso 2: Seleccionamos el sistema operativo que tenemos y lo descargamos.



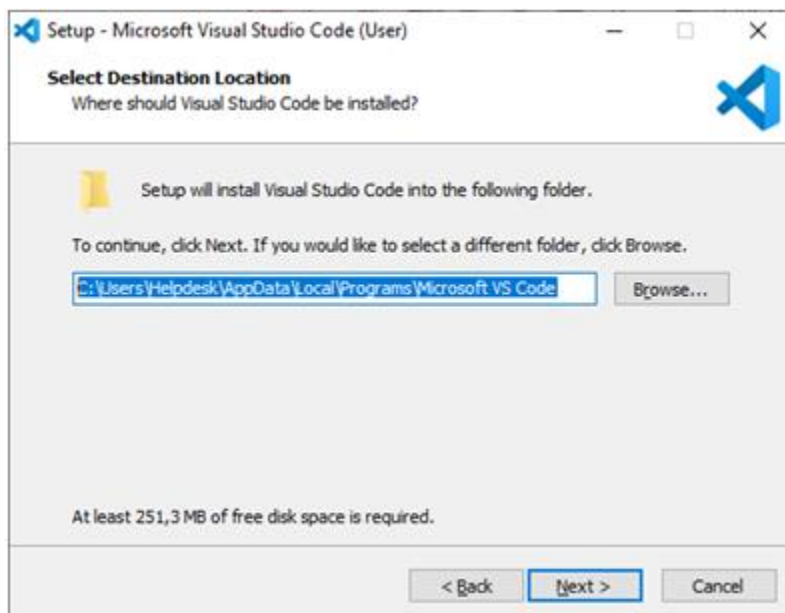
Paso 3: Al darle clic nos descargará un .exe, al cual le daremos clic encima.



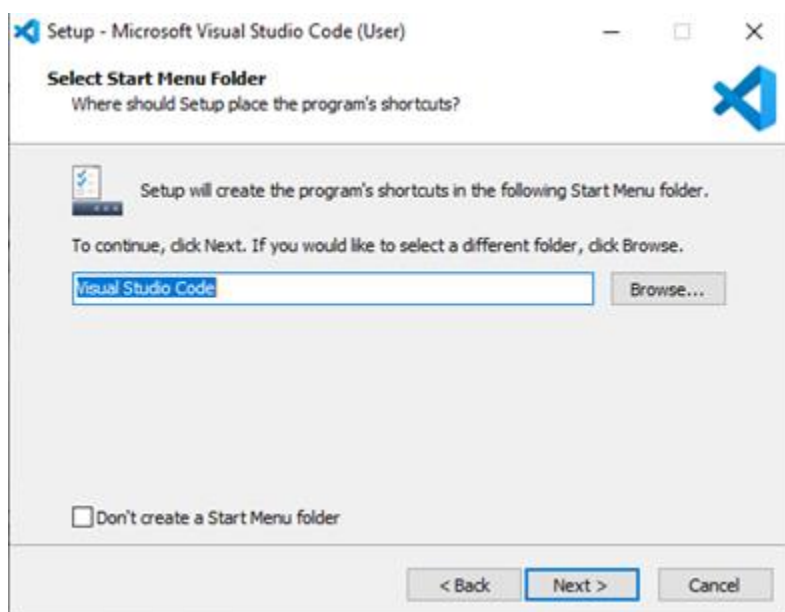
Paso 4: Lee y acepta el acuerdo de licencia. Haz clic en Next para continuar.



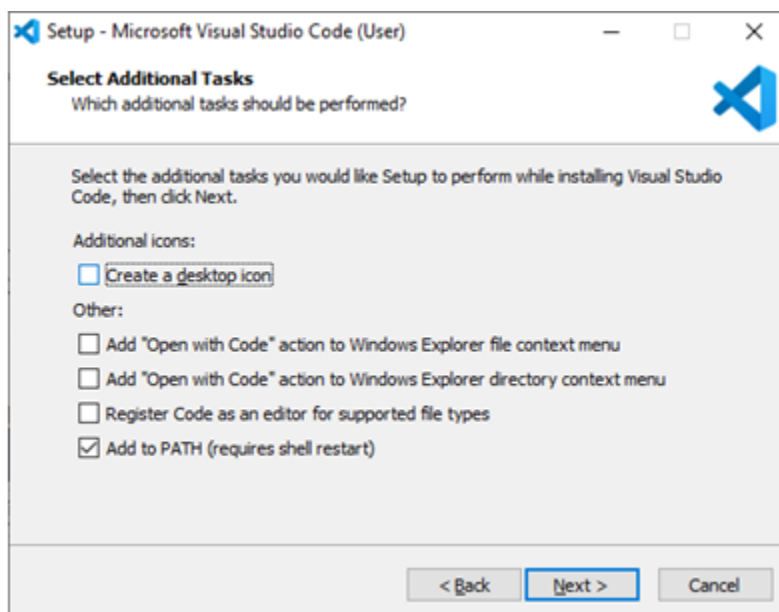
Paso 5: Puedes cambiar la ubicación de la carpeta de instalación o mantener la configuración predeterminada. Haz clic en Next para continuar.



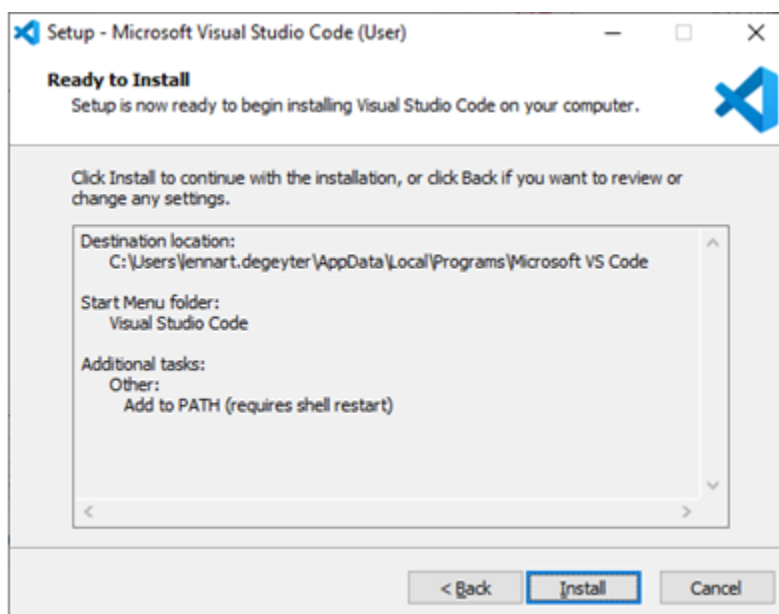
Paso 6: Elige si deseas cambiar el nombre de la carpeta de accesos directos en el menú Inicio o si no deseas instalar accesos directos en absoluto. Haz clic en Next.



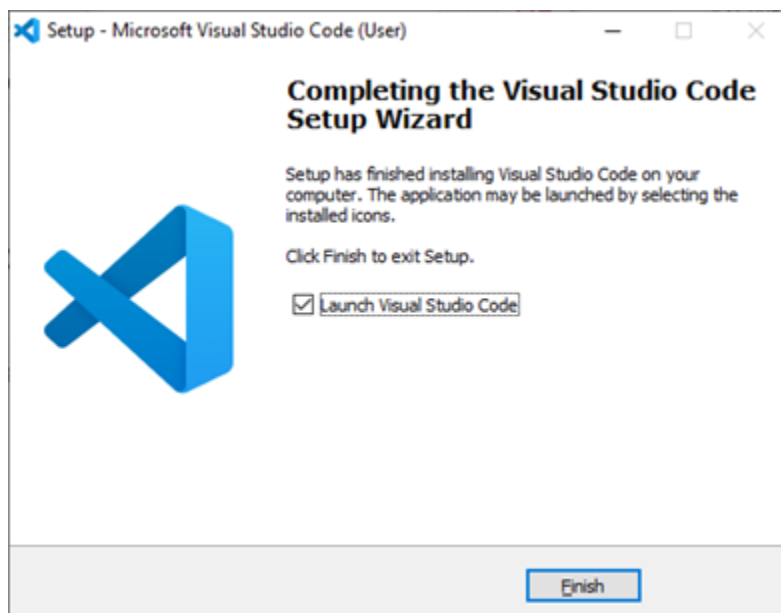
Paso 7: Selecciona las tareas adicionales, por ej. crear un icono en el escritorio o añadir opciones al menú contextual de Windows Explorer. Haz clic en Next.



Paso 8: Haz clic en Install para iniciar la instalación.



Paso 9: El programa está instalado y listo para usar. Haz clic en Finish para finalizar la instalación y lanzar el programa.





Creación de archivos y carpetas

Paso 1: Creamos una carpeta raíz. En nuestro caso la llamaremos “carritocompra”.

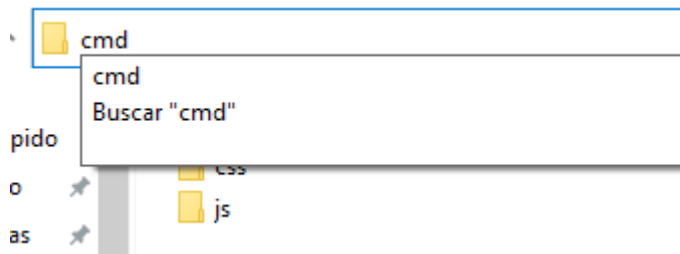
Nombre	Fecha de modificación	Tipo
 carritocompra	10/09/2023 1:25 a. m.	Carpeta de archivos

Paso 2: Dentro de esta, crearemos otras dos carpetas llamadas “css”, la cual contendrá todo lo estético de nuestra página y otra llamada “js” la cual contendrá la lógica de nuestra página.

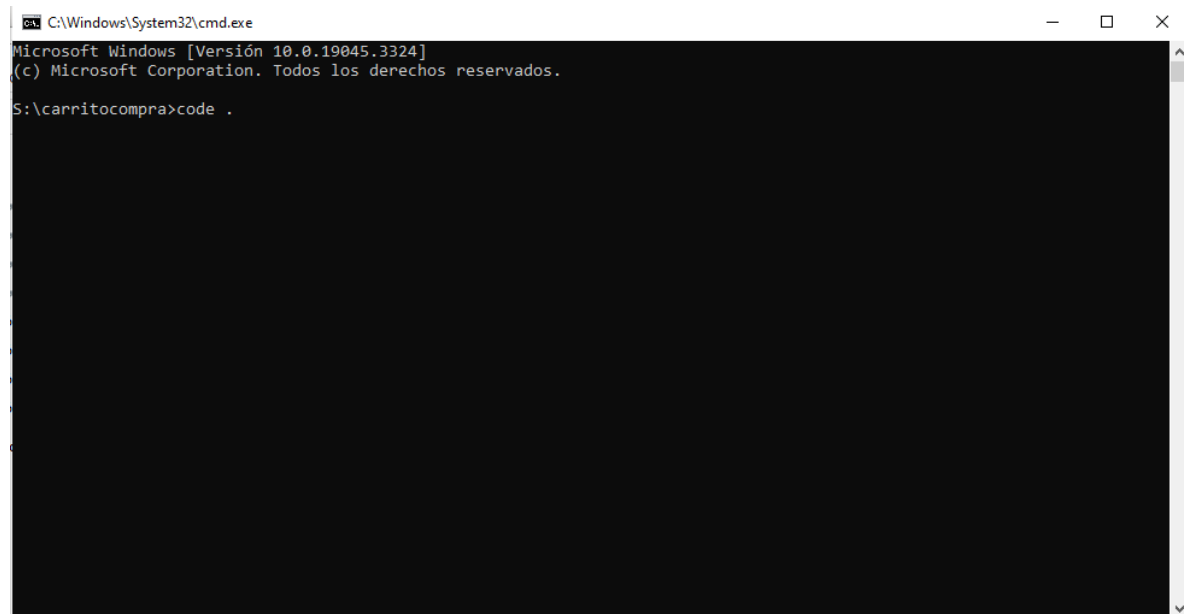
ste equipo > adso (S:) > carritocompra >					Buscar en carritocompra
Nombre	Fecha de modificación	Tipo	Tamaño		
 css	28/08/2023 1:35 a. m.	Carpeta de archivos			
 js	28/08/2023 1:32 a. m.	Carpeta de archivos			

Carrito de compras

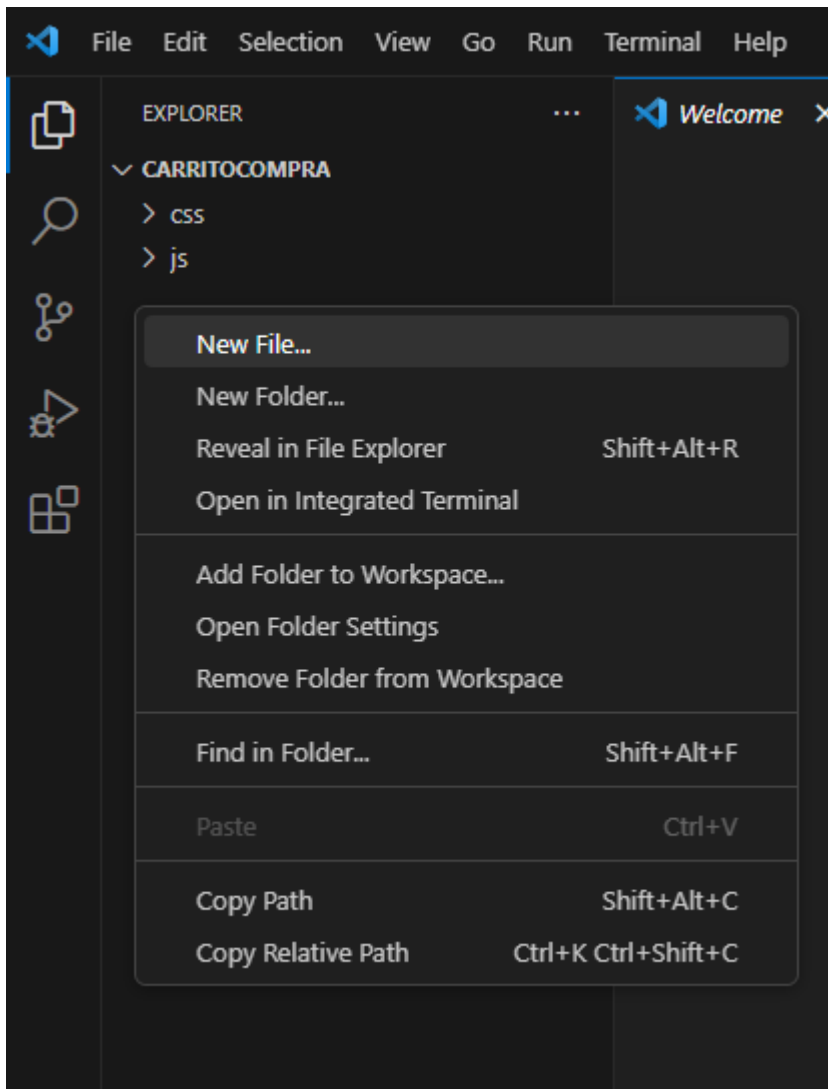
Paso 1: Para abrir el Visual Studio Code, haremos lo siguiente: Dentro de la carpeta raíz, en la barra superior, escribiremos cmd y le damos enter.



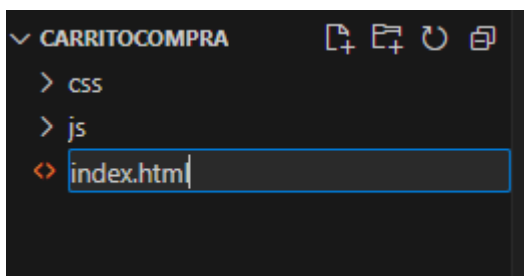
Paso 2: Eso nos abrirá una terminal, solo tendremos que escribir “code .”.



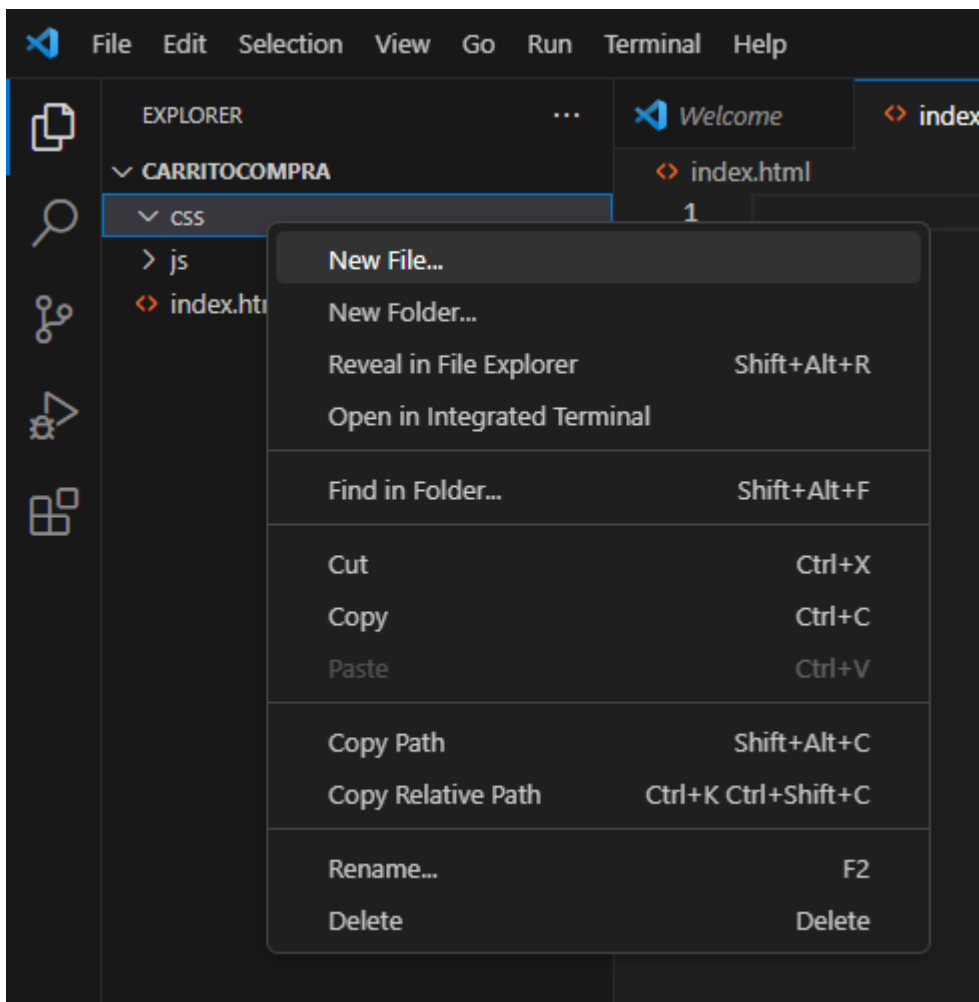
Paso 3: Esto nos abrirá el visual studio code. Ahora, ya que estamos dentro, crearemos un archivo llamado “index.html” dándole click derecho a la barra de la izquierda y luego click izquierdo en “new file” o “nuevo archivo”



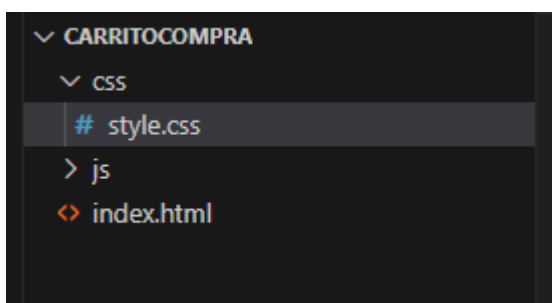
Paso 4: Llamaremos el archivo “index.html”



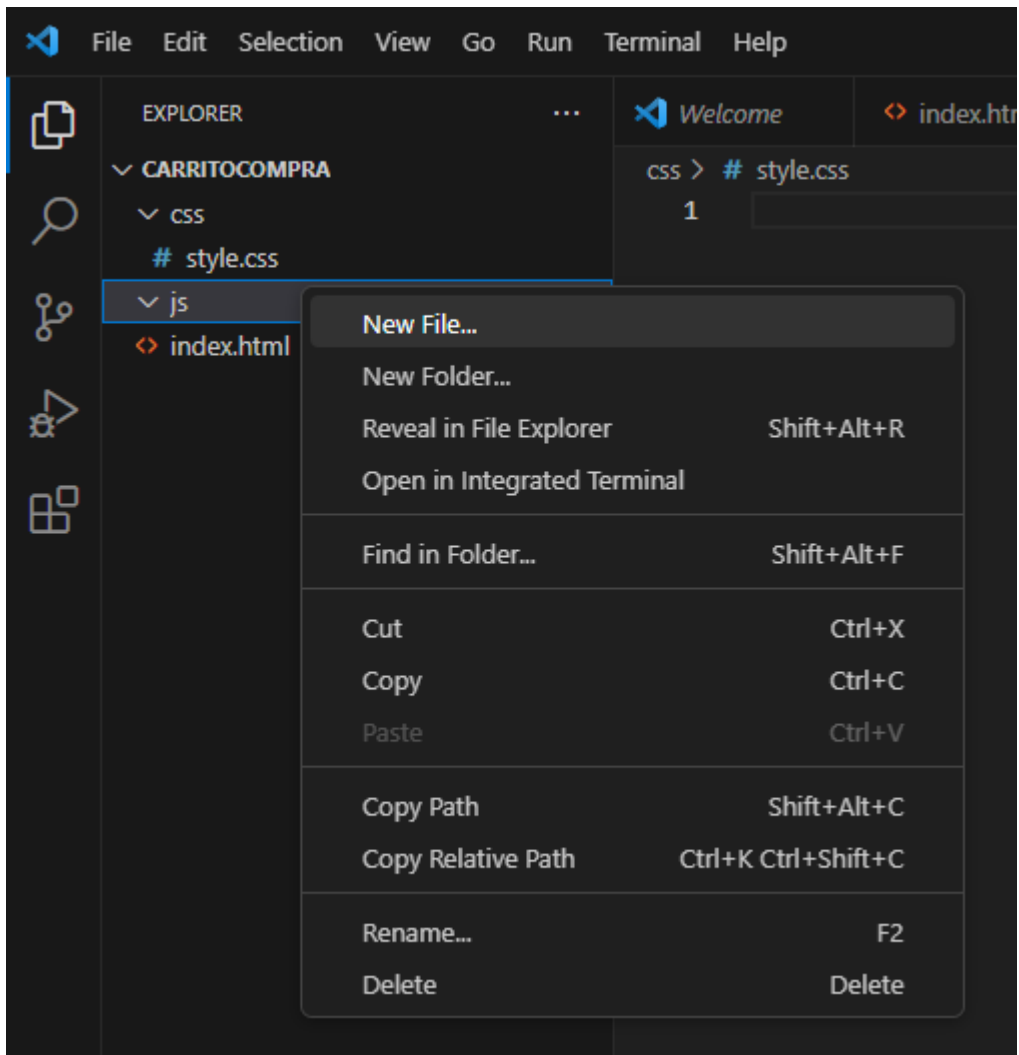
Paso 5: Ahora crearemos nuestro archivo de estilos dándole click derecho en nuestra carpeta llamada “css” y click izquierdo en “new file”



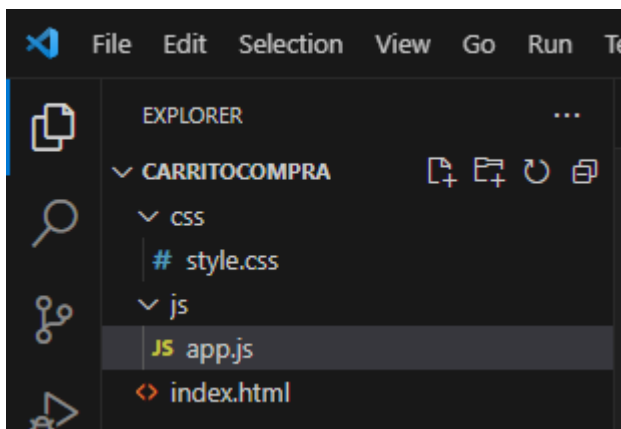
Paso 6: El archivo de llamara “style.css”



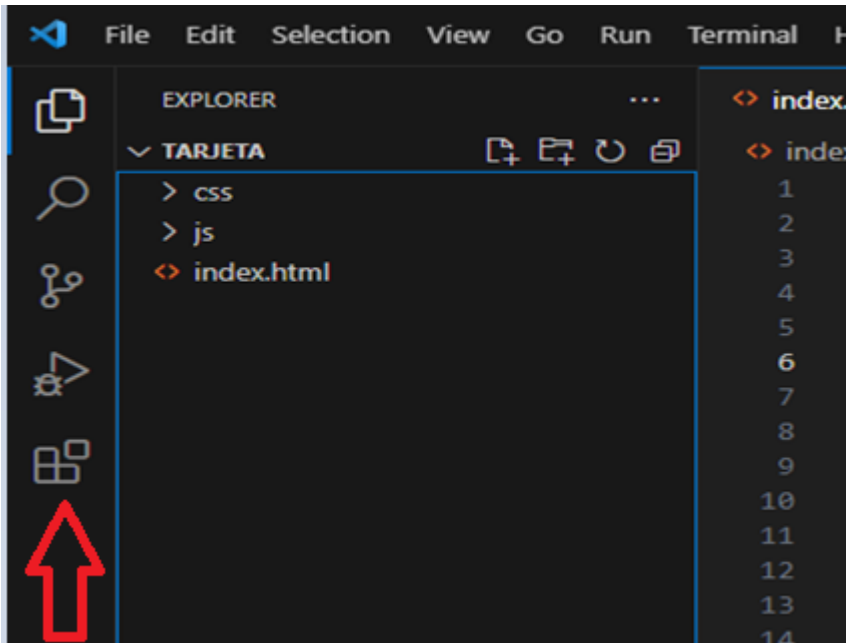
Paso 7: Ahora vamos a crear nuestro archivo js dándole click derecho en la carpeta “js” de la izquierda y luego click izquierdo en “new file”



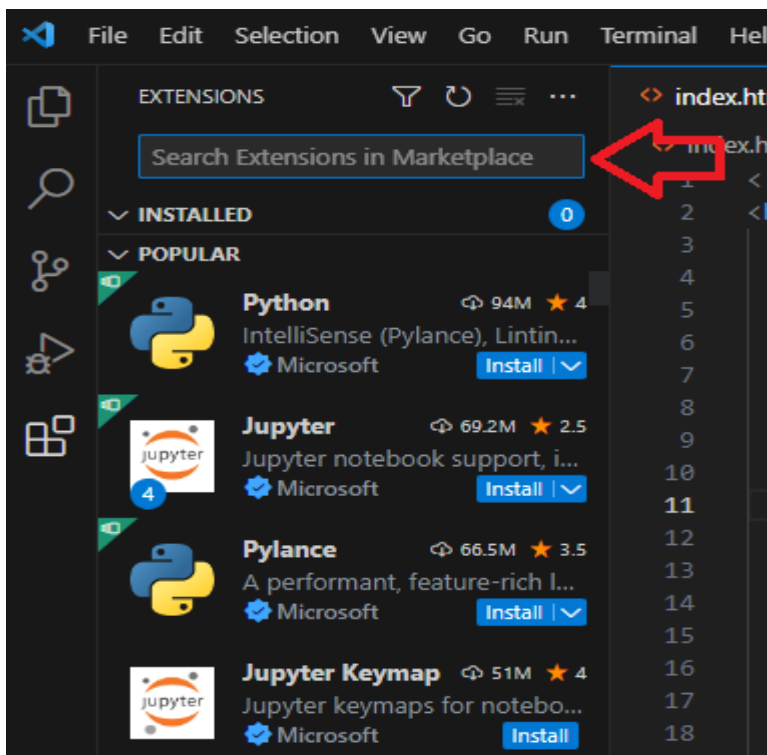
Paso 8: Llamamos el archivo como “app.js” y así tuvo que quedar nuestra estructuración de carpetas y archivos



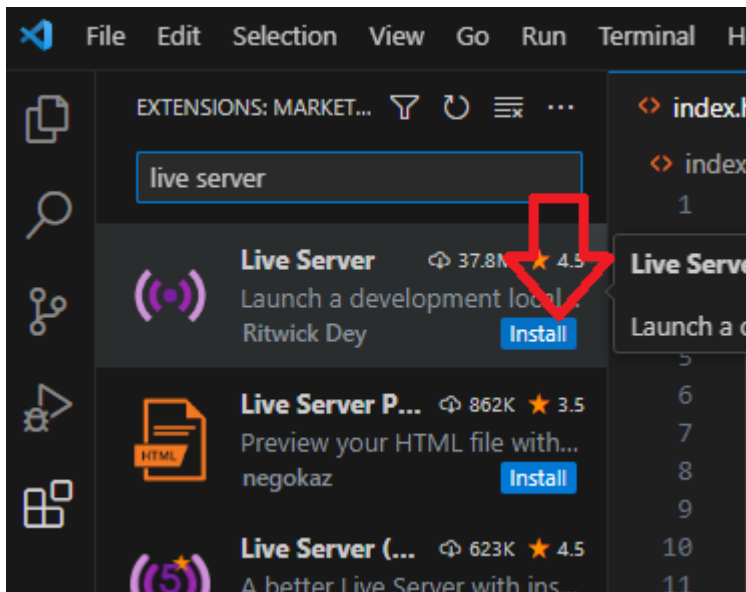
Paso 9: Ahora para poder visualizar como se está viendo nuestro proyecto instalaremos una extensión de visual estudio code, dándole click al icono de abajo a la izquierda.



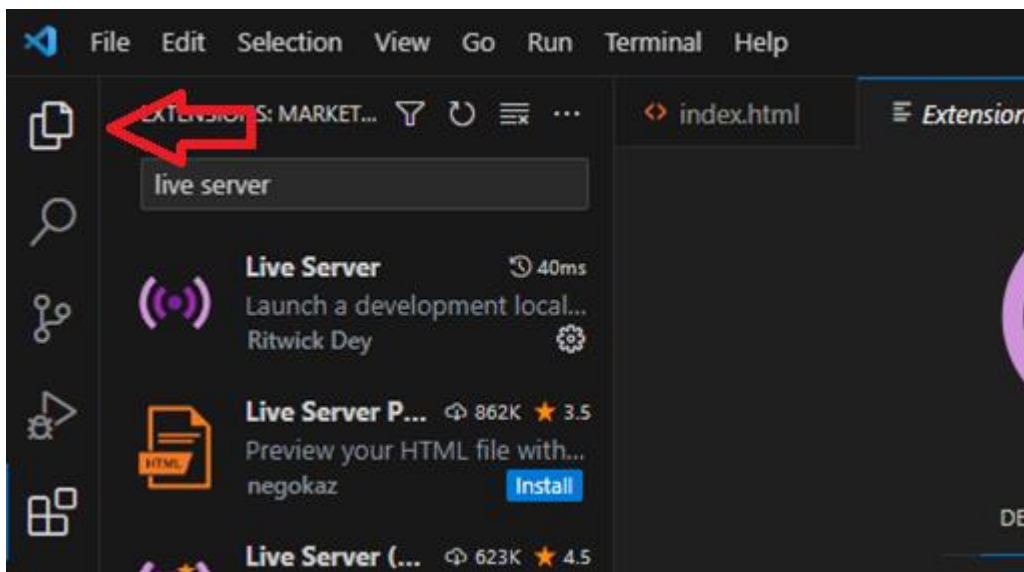
Paso 10: Nos saldrá esta pestaña y buscaremos live server, en la barra de búsqueda



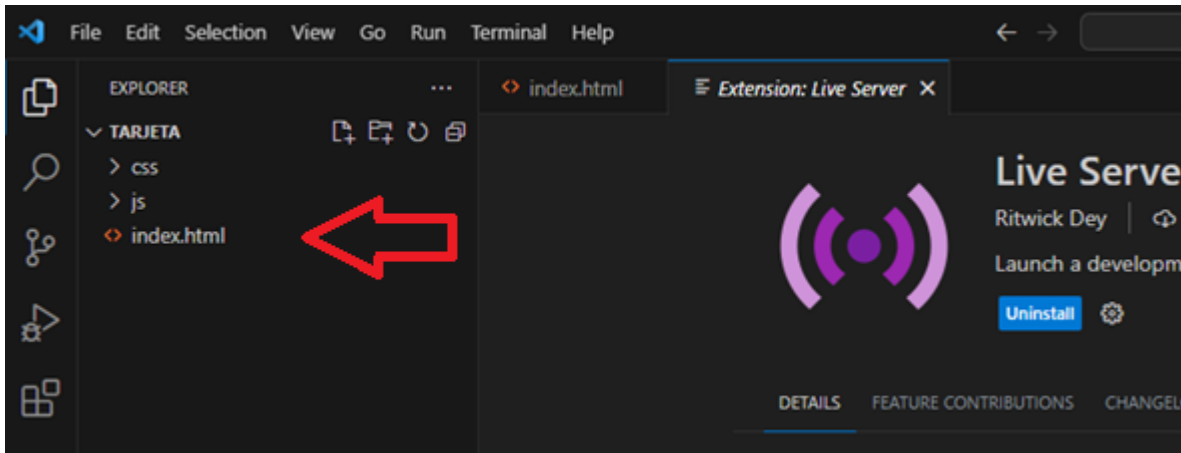
Paso 11: Ahora que encontramos lo que necesitamos, le damos click a “install”



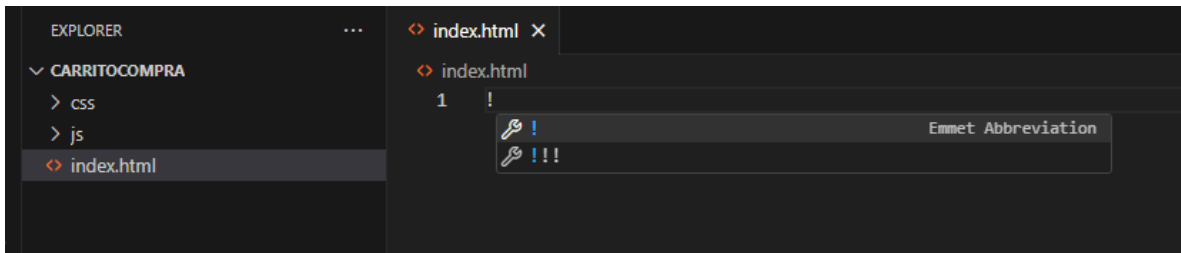
Paso 12: Ahora que tenemos instalado el live server nos debería de aparecer algo así donde podremos volver a nuestro proyecto dando click al explorador.



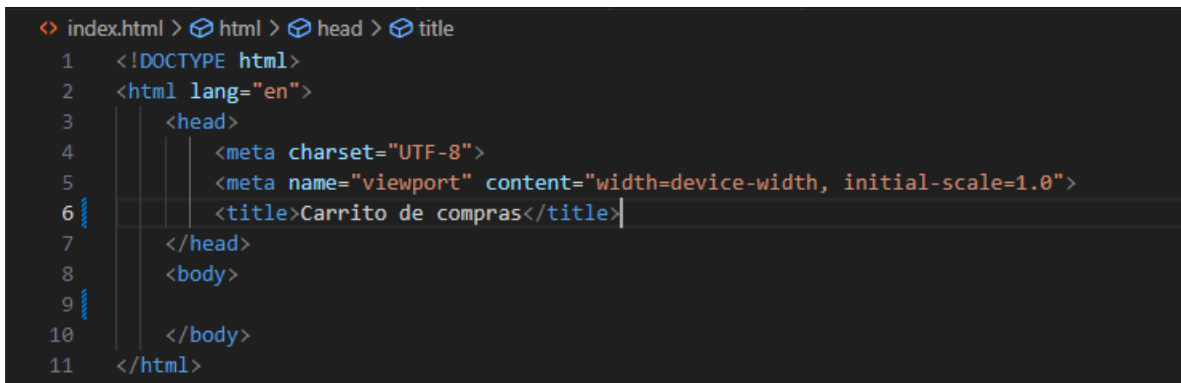
Paso 13: Una vez te sale el explorador volvemos al Index.html



Paso 14: Una vez estemos en el archivo index de nuevo vamos a colocar el código “!” y escogemos la primera opción.



Paso 15: Ahora vamos a cambiar el titulo (línea 6) por “Carrito de compras”



Paso 16: Vinculamos nuestro css dentro del head (línea 8) para dentro de un futuro modificar los diseños



Paso 17: Dentro del body crearemos un div principal con el id “app” (línea 12) dentro de ese div creamos 2 divs más un con la clase “menu” (línea 14) y otro con el id “store-container”(línea 19) y por último creamos otro div dentro del div con la clase “menu” (línea 16) con el id “shopping-cart-container” y le ponemos la clase “hide”

```
10      <body>
11      |   <!-- Div principal -->
12      |   <div id="app">
13      |       <!-- Div que contendrá el carro de compra-->
14      |       <div class="menu">
15      |           <!-- Div carro de compra -->
16      |           <div id="shopping-cart-container" class="hide"></div>
17      |       </div>
18      |       <!-- Elementos base de datos simulación -->
19      |       <div id="store-container"></div>
20      |   </div>
21      </body>
```

Paso 18: Ahora vinculamos nuestro javascript antes de terminar el body (línea 22)

```
20      </div>
21      <!-- JavaScript -->
22      <script src="js/app.js"></script>
23      </body>
```

Paso 19: Ahora iremos a nuestro archivo javascript “app.js” (Paso 8) y creamos un objeto para simular una base de datos llamado “db” (línea 2)

```
js > JS app.js > ...
1  // Definición de una base de datos de productos y sus métodos
2  const db = { //Inicio de base de datos
3  }; //Final de base de datos
4
```

Paso 20: Ahora crearemos un contenedor para almacenar los metodos (línea 3)

```
1  // Definición de una base de datos de productos y sus métodos
2  const db = { // inicio base de datos
3      methods: { //Inicio metodos
4
5      }, //Final métodos
6  }; // Final base de datos
7
```

Paso 21: Dentro de nuestro contenedor de métodos crearemos un método para encontrar un elemento por su id (línea 5)

```
3      methods: {} //Inicio metodos
4      // Método para encontrar un elemento por su id
5      find: (id) => {
6          return db.items.find((item) => item.id === id);
7      },
8
9  }, // Final metodos
```

Paso 22: Ahora creamos un nuevo método para remover items (línea 9), recorremos todos los items (línea 11) encuentra el producto correspondiente (línea 13) y reduce su cantidad (línea 15) y colocamos que se muestre actualizada en la consola (línea 19)

```
7      },
8      // Método para actualizar el inventario al remover items del carrito
9      remove: (items) => {
10         // Recorre la lista de items a remover
11         items.forEach((item) => {
12             // Encuentra el producto correspondiente en la base de datos
13             const product = db.methods.find(item.id);
14             // Reduce la cantidad de inventario del producto en la cantidad del item removido
15             product.qty = product.qty - item.qty;
16         });
17
18         // Muestra la base de datos actualizada en la consola
19         console.log(db);
20     },
21 }, // Final metodos
```

Paso 23: Ahora creamos un array con los items que va a simular la base de datos. (línea 23) vamos a colocarles un id diferente a cada uno, también el título, el precio de ese objeto y la cantidad que hay.

```
21 }, // Final metodos
22 // Lista de productos en la base de datos
23 items: [
24   {
25     id: 0,
26     title: "Funko Pop",
27     price: 250,
28     qty: 5,
29   },
30   {
31     id: 1,
32     title: "Harry Potter DVD",
33     price: 345,
34     qty: 50,
35   },
36   {
37     id: 2,
38     title: "Playstation 5",
39     price: 1300,
40     qty: 80,
41   },
42 ],
43 }; // Final base de datos
```

Paso 24: Ahora crearemos un objeto que va a representar el carrito de compras llamado "shoppingCart" (línea 46)

```
43 }; // Final base de datos
44
45 // Objeto que representa el carrito de compras
46 const shoppingCart = { //Inicio shoppingCart
47
48 }; //Final shoppingCart
49
```


Paso 25: Dentro de “shoppingCart” vamos a crear un array con los items (línea 48) y comenzamos con los métodos (línea 49) primero vamos a añadir el método para agregar un item al carrito y lo llamaremos “add”, (línea 51) y adentro de este método vamos a mirar si ya esta presente el objeto en el carrito (línea 53) añadiendo un condicional(línea 54) y adentro otro condicional que mirara si hay suficientes items del objeto para añadirlo al carrito (línea 56) donde utilizaremos un método que aun no tenemos llamado “hasInventory” y en caso de que no haya en el inventario que mande un mensaje que no hay objetos (línea 61) y en caso de que no este el item en el carrito vamos a añadirlo (línea 65)

```
47 // Lista de items en el carrito
48 items: [],
49 methods: { // Inicio metodos
50 // Método para agregar elementos al carrito
51 add: (id, qty) => {
52 // Obtiene el item del carrito si ya está presente
53 const cartItem = shoppingCart.methods.get(id);
54 if (cartItem) {
55 // Verifica si hay suficiente inventario para agregar la cantidad especificada
56 if (shoppingCart.methods.hasInventory(id, qty + cartItem.qty)) {
57 // Incrementa la cantidad en el carrito
58 cartItem.qty++;
59 } else {
60 // Muestra una alerta si no hay suficiente inventario
61 alert("No hay más inventario");
62 }
63 } else {
64 // Agrega el item al carrito si no está presente
65 shoppingCart.items.push({ id, qty });
66 }
67 },
68 }, //Final metodos
69 }; //Final shoppingCart
```

Paso 26: Ahora vamos a crear un nuevo método el cual va a remover elementos del carrito (línea 69) lo llamaremos “remove” lo primero será obtener el id del carrito (línea 71) en caso de que la cantidad sea mayor a 1 lo resta (línea 74) y de lo contrario lo quita (línea 78)

```
68 // Método para remover elementos del carrito
69 remove: (id, qty) => {
70 // Obtiene el item del carrito
71 const cartItem = shoppingCart.methods.get(id);
72
73 // Verifica si la cantidad es mayor que 1 para decrementarla, sino, remueve el item del carrito
74 if (cartItem.qty - 1 > 0) {
75 cartItem.qty--;
76 } else {
77 // Filtra la lista de items para remover el item con el id especificado
78 shoppingCart.items = shoppingCart.items.filter(
79 (item) => item.id !== id
80 );
81 }
82 },
83 }, //Final metodos
```

Paso 27: Ahora un nuevo método para contar la cantidad de objetos que hay (línea 84), la llamaremos “count”

```
82     },  
83     // Método para contar la cantidad total de items en el carrito  
84     count: () => {  
85         // Reduce la lista de items en el carrito para obtener la cantidad total  
86         return shoppingCart.items.reduce((acc, item) => acc + item.qty, 0);  
87     },  
88     //Final metodos
```

Paso 28: Ahora crearemos otro método llamado “get” (línea 89) para obtener un objeto del carrito (paso 89) donde se utilizara el código “.findIndex” (línea 91) para encontrar el objeto y en caso de que este en el carro lo retorna de lo contrario solo envía nulo.

```
88     // Método para obtener un item específico del carrito  
89     get: (id) => {  
90         // Encuentra el índice del item en la lista del carrito  
91         const index = shoppingCart.items.findIndex((item) => item.id === id);  
92         // Retorna el item si se encuentra en el carrito, de lo contrario, retorna null  
93         return index >= 0 ? shoppingCart.items[index] : null;  
94     },  
95     //Final metodos
```

Paso 29: Este método llamado “getTotal” (línea 96) funcionara para calcular el total de la compra con una variable que se llamara “total” (línea 97) y haremos que total sea el precio multiplicado por la cantidad en el carro y retorne el total (línea 103)

```
95     // Método para calcular el total de la compra  
96     getTotal: () => {  
97         let total = 0;  
98         // Calcula el total multiplicando el precio de cada item por su cantidad en el carrito  
99         shoppingCart.items.forEach((item) => {  
100             const found = db.methods.find(item.id);  
101             total += found.price * item.qty;  
102         });  
103         return total;  
104     },  
105     //Final metodos
```

Paso 30: Crearemos un nuevo método para saber si hay suficiente inventario para añadir al carro y lo llamaremos “hasInventory” (línea 106) y encontrara el objeto en la base de datos y mirara su cantidad, en caso de que sea mayor o igual a 0 al restar significa que hay inventario. (línea 108)

```
105     // Método para verificar si hay suficiente inventario para agregar al carrito  
106     hasInventory: (id, qty) => {  
107         // Encuentra el item en la base de datos y verifica si la cantidad en el inventario es suficiente  
108         return db.items.find((item) => item.id === id).qty - qty >= 0;  
109     },  
110     //Final metodos
```

Paso 31: Y creamos el ultimo método llamado “purchase” y a su vez cerramos “shoppingCart” el cual habíamos comenzado en la línea 46 (paso 24)

```
110 // Método para realizar la compra y actualizar el inventario
111 purchase: () => {
112   // Llama al método "remove" de la base de datos para actualizar el inventario
113   db.methods.remove(shoppingCart.items);
114 },
115 }, //Final metodos
116 }; //Final shoppingCart
```

Paso 32: Llamamos una función llamada “renderStore” (línea 119) y la creamos abajo (línea 122)

```
116 }; //Final shoppingCart
117
118 // Renderiza la tienda
119 renderStore();
120
121 // Función para renderizar la tienda
122 function renderStore() { //Inicio función
123
124 } //Final función
```

Paso 33: Ahora dentro de la función llamada “renderStore” que servirá para que se muestre nuestros objetos en la página vamos a generar elementos HTML por cada producto que hemos creado anteriormente con la función .map (línea 124) y luego vamos a generar el html (línea 126 a la 132)

```
121 // Función para renderizar la tienda
122 function renderStore() { //Inicio función
123   // Genera elementos HTML para cada producto en la base de datos
124   const html = db.items.map((item) => {
125     return `
126       <div class="item">
127         <div class="title">${item.title}</div>
128         <div class="price">${numberToCurrency(item.price)}</div>
129         <div class="qty">${item.qty} Unidades</div>
130         <div class="actions"><button class="add" data-id="${item.id}
131       >Añadir al carro de compras</button></div>
132       </div>`;
133   });
134
135 } //Final función
```

Paso 34: Dentro de la función “renderStore” vamos a agregar los elementos al contenedor de la tienda (línea 136)

```
133     });  
134  
135     // Agrega los elementos HTML generados al contenedor de la tienda  
136     document.querySelector("#store-container").innerHTML = html.join("");  
137  
138 } //Final función
```

Paso 35: Dentro de la misma función “renderStore” vamos a agregar un evento click a los botones de agregar al carrito (línea 139)

```
137  
138     // Agrega un evento de clic a los botones de agregar al carrito  
139     document.querySelectorAll(".item .actions .add").forEach((button) => {  
140         button.addEventListener("click", (e) => { //Inicio evento  
141             |  
142         }); //Final evento  
143     });  
144 } //Final función
```

Paso 36: Ahora dentro del evento vamos a obtener el id (línea 142) y encontrar el producto (línea 144) va a verificar si hay inventario y lo agrega al carrito (línea 147) y si lo hay actualiza el carrito con una función que crearemos ahora llamada “renderShoppingCart” (línea 150) en caso de que no haya va a enviar una alerta avisando que no hay existencias (línea 153)

```
138     // Agrega un evento de clic a los botones de agregar al carrito  
139     document.querySelectorAll(".item .actions .add").forEach((button) => {  
140         button.addEventListener("click", (e) => { //Inicio evento  
141             // Obtiene el id del producto a partir del atributo data-id del botón  
142             const id = parseInt(button.getAttribute("data-id"));  
143             // Encuentra el producto en la base de datos  
144             const item = db.methods.find(id);  
145  
146             // Verifica si hay inventario suficiente y agrega al carrito  
147             if (item && item.qty - 1 > 0) {  
148                 shoppingCart.methods.add(id, 1);  
149                 console.log(db, shoppingCart);  
150                 renderShoppingCart();  
151             } else {  
152                 // Muestra una alerta si no hay suficiente inventario  
153                 alert("Ya no hay existencia de ese artículo");  
154             }  
155         }); //Final evento  
156     });  
157 } //Final función
```

Paso 37: Ahora vamos a crear la función llamada “renderShoppingCart” que mencionamos anteriormente (línea 160)

```
157 } //Final función
158
159 // Función para renderizar el carrito de compras
160 function renderShoppingCart() { //Inicio función
161
162 } //Fin función
```

Paso 38: En la función “renderShoppingCart” vamos a generar elementos html con los objetos que se añadan

```
160 function renderShoppingCart() { //Inicio función
161     // Genera elementos HTML para cada item en el carrito de compras
162     const html = shoppingCart.items.map((item) => {
163         const dbItem = db.methods.find(item.id);
164         return `
165             <div class="item">
166                 <div class="title">${dbItem.title}</div>
167                 <div class="price">${numberToCurrency(dbItem.price)}</div>
168                 <div class="qty">${item.qty} Unidades</div>
169                 <div class="subtotal">Subtotal: ${numberToCurrency(
170                     item.qty * dbItem.price
171                 )}</div>
172                 <div class="actions">
173                     <button class="addOne" data-id="${dbItem.id}">+</button>
174                     <button class="removeOne" data-id="${dbItem.id}">-</button>
175                 </div>
176             </div>
177         `;
178     });
179
180 } //Fin función
```

Paso 39: Ahora creamos unos elementos html para cerrar el carrito (línea 180) y otra para terminar la compra (línea 184) y una nueva línea que mostrara el total del precio (línea 191)

```
179 // Genera el contenido para el cierre del carrito y el botón de compra
180 const closeButton = `
181   <div class="cart-header">
182     <button id="bClose">Cerrar</button>
183   </div>`;
184 const purchaseButton =
185   shoppingCart.items.length > 0
186   ? `<div class="cart-actions">
187     <button id="bPurchase">Terminar compra</button>
188   </div>`
189   : "";
190 const total = shoppingCart.methods.getTotal();
191 const totalDiv = `<div class="total">Total: ${numberToCurrency(total)}</div>`;
192
193
194 } //Fin función
```

Paso 40: Ahora debajo de lo que muestra el total vamos a agregar lo elementos seleccionados al contenedor del carrito de compras (línea 194)

```
191 const totalDiv = `<div class="total">Total: ${numberToCurrency(total)}</div>`;
192
193 // Agrega los elementos HTML generados al contenedor del carrito de compras
194 document.querySelector("#shopping-cart-container").innerHTML =
195   closeButton + html.join("") + totalDiv + purchaseButton;
```

Paso 41: Ahora les añadimos una clase para mostrarlo y le quitamos la anterior (línea 198-199)

```
196
197 // Muestra el contenedor del carrito y oculta el botón de compra
198 document.querySelector("#shopping-cart-container").classList.remove("hide");
199 document.querySelector("#shopping-cart-container").classList.add("show");
200
```

Paso 42: Ahora agregamos los eventos para añadir items al carrito y actualizamos el carrito llamando la función “renderShoppingCart” (línea 206)

```
200
201 // Agrega eventos a los botones de agregar y remover items del carrito
202 document.querySelectorAll(".addOne").forEach((button) => {
203   button.addEventListener("click", (e) => {
204     const id = parseInt(button.getAttribute("data-id"));
205     shoppingCart.methods.add(id, 1);
206     renderShoppingCart();
207   });
208 });
```

Paso 43: Ahora agregamos el evento para remover los items (línea 210) y actualizamos el carrito de compras (línea 214)

```
209
210 document.querySelectorAll(".removeOne").forEach((button) => {
211     button.addEventListener("click", (e) => {
212         const id = parseInt(button.getAttribute("data-id"));
213         shoppingCart.methods.remove(id, 1);
214         renderShoppingCart();
215     });
216 });
```

Paso 44: Ahora vamos a añadir el evento para cerrar el carrito de compras

```
217
218 // Agrega evento para cerrar el carrito de compras
219 document.querySelector("#bClose").addEventListener("click", (e) => {
220     document.querySelector("#shopping-cart-container").classList.remove("show");
221     document.querySelector("#shopping-cart-container").classList.add("hide");
222 });
223
```

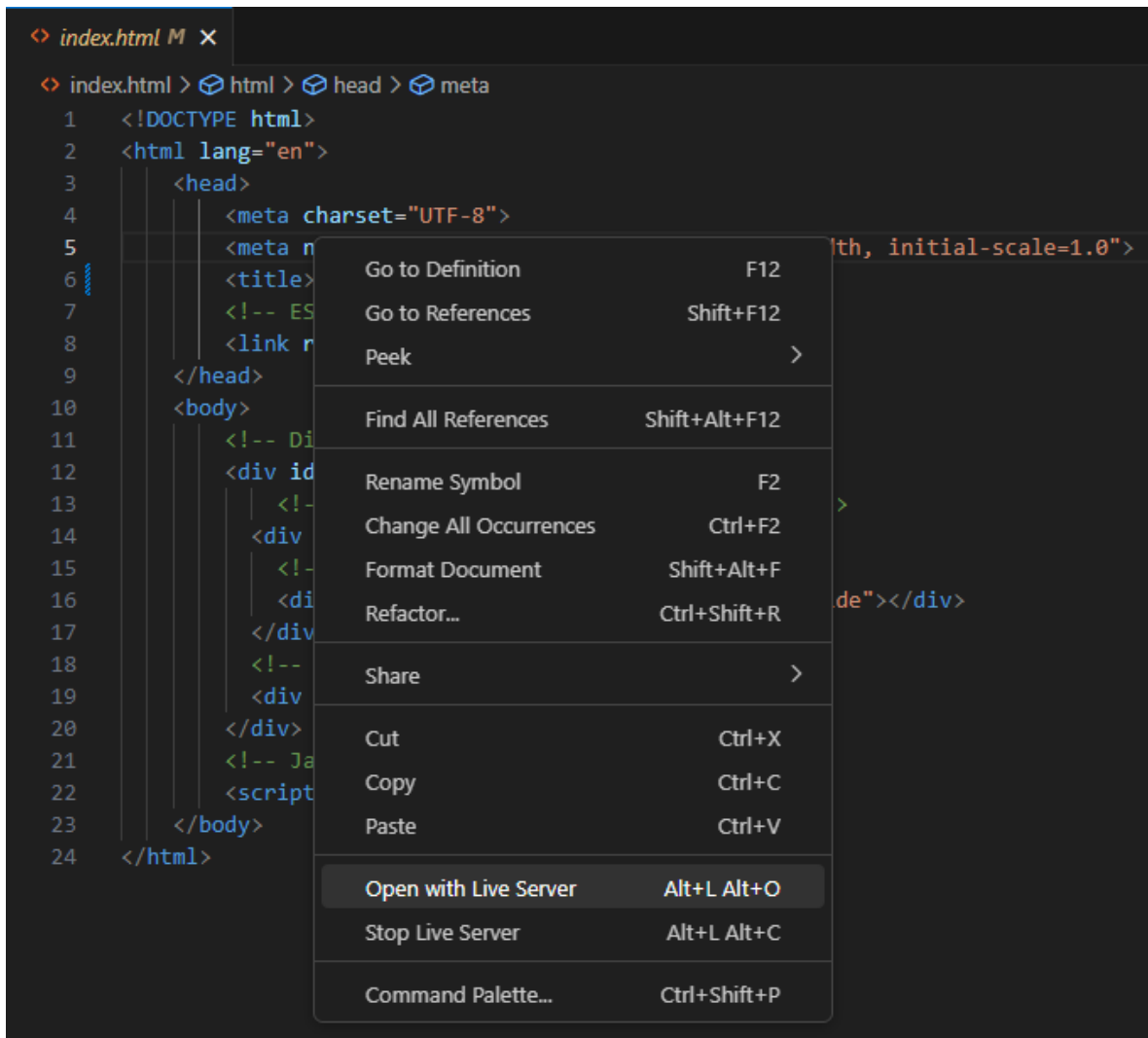
Paso 45: Y por último un evento para realizar la compra (línea 225) y que al apretar el botón active el evento de purchase "línea 228"

```
223
224 // Agrega evento para realizar la compra
225 const bPurchase = document.querySelector("#bPurchase");
226 if (bPurchase) {
227     bPurchase.addEventListener("click", (e) => {
228         shoppingCart.methods.purchase();
229     });
230 }
231 } //Fin función
```

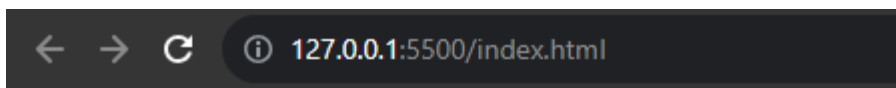
Paso 46: Y creamos una ultima función para convertir un número en formato de moneda.

```
232
233 // Función para convertir un número en formato de moneda
234 function numberToCurrency(n) {
235     return new Intl.NumberFormat("en-US", {
236         maximumSignificantDigits: 2,
237         style: "currency",
238         currency: "USD",
239     }).format(n);
240 }
```

Paso 47: Ahora vamos a mirar nuestro proyecto gracias a la extensión “live server” (paso 9 a 13) yendo a nuestro archivo “index.html” y daremos click derecho en cualquier parte del código luego click izquierdo en “open with live server”



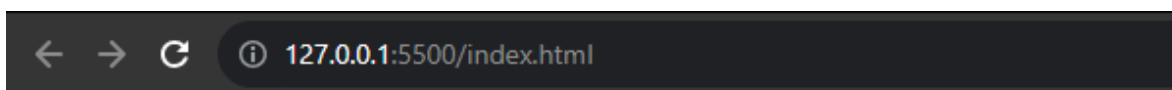
Paso 48: Gracias a esto se abrirá una pestaña en Google y te mostrará como va nuestro proyecto.



Funko Pop
\$250
5 Unidades

Harry Potter DVD
\$350
50 Unidades

Playstation 5
\$1,300
80 Unidades



Funko Pop
\$250
1 Unidades
Subtotal: \$250

Total: \$250

Funko Pop
\$250
5 Unidades

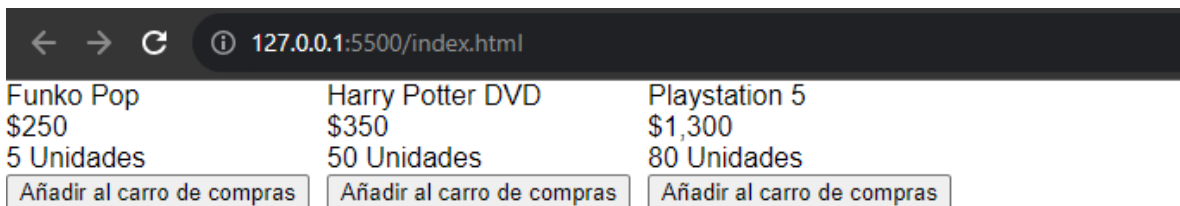
Harry Potter DVD
\$350
50 Unidades

Playstation 5
\$1,300
80 Unidades

Paso 49: Ahora para añadirle diseño a nuestra aplicación iremos a nuestro archivo "style.css" y cambiaremos un poco el body y el contenedor de la tienda.

```
# style.css M X
css > # style.css > #store-container
1  /* Estilos generales del cuerpo de la página */
2  body {
3      font-family: Arial, Helvetica, sans-serif; /* Fuente utilizada para el texto */
4      margin: 0; /* Margen exterior del cuerpo de la página */
5      padding: 0; /* Espacio interior del cuerpo de la página */
6  }
7
8  /* Estilos para el contenedor de la tienda */
9  #store-container {
10     display: flex; /* Se utiliza flexbox para organizar los elementos en fila */
11     gap: 10px; /* Espacio entre los elementos */
12 }
```

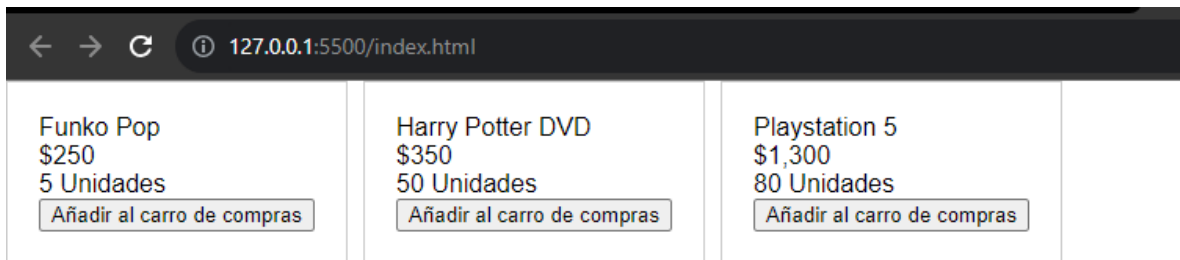
Paso 50: Ahora miramos como está quedando



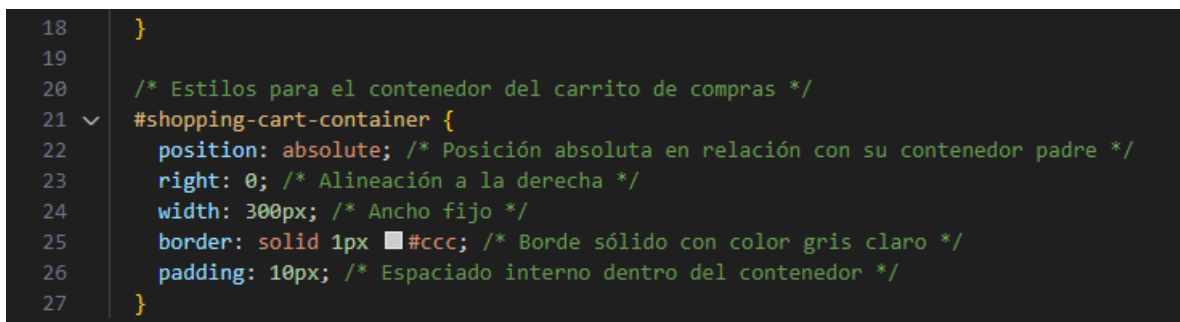
Paso 51: Cambiamos el diseño de los ítems

```
12
13
14  /* Estilos para cada ítem de la tienda */
15  #store-container .item {
16     border: solid 1px #ccc; /* Borde sólido con color gris claro */
17     padding: 20px; /* Espaciado interno dentro del ítem */
18 }
```

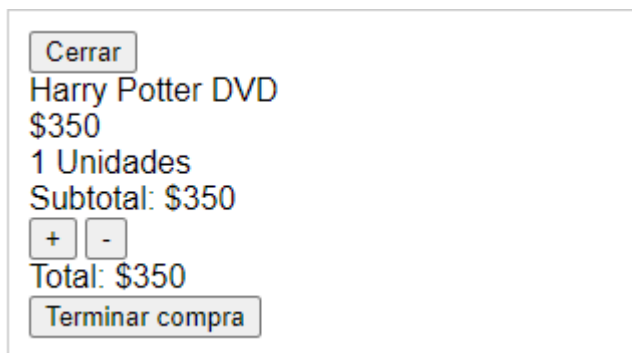
Paso 52: Así han quedado nuestros objetos por ahora



Paso 53: Ahora cambiamos un poco el diseño de nuestro carrito de compras.



Paso 54: Así va de momento



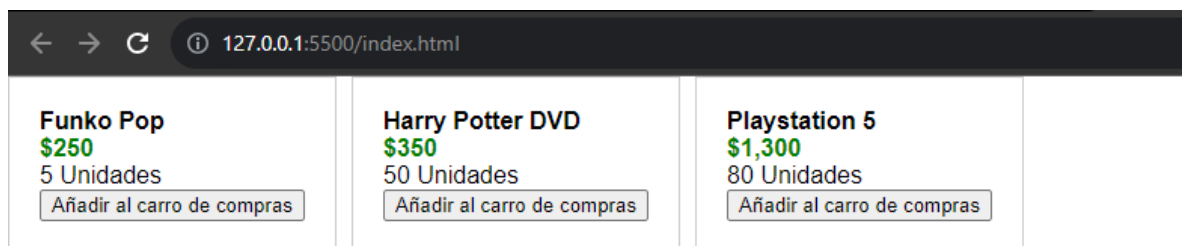
Paso 55: Ahora cambiamos la clase para que sea visible o no.

```
28
29  /* Clase para mostrar elementos */
30  .show {
31      display: block; /* Hace que el elemento sea visible */
32  }
33
34  /* Clase para ocultar elementos */
35  .hide {
36      display: none; /* Hace que el elemento sea invisible */
37  }
```

Paso 56: Ahora a nuestros items les modificamos un poco el diseño poniendo el titulo con negrilla y demás cosas.

```
38
39  /* Estilos para el título de cada ítem */
40  .item .title {
41      font-weight: bold; /* Texto en negrita */
42  }
43
44  /* Estilos para el precio de cada ítem */
45  .item .price {
46      color: green; /* Color del texto en verde */
47      font-weight: bold; /* Texto en negrita */
48  }
```

Paso 57: Así está quedando



Paso 58: Ahora cambiamos el diseño de los contenedores, el total y el botón.

```
49
50  /* Estilos para los contenedores de texto */
51  .item div {
52    padding: 5px 0; /* Espaciado interno vertical */
53  }
54
55  /* Estilos para el total en el carrito de compras */
56  .total {
57    padding: 10px 0; /* Espaciado interno vertical */
58    font-weight: bold; /* Texto en negrita */
59    font-size: 20px; /* Tamaño de fuente más grande */
60  }
61
62  /* Estilos para los botones */
63  button {
64    border: none; /* Sin borde */
65    background-color: rgb(241, 217, 28); /* Color de fondo amarillo */
66    padding: 5px 10px; /* Espaciado interno */
67    border-radius: 5px; /* Bordas redondeados */
68  }
```

Paso 59: Así ha quedado finalmente nuestro proyecto.

