Universidad de Nariño.
Ingeniería de Sistemas.
Diplomado de actualización en nuevas tecnologías para el desarrollo de Software.
Taller Unidad 3 Frontend

Santiago Alejandro Coral Ruano
217036022

Desarrollar una aplicación Frontend en Express que haga uso del Backend desarrollado en el taller anterior.

Para el desarrollo del Frontend, se tiene en cuenta el proyecto que veníamos trabajando desde la clase, en el cual se inició con la instalación de Angular Cli.

Para iniciar el nuevo proyecto se ejecutó el comando *ng new FruverFE* y cuando nos solicite aplicar routing digitamos *Yes*, además de elegir los estilos CSS para aplicar a nuestro proyecto. Aquí debemos esperar a que se instalen las dependencias del módulo 'node_modules'.

```
PS C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\Taller3 Frontend> ng new FruverFE

? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE FruverFE/angular.json (2711 bytes)
CREATE FruverFE/package.json (1040 bytes)
CREATE FruverFE/README.md (1062 bytes)
CREATE FruverFE/tsconfig.json (901 bytes)
CREATE FruverFE/.editorconfig (274 bytes)
CREATE FruverFE/.gitignore (548 bytes)
CREATE FruverFE/tsconfig.app.json (263 bytes)
CREATE FruverFE/tsconfig.spec.json (273 bytes)
CREATE FruverFE/.vscode/extensions.json (130 bytes)
CREATE FruverFE/.vscode/launch.json (470 bytes)
CREATE FruverFE/.vscode/tasks.json (938 bytes)
```

Cuando el proyecto se haya iniciado, procedemos a crear nuestros componentes. Inicialmente, creamos los componentes para listar y editar productos, haciendo uso del comando *ng generate component listar-productos*, el cual nos dará las opciones para administrar los productos de la base de datos.

```
$ ng generate component listar-productos
CREATE src/app/listar-productos/listar-productos.component.html (31 bytes)
CREATE src/app/listar-productos/listar-productos.component.spec.ts (623 bytes)
CREATE src/app/listar-productos/listar-productos.component.ts (241 bytes)
CREATE src/app/listar-productos/listar-productos.component.css (0 bytes)
UPDATE src/app/app.module.ts (513 bytes)
```

Hacemos lo mismo con el componente editar-productos, el cual nos servirá para creación y actualización de productos.

```
Santiago Coral@LAPTOP-3TH8BD4A MINGW64 ~/Documents/Diplomado/Modulo2 Diplomado
$ ng generate component editar-productos
CREATE src/app/editar-productos/editar-productos.component.html (31 bytes)
CREATE src/app/editar-productos/editar-productos.component.spec.ts (623 bytes)
CREATE src/app/editar-productos/editar-productos.component.ts (241 bytes)
CREATE src/app/editar-productos/editar-productos.component.css (0 bytes)
UPDATE src/app/app.module.ts (633 bytes)
```

Creamos un componente para mostrar el NavBar en la parte superior en todas las páginas:

```
PS C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\FruverSC\Frontend\FruverFE> ng generate component navbar
CREATE src/app/navbar/navbar.component.html (21 bytes)
CREATE src/app/navbar/navbar.component.spec.ts (559 bytes)
CREATE src/app/navbar/navbar.component.ts (202 bytes)
CREATE src/app/navbar/navbar.component.css (0 bytes)
UPDATE src/app/app.module.ts (1121 bytes)
PS C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\FruverSC\Frontend\FruverFE>
```

Creamos el componente mostrar-productos, el cual contendrá la pagina que visualizarán los usuarios.

```
PS C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\FruverSC\Frontend\FruverFE> ng generate component mostrar-productos
CREATE src/app/mostrar-productos/mostrar-productos.component.html (32 bytes)
CREATE src/app/mostrar-productos/mostrar-productos.component.spec.ts (630 bytes)
CREATE src/app/mostrar-productos/mostrar-productos.component.ts (245 bytes)
CREATE src/app/mostrar-productos/mostrar-productos.component.css (0 bytes)
UPDATE src/app/app.module.ts (1039 bytes)
PS C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\FruverSC\Frontend\FruverFE>
```

Generamos un componente comprar, el cual servirá para el formulario de registro de compra.

```
$ ng generate component comprar
CREATE src/app/comprar/comprar.component.html (22 bytes)
CREATE src/app/comprar/comprar.component.spec.ts (566 bytes)
CREATE src/app/comprar/comprar.component.ts (206 bytes)
CREATE src/app/comprar/comprar.component.css (0 bytes)
UPDATE src/app/app.module.ts (1207 bytes)
```

Rutas o **Routing**: Para establecer los enlaces y conexiones entre los componentes y vistas de la aplicación, nos dirigimos al archivo 'app-routing.module.ts' y agregamos las rutas necesarias para la app.

```
import { MostrarProductosComponent } from './mostrar-productos/mostrar-productos.component';

const routes: Routes = [
  {path:'mostrar_productos',component: MostrarProductosComponent},
  {path:'productos',component: ListarProductosComponent},
  {path:'productos/editar/:idProducto',component: EditarProductosComponent},
  {path:'productos/agregar',component: EditarProductosComponent},
  {path:'comprar',component: ComprarComponent},
  {path:'**',redirectTo:'/productos',pathMatch:'full'},
```

```
...
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Para aplicar el routing, hay que tener en cuenta que en el componente html principal (app-component.html) se debe remplazar el selector por <router-outlet>. Esta etiqueta permitirá el llamado a las diferentes rutas por su nombre.

```
<div>       You, hace 3 días • CRUD pro
  <!-- <h3>Fruver Productos</h3> -->
  <router-outlet></router-outlet>
</div>
```

Ahora, se procede a crear los modelos para productos, carritos, productos carritos y compras, por lo que creamos una carpeta denominada 'shared' la cual contendrá estos módulos, además de los servicios que se crearán más adelante.

Creamos el archivo 'producto.model.ts' en el cual definimos los atributos del modelo Producto:

```
export class ProductoModel {
    constructor(
        public idProducto: string,
        public nombre: string,
        public precio: number,
        public cantidad_producto: number,
        public descripcion: string,
        public categoria: string,
        public imagen:string
    )
    {
        // this.idProducto=idProducto;
    }
}
```

Creamos el archivo 'carrito.model.ts', en el que se definen los atributos para modelo Carrito:

```
export class CarritoModel {
    constructor(

        public idCarrito: string,
        public idUsuario: string,
        public valor_total: number,
        public fechaCreacion: string //
    ) {
        // this.idCarrito=idCarrito;
    }
}
```

Creamos el archivo ‘productos_carrito.model.ts’, el cual tendrá los atributos para el modelo Producto de Carrito:

```
import { ProductoModel } from "./producto.model";

...
export class ProductosCarritoModel {
    constructor(
        public idProductoCarrito: string,
        public CarritoId: string,
        public ProductoId: string,
        public cantidad: number,
        public valor_parcial: number,
        public producto: ProductoModel

    ) {
    }
}
```

Creamos el archivo ‘compras.model.ts’ en el que definimos los atributos para el proceso de compra:

```
export class CompraModel {
    constructor(

        public idCompra: string,
        public idCarritoCompra: string,
        public nombres: string,
        public correo: string,
        public direccion: string,
        public metodo_pago: string,
        public fechaCompra: string,
```

Luego de crear los modelos necesarios, se procede a crear los servicios que permitan hacer las peticiones HTTP hacia el backend, por lo que en la terminal ejecutamos el comando *ng generate service shared/producto* para guardar los métodos correspondientes al módulo Producto:

```
$ ng generate service shared/producto
CREATE src/app/shared/producto.service.spec.ts (367 bytes)
CREATE src/app/shared/producto.service.ts (137 bytes)
```

Crear servicio carrito en la carpeta ‘shared’ para establecer los métodos correspondientes al módulo Carrito

```
PS C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\FruverSC\Frontend\FruverFE> ng generate service shared/carrito
CREATE src/app/shared/carrito.service.spec.ts (362 bytes)
CREATE src/app/shared/carrito.service.ts (136 bytes)
PS C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\FruverSC\Frontend\FruverFE>
```

Crear servicio para agregar productos al carrito

```
PS C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\FruverSC\Frontend\FruverFE> ng generate service shared/productos-carrito
CREATE src/app/shared/productos-carrito.service.spec.ts (408 bytes)
CREATE src/app/shared/productos-carrito.service.ts (145 bytes)
PS C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\FruverSC\Frontend\FruverFE>
```

Y crear el servicio para el proceso de compra:

```
$ ng generate service shared/comprar
CREATE src/app/shared/comprar.service.spec.ts (362 bytes)
CREATE src/app/shared/comprar.service.ts (136 bytes)
```

En cada uno de los servicios se deben crear los métodos que permitan hacer las peticiones necesarias al backend, por ejemplo, en el servicio del módulo Producto se tiene:

```
export class ProductoService {
  BASE_URL = 'http://localhost:3000';
  constructor(private http: HttpClient) {

  }
  obtenerProductos(){
    return this.http.get<ProductoModel[]>(`${this.BASE_URL}/productos`);
  }
  obtenerProductosByCategoria(categoria:string){
    return this.http.get<ProductoModel[]>(`${this.BASE_URL}/productos_cat/${categoria}`);
  }
  obtenerProducto(idProducto:string){///Un solo producto
    return this.http.get<ProductoModel[]>(`${this.BASE_URL}/productos/${idProducto}`);
  }
  agregarProducto(producto:ProductoModel){
    return this.http.post<string>(`${this.BASE_URL}/productos`,producto);
  }
  actualizarProducto(producto: ProductoModel) {
    return this.http.put<string>(`${this.BASE_URL}/productos/${producto.idProducto}`,producto);
  }
  borrarProducto(idProducto: string) {
    return this.http.delete<string>(`${this.BASE_URL}/productos/${idProducto}`);
  }
```

Así mismo se han definido los diferentes métodos en cada servicio, teniendo en cuenta las rutas establecidas en el Backend.

Para ejecutar nuestro proyecto de Frontend ejecutamos el comando *ng serve* o *ng s*:

```
$ ng s -o
✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size
vendor.js             | vendor        |   2.57 MB |
polyfills.js          | polyfills     | 332.98 kB |
styles.css, styles.js | styles        | 230.55 kB |
main.js               | main          | 164.15 kB |
runtime.js            | runtime       |   6.51 kB |

                      | Initial Total |   3.28 MB

Build at: 2023-07-17T15:53:40.793Z - Hash: 17e9f80c6c8b33c0 - Time: 19805ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


✓ Compiled successfully.
```

Así podemos visualizar la app desde el enlace http://localhost:4200/ en el navegador.

Ahora se procede a estructurar las diferentes vistas de pagina en cada componente, por ejemplo, para el Nabvar que se visualiza en todos los componentes se implementa Bootstrap para diseñar la interfaz de usuario.

Listado de productos registrados

Para los enlaces del Nabvar se implementa la directiva **routerLink** para redireccionar al componente correspondiente:

```
<a class="navbar-brand btn " [routerLink]="['/mostrar_productos']"><img
      src="../../assets/2.png" alt="Bootstrap" width="200" height="75"></a>
```

Para el icono que aparece en el Navbar en el que se miran la cantidad de productos en carrito se creará una variable (total_en_carrito) dentro de la clase del componente typescript. En el componente html se agrega en la etiqueta :

```
<a class="carrito">
    <button class="btn btn-outline-light" type="button" data-bs-toggle="offcanvas"
        data-bs-target="#offcanvasRight" aria-controls="offcanvasRight" (click)
        ="obtenerProductosDeCarrito()">
        <span>Cart</span>
        <svg xmlns="http://www.w3.org/2000/svg" width="35" height="35" fill="currentColor"
            class="bi bi-cart-fill" viewBox="0 0 16 16">
        <span class="badge bg-dark text-white ms-1 rounded-pill">{{total_en_carrito}}</
        span>
    </button>
</a>
```

Y en el componente typescript se crea la variable:

```
total_en_carrito: number = 0;
```

Para calcular esa cantidad, se hace uso del método obtenerProductosByIDCarrito del servicio del carrito, por lo que lo definimos con una variable dentro del constructor de la clase. Así obtenemos los productos que se han guardado en el carrito y calculamos la cantidad mediante el método **length**.

```
constructor(private carritoService: CarritoService, private productosCarritoService:
ProductosCarritoService, private productoService: ProductoService) { }

ngOnInit() {
  const idCarritoSesion = sessionStorage.getItem("idCarritoS");
  console.log(idCarritoSesion);
  if (idCarritoSesion) {
    this.carrito = this.carritoService.obtenerCarritoByID(idCarritoSesion);
    this.carrito.subscribe(data => {
      this.carritoInfo = data[0];
    });

    this.productosEnCarrito = this.productosCarritoService.obtenerProductosByIDCarrito(idCarritoSesion);
    this.productosCarritoService.obtenerProductosByIDCarrito(idCarritoSesion).subscribe((productos) => {
      this.total_en_carrito = productos.length;
    });;

  }
}
```

Al definir nuestro Navbar lo podemos utilizar en cada uno de los componentes creados agregando en el inicio de los archivos html la etiqueta:

```
<app-navbar></app-navbar>
```

En el componente listar-productos html, se implementa una tabla en la que se mostrarán los detalles y botones para editar o eliminar cada producto:

```
<table class="table table-striped">
  <thead>
    <tr class="row">
      <th class="col-sm-2"> ID </th>
      <th class="col-sm-4"> NOMBRE </th>
      <th class="col-sm-4"> DETALLE </th>
      <th class="col-sm-2"> ACCIONES</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let producto of productos | async" class="row">
      <td class="col-sm-2"><i>{{ producto.idProducto }}</i></td>
      <td class="col-sm-4"><i>{{ producto.nombre }}</i></td>
      <td class="col-sm-4"><i>{{ producto.detalle }}</i></td>
      <td class="col-sm-1">
        <fieldset class="form-group col-sm-1">
          <a class="btn btn-info" [routerLink]="['/productos/editar/', producto.idProducto]">Editar</a>
        </fieldset>
      </td>
      <td class="col-sm-1">
        <fieldset class="form-group col-sm-1">
          <a class="btn btn-danger" (click)="borrarProducto(producto.idProducto)">Borrar</a>
        </fieldset>
```

Además del botón para agregar uno nuevo, el cual nos redirecciona al componente editar-productos mediante la directiva routerLink:

```
<fieldset class="form-group col-sm-1">
  <a class="btn btn-primary" [routerLink]="['/productos/agregar']">Nuevo Producto</a>
</fieldset>
```

Y en el componente typescript se hace la lógica para obtener esos productos y guardarlos en la variable productos:

```
constructor(private productoService:ProductoService){}

ngOnInit() {
  this.productos = this.productoService.obtenerProductos();
}

borrarProducto(idProducto:string){
  this.productoService.borrarProducto(idProducto).subscribe(data=>{
    console.log("Registro eliminado correctamente");
    this.ngOnInit();
  });
}
```

Ahora, en el componente editar-productos se implementa un formulario para obtener los atributos del producto ya sea para agregar o editar, mediante la directiva ngModel:

```
<form (ngSubmit)="onSubmit()" #productoForm="ngForm">
  <fieldset class="form-group">
    <div class="col-sm-12">
      <label class="control-label" for="nombre">Nombre</label>
      <input type="text" class="form-control" required [(ngModel)]="producto.nombre" name="nombre">
    </div>
  </fieldset>
  <fieldset class="form-group">
    <div class="col-sm-12">
      <label class="control-label" for="email">Detalle</label>
      <input type="email" class="form-control" required [(ngModel)]="producto.detalle" name="detalle">
    </div>
  </fieldset>
  <fieldset class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <button type="submit" class="btn btn-primary" [disabled]="!productoForm.form.valid">Enviar</button>
    </div>
  </fieldset>
</form>
```

Y en el componente typescript se desarrolla la lógica para agregar o editar un producto dependiendo de si se pasa el parámetro idProducto:

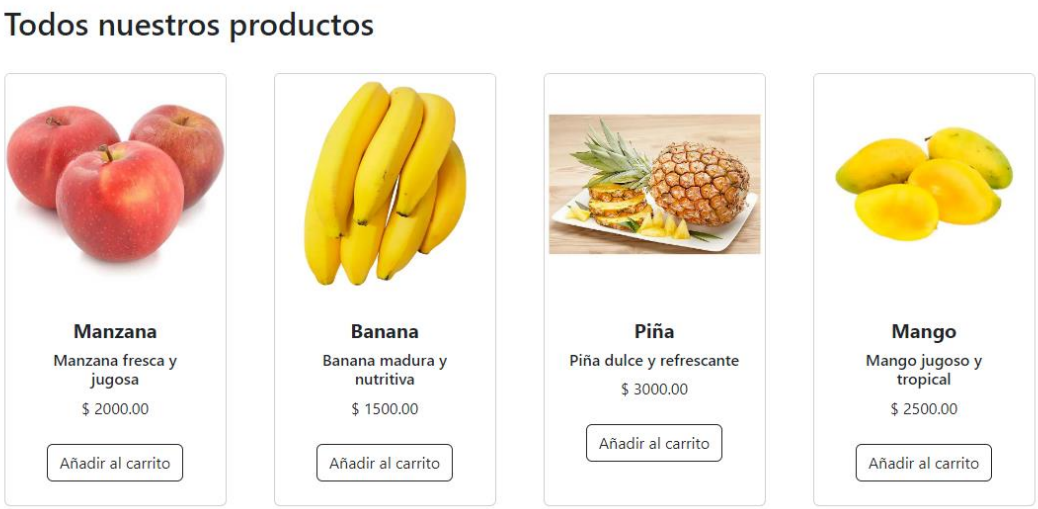
```
idProducto = '';
producto = new ProductoModel("", "", "");
constructor(private productoService: ProductoService, private route: ActivatedRoute,private router:Router) { }
ngOnInit() {
  this.idProducto = this.route.snapshot.params['idProducto'];
  console.log(`El id del producto es ${this.idProducto}`);
  if (this.idProducto) {
    //Si hay un id es porque se va a editar uno existente
    this.productoService.obtenerProducto(this.idProducto).subscribe(data => {
      this.producto = data[0];
    }, error => {
      console.log(error);
    });

  } else {
    //Si no hay ningun id es para agregar uno nuevo
    console.log("Agregar Producto");
  }
}
```

Podemos utilizar la lógica también para el componente que servirá para comprar productos (mostrar productos.html). Solo que, en lugar de una tabla, se utilizarán contenedores tipo Card, en las que se visualizarán detalles de cada producto.

```
<div class="card h-100">
    <form (ngSubmit)="crearCarrito();abrirModal(producto)">
        <!-- Product image-->
        <img class="card-img-top" src="{{ producto.imagen }}" alt="{{ producto.imagen }}"
        >
        <!-- Product details-->
        <div class="card-body p-4">
            <div class="text-center">
                <!-- Product name-->
                <h5 class="fw-bolder">{{ producto.nombre }}</h5>
                <h6>{{producto.descripcion}}</h6>
                <!-- Product  public (property) ProductoModel.precio: number  completo …
                $ {{producto.precio}}
            </div>
        </div>
```

```
<!-- Product actions-->
<div class="card-footer p-4 pt-0 border-top-0 bg-transparent">
    <div class="text-center"><button class="btn btn-outline-dark mt-auto"
    data-bs-toggle="modal"
            data-bs-target="#modalProducto" type="submit">Añadir al carrito</
            button>
    </div>
</div>
```

Así nos queda en la parte visual:



Ahora, para mostrar los productos que se han agregado a carrito haremos uso de una lista para indicar detalles como cantidad y precio de cada elemento:

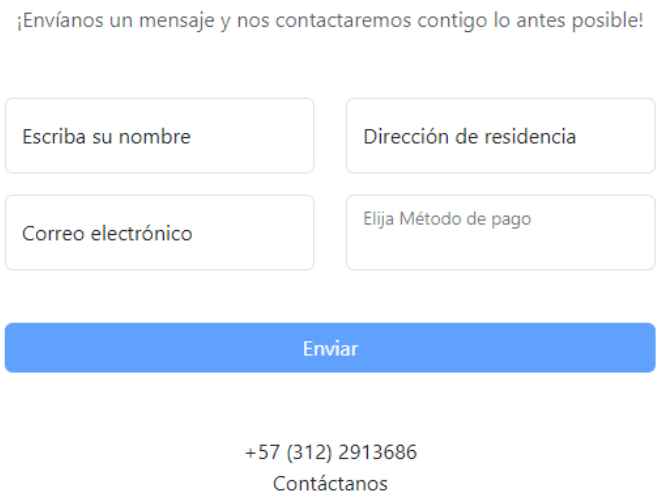
```
<li *ngFor="let prod of productosEnCarrito | async; let i = index" class="list-group">
    <!--  -->
    <div class="card" style="max-height: 300px;">
        <div class="row g-0">
            <div class="col-md-3">
                <img src="{{prod.producto.imagen}}" class="img-fluid rounded-start"
                    alt="...">
            </div>
            <div class="col-md-8 text-center">
                <h6 class="t">{{i+1}}. {{ prod.producto.nombre }}:   ${{
                    prod.valor_parcial }}</h6>

                <form #productoForm="ngForm">
                    <span class="card-text">Cant: {{prod.cantidad}} </span>
                    <span class="small">(Val Unit: ${{prod.producto.precio}})</span>
                    <div class="row">
                        <div class="col">
                            <input class="form-control" type="number" name="prod.cantidad"
                            [(ngModel)]="prod.cantidad"
                                min="1"
                            max="{{prod.producto.cantidad_producto}}" value="{{prod.
                            cantidad}}"
                                [value]="prod.cantidad" required>
```

En el componente typescript tenemos la asignación a la variable que contendrá los productos, mediante el método *ObtenerProductosCarrito()*, enviándole el id del carrito que está en la sesión:

```
obtenerProductosDeCarrito() {
  const idCarrito = sessionStorage.getItem("idCarritoS");
  if (idCarrito) {
    this.productosEnCarrito = this.productosCarritoService.obtenerProductosByIDCarrito(idCarrito);
  }

}
```

La vista que se tiene en el navegador es la siguiente:

Por último, para el formulario de registro de compra se tiene una pagina en la que se solicitan datos personales del comprador, como Nombres, dirección, correo y método de pago para terminar con la compra de los productos en carrito.



```
<form id="contactForm" data-sb-form-api-token="API_TOKEN" (ngSubmit)="onSubmit()"
#productoForm="ngForm">
 <input class="form-control" id="nombre" name="nombre" [(ngModel)]="compra.
 nombres" type="text"
     placeholder="Enter your name..." data-sb-validations="required"
     required />
 <label for="nombre">Escriba su nombre</label>
 <div class="invalid-feedback" data-sb-feedback="nombre:required">Este
 campo es
     obligatorio.
 </div>
```

```
<div class="form-floating mb-3">
   <select id="pago" name="pago" [(ngModel)]="compra.metodo_pago"
   class="form-control" required>
       <option value="efectivo">Efectivo</option>
       <option value="transferencia bancaria">Transferencia Bancaria</option>
       <option value="tarjeta de credito">Tarjeta de crédito</option>
       <option value="tarjeta debito">Tarjeta debito</option>

   </select><br>

   <label for="pago">Elija Método de pago</label>
   <div class="invalid-feedback" data-sb-feedback="pago:required">Este campo
   es
       obligatorio.
   </div>
</div>
```

```
 <!-- Submit Button-->
 <div class="d-grid"><button class="btn btn-primary btn-xl" [disabled]="!productoForm.
 form.valid" id="submitButton"
       type="submit">Enviar</button></div>
```

Mediante el método onSubmit() se obtendrán los datos para registrar la compra en la base de datos:

```
onSubmit() {
  console.log("Comprar");
  console.log(this.carrito);
  this.compra.idCarritoCompra = this.carrito.idCarrito;
  console.log(this.compra);
  this.compraService.agregarCompra(this.compra).subscribe(data => {
    //////
    sessionStorage.removeItem('idCarritoS');
    localStorage.removeItem('idCarritoL');
    const miModalElement = document.getElementById('exampleModalCenter');
    if (miModalElement) {
      console.log("Modal");
      miModalElement.style.display = 'block'; // Mostrar el modal
    }

  });
}
```

Al final, si la compra fue exitosa se mostrará un mensaje de confirmación mediante un modal: