DIPLOMADO DE ACTUALIZACIÓN EN NUEVAS TECNOLOGÍAS PARA EL DESARROLLO DE SOFTWARE

Taller 3 Unidad FrontEnd

Autor

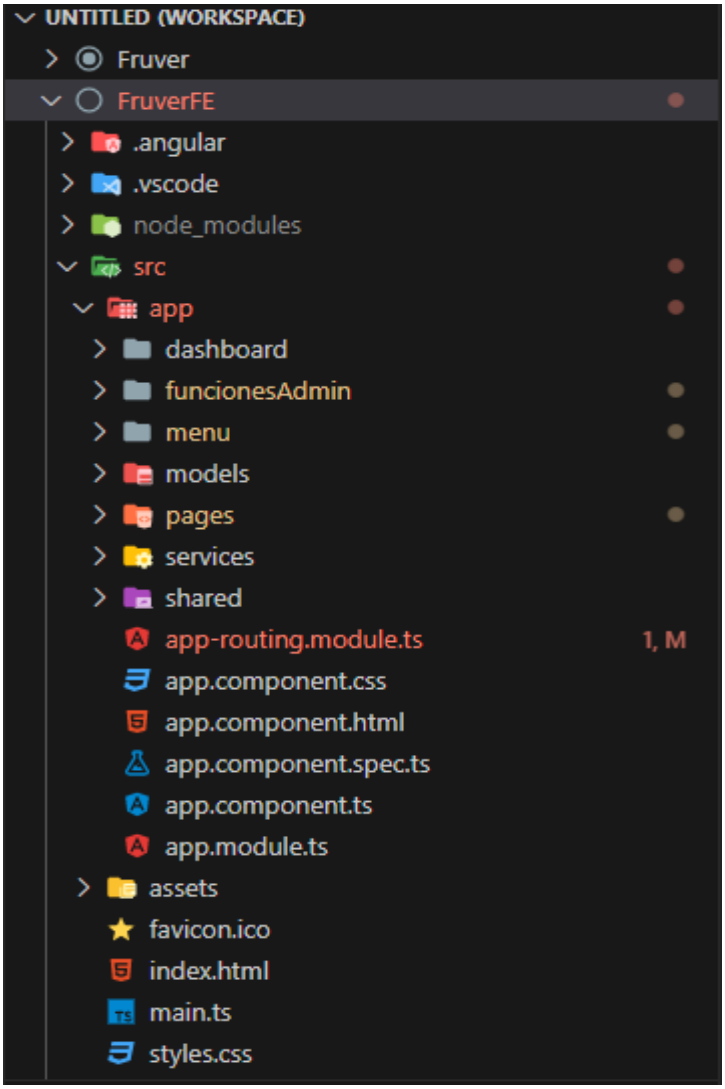
MAFLA ANDRES

Profesor
VICENTE AUX REVELO

DEPARTAMENTO DE SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE NARIÑO
JULIO, 2023

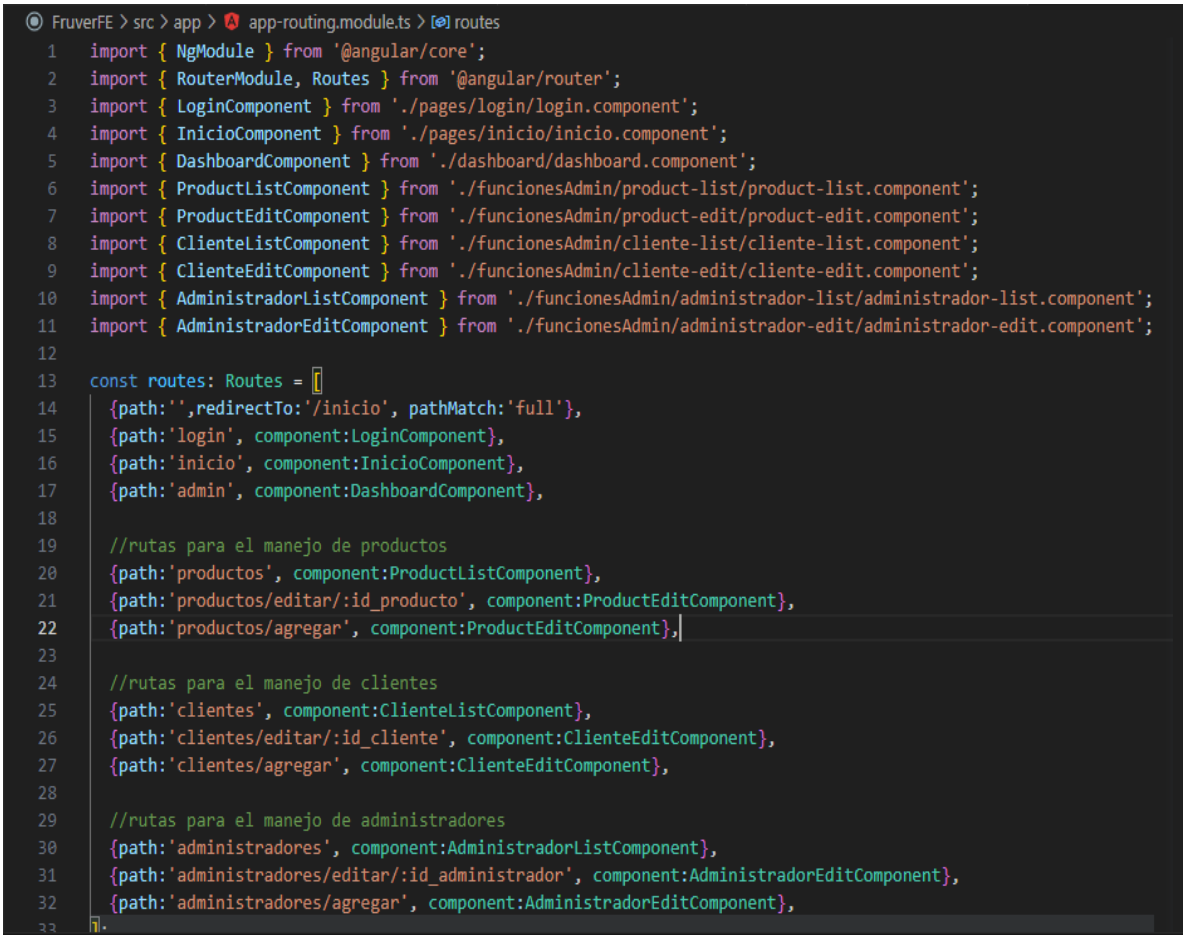
Taller Unidad 3 Frontend

El presente proyecto es un aplicativo web desarrollado en Angular que tiene como objetivo facilitar la compra de frutas y verduras en línea. Durante el proceso de construcción, se aplicaron los conocimientos adquiridos en nuestras clases sobre desarrollo de software de ultima generación. La aplicación se conecta al backend implementado con tecnologías como Node.js, el cual proporciona los datos necesarios para mostrar el catálogo de productos y gestionar el proceso de compra. Para lograr una interfaz de usuario intuitiva y atractiva. Además, se implementaron servicios y módulos para garantizar un código organizado, reutilizable y fácil de mantener. A lo largo de este proceso de desarrollo. A continuación, se presenta una descripción detallada de las funcionalidades implementadas y las decisiones de diseño que se tomaron en cada etapa del proyecto.

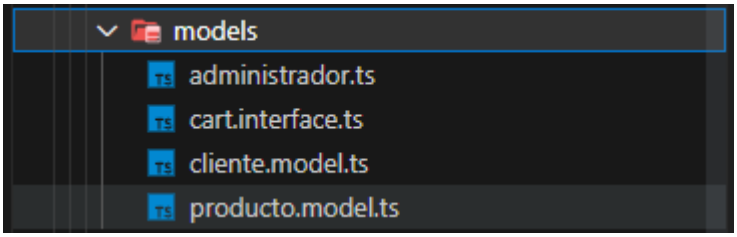


En el desarrollo del aplicativo web en Angular, se creó el módulo "AppRoutingModule" con el propósito de gestionar las rutas y la navegación del sistema. Mediante la importación de los módulos "NgModule" y "RouterModule" proporcionados por Angular, se configuraron las rutas de manera efectiva.

La estructura de rutas definida en el archivo "AppRoutingModule" aseguró una navegación coherente y fluida dentro de la aplicación. Al acceder a una ruta específica, el componente correspondiente se cargó automáticamente en la interfaz de usuario. Esto proporcionó a los usuarios una interacción sencilla y sin problemas con las diversas funcionalidades del aplicativo fruver.



Se creo la carpeta "models" la cual contiene los modelos de "producto", "cliente", "administrador" y "cart", que representan la estructura de datos de cada entidad en el aplicativo.



Para la creación de la pagina principal se realizo un componente “header” el cual contenía la barra de navegación principal, esta contiene elementos como lo son el logo del Fruver, una barra de búsqueda de productos, un carrito de compras y el botón usuario que permite realizar el login.

FruverFE > src > app > shared > header > header.component.html > nav.navbar.navbar-expand.navbar-light.bg-light

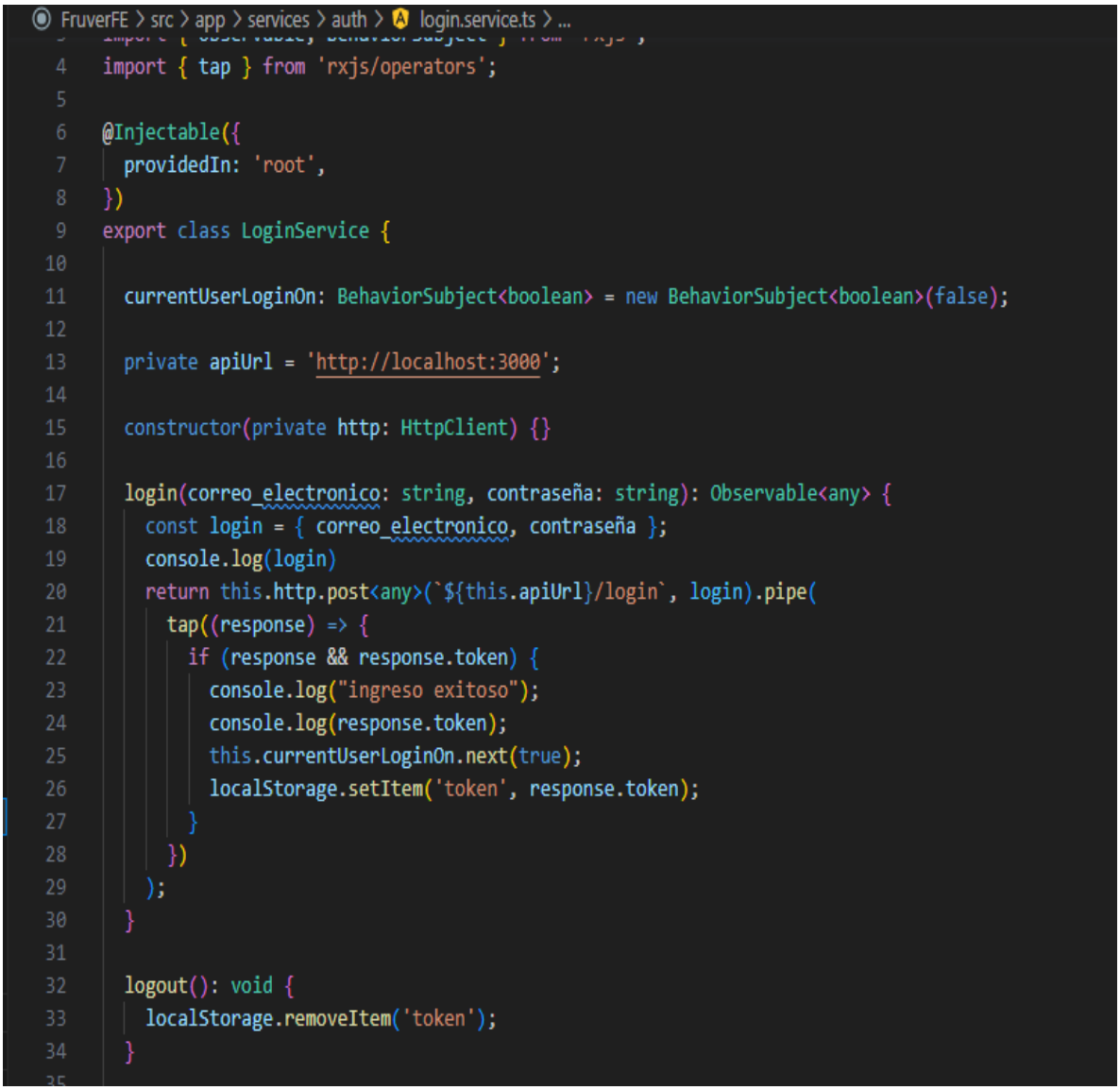
```
<a class="navbar-brand" href="/inicio">
  <img src="assets/imagenes/logo.png" alt="Logo de la empresa" height="70">
</a>
<div class="input-group d-flex justify-content-center">
  <div class="form-outline w-50">
    <input type="search" id="form1" class="form-control" />
  </div>
  <button type="button" class="btn btn-primary">
    <img src="assets/imagenes/lupa.png" alt="Carro de compras" height="20">
  </button>
</div>
<ul class="navbar-nav ml-auto ">
  <div class="navbar-cart" (click)="onToggleCart()">
    <div class="navbar-cart-items" *ngIf="(myCart$ | async)?.length">
      <p>{{ (myCart$ | async)?.length }}</p>
    </div>
    <i class="fas fa-shopping-cart"></i>
  </div>
  <li>
    <div class="dropdown">
      <button class="btn dropdown-toggle" type="button" id="userDropdown" data-bs-toggle="dropdown
        aria-expanded="false">
        <i class="fas fa-user"></i> Usuario
      </button>
      <ul class="dropdown-menu" aria-labelledby="userDropdown">
        <li *ngIf="!userLoginOn"><a class="dropdown-item" href="/login">Iniciar sesión</a></li>
        <li *ngIf="userLoginOn"><a class="dropdown-item" href="/login">Cerrar sesión</a></li>
      </ul>
    </div>
  </li>
</ul>
</nav>
```

El archivo “header.component.ts” utiliza servicios como "LoginService" y "CarritoService" para gestionar el inicio de sesión y el carrito de compras. Además, muestra el correo electrónico del usuario logueado y permite alternar la visualización del carrito. El componente se inicializa suscribiéndose a cambios en el estado de inicio de sesión para actualizar la interfaz en consecuencia.

FruverFE > src > app > shared > header > header.component.ts > ...

```ts
import { Component, OnInit } from '@angular/core';
import { CarritoService } from 'src/app/services/auth/carrito.service';
import { LoginService } from 'src/app/services/auth/login.service';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent implements OnInit{
  userLoginOn:boolean=false;
  correo?:'';
  viewCart: boolean = false;
  myCart$ = this.carritoService.myCart$;

  constructor(private loginService: LoginService, private carritoService:CarritoService){}
  getCorreo():string{
    return this.loginService.getCorreo()?? '';
  }

  onToggleCart() {
    this.viewCart = !this.viewCart
  }

  ngOnInit(): void {
    this.loginService.currentUserLoginOn.subscribe({
      next:(userLoginOn)=>{
        this.userLoginOn=userLoginOn;
      }
    })

  }
}
```

el servicio "LoginService", está encargado de gestionar la autenticación de usuarios. Utiliza el HttpClient para realizar peticiones HTTP al servidor de autenticación y manejar el inicio y cierre de sesión. Almacenando el estado de inicio de sesión en un BehaviorSubject, se actualiza la interfaz según el estado del usuario. Además, proporciona métodos para obtener el token de autenticación, el correo electrónico del usuario y verificar si el usuario está logueado.



El servicio "ProductoService" es fundamental para gestionar los productos disponibles que van a ser mostrados en la página principal y el carrito de compras en el aplicativo. Utilizando el HttpClient, se comunicará con el servidor para realizar diversas operaciones CRUD, como obtener la lista de productos disponibles, agregar nuevos productos, actualizar información de productos existentes y eliminar productos del inventario. Con esto, el servicio facilitará una experiencia de usuario fluida y eficiente al interactuar con los productos y al realizar compras en el aplicativo.

Para completar la pagina principal se creo un footer el cual se posiciono en la parte inferior y lleva información acerca del sitio web

FruverFE > src > app > shared > footer > footer.component.html > ...

Go to component

```html
<footer class="bg-dark text-white text-center py-3 fixed-bottom">
  <div class="container">
    <p class="mb-1">Fruver: Tu tienda de frutas y verduras frescas</p>
    <p class="mb-1">¡Siempre la mejor calidad al mejor precio!</p>
    <p>Creado por Andrés Mafla</p>

    <div class="social-icons mt-3">
      <a href="#" class="text-white mx-2"><i class="fab fa-facebook"></i></a>
      <a href="#" class="text-white mx-2"><i class="fab fa-twitter"></i></a>
      <a href="#" class="text-white mx-2"><i class="fab fa-instagram"></i></a>
    </div>
  </div>
</footer>
```

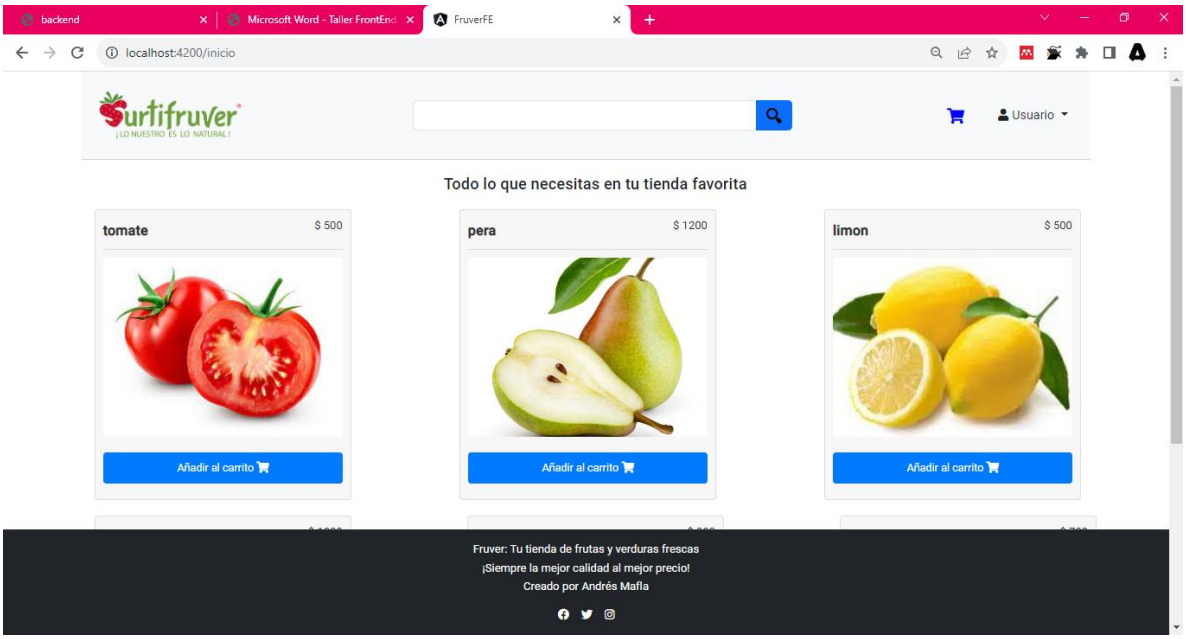
Todos los elementos presentados hasta ahora hicieron parte de la pagina principal la cual se manejo con el componente header, fotter y productos.

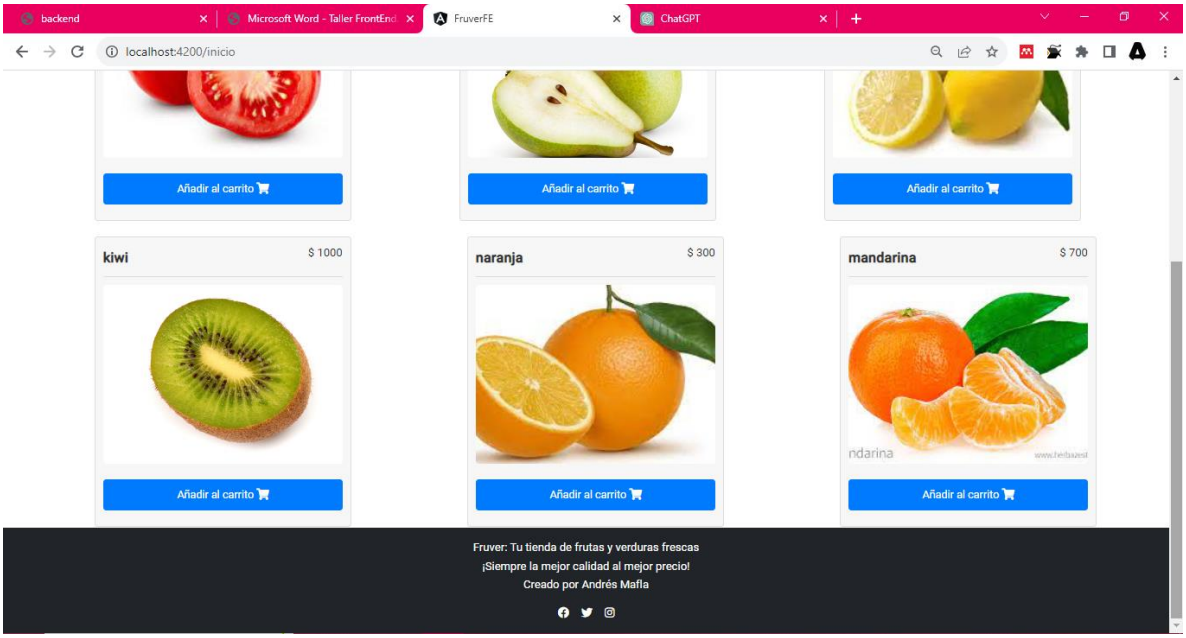
FruverFE > src > app > app.component.html > ...

Go to component

```html
<div class="container">
    <app-header></app-header>
    <router-outlet></router-outlet>
    <app-footer></app-footer>
</div>
```

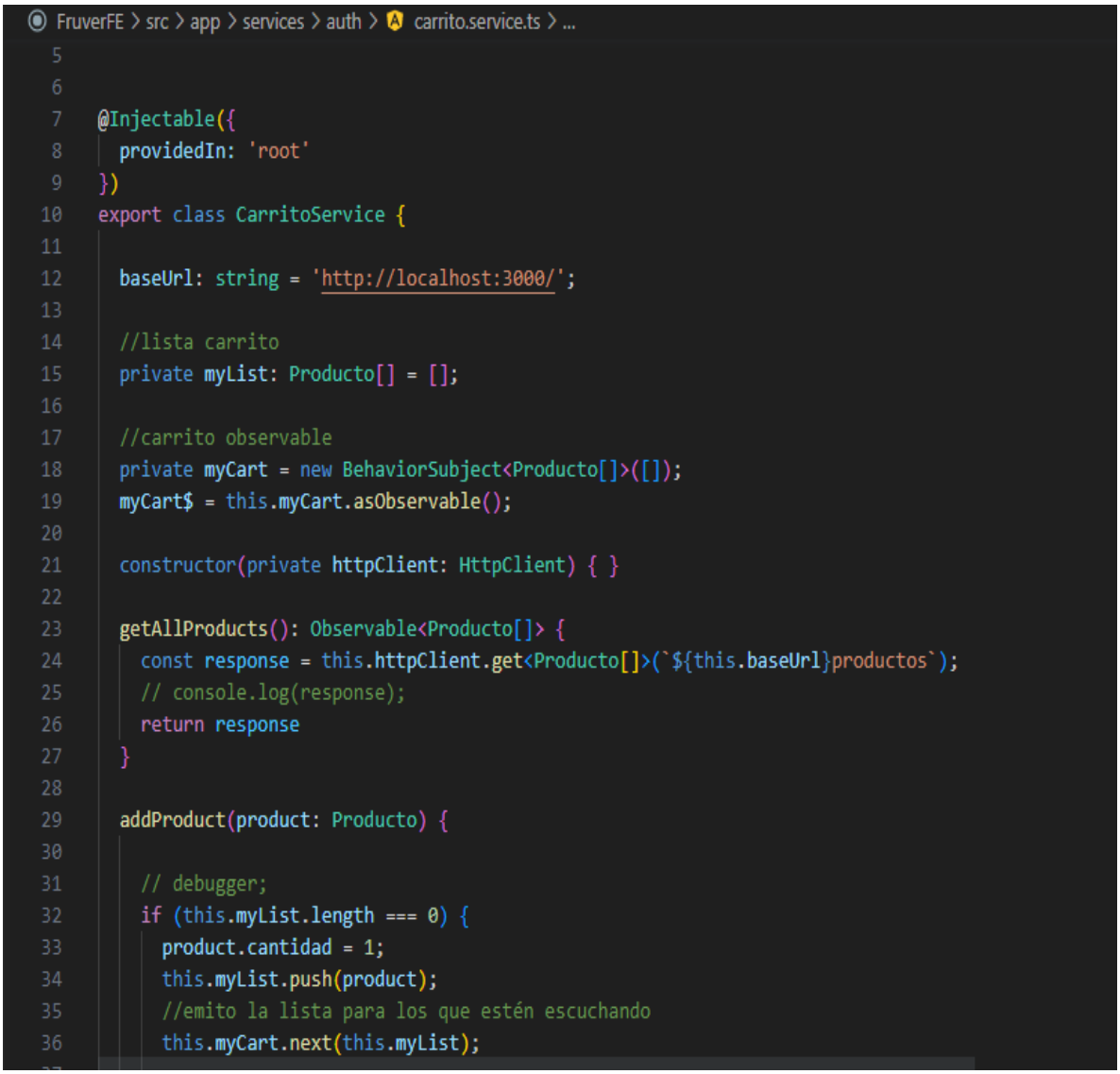
Como podremos ver a continuación se logro obtener una pagina de inicio muy organizada y amable con el usuario, esta contiene los elementos necesarios para gestionar una aplicación de Fruver, presentado una barra de navegación que nos permite buscar productos, gestionar un carrito de compras y realizar un inicio de sesión, en la parte central nos presenta los productos disponibles en la tienda y finalmente nos encontramos con el fotter





Para el manejo del carrito de compras se implementó el servicio llamado “CarritoService”. Este servicio se encarga de gestionar un carrito de compras. El servicio mantiene una lista de productos seleccionados y emite cambios a través de un observable llamado myCart$. Los productos se agregan al carrito mediante el método addProduct(), el cual controla la cantidad de cada artículo y actualiza el carrito.

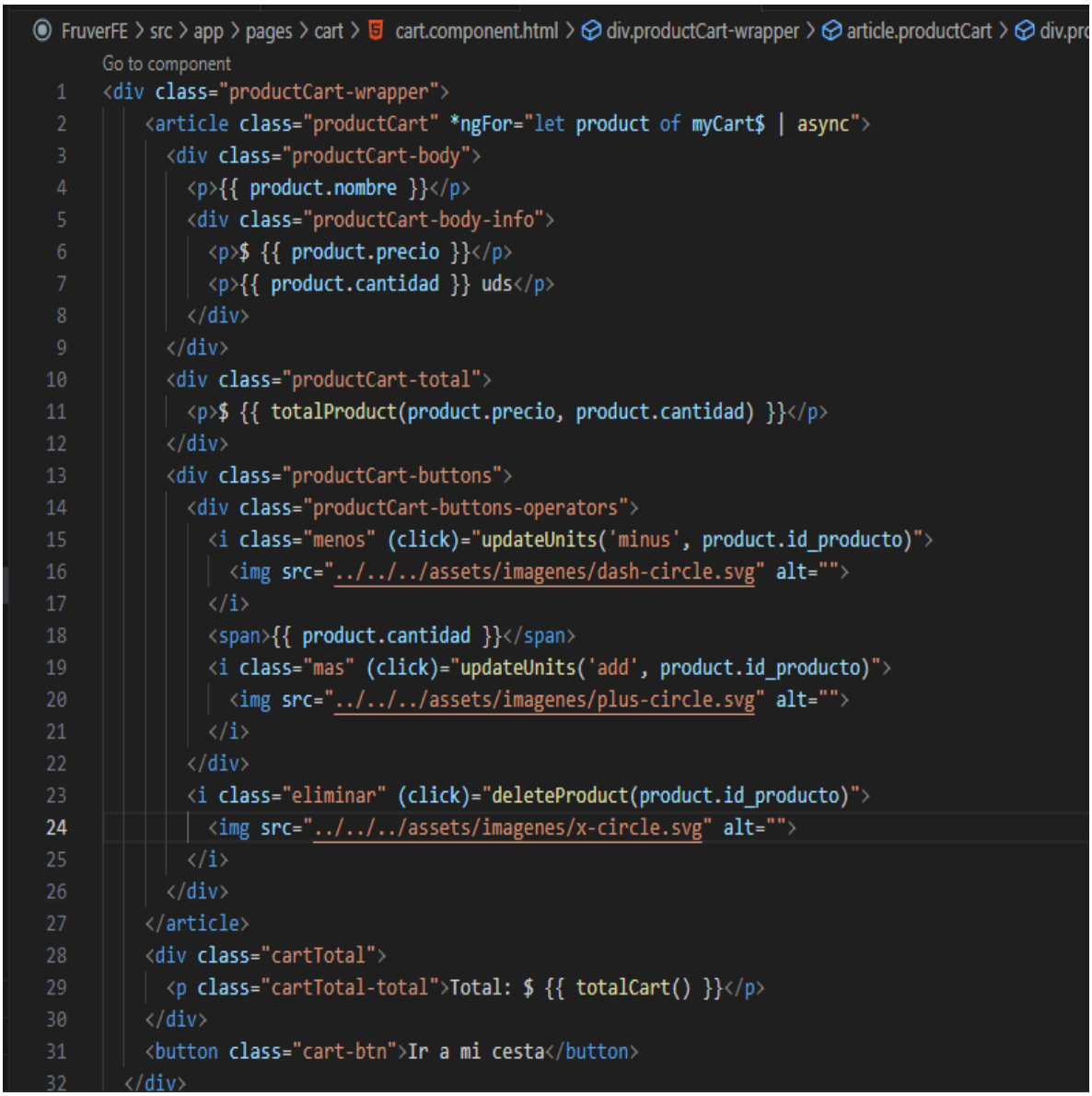
Además, el servicio proporciona funciones para buscar productos por su ID y eliminarlos del carrito. La función totalCart() calcula el precio total del carrito sumando el precio de cada producto multiplicado por su cantidad.



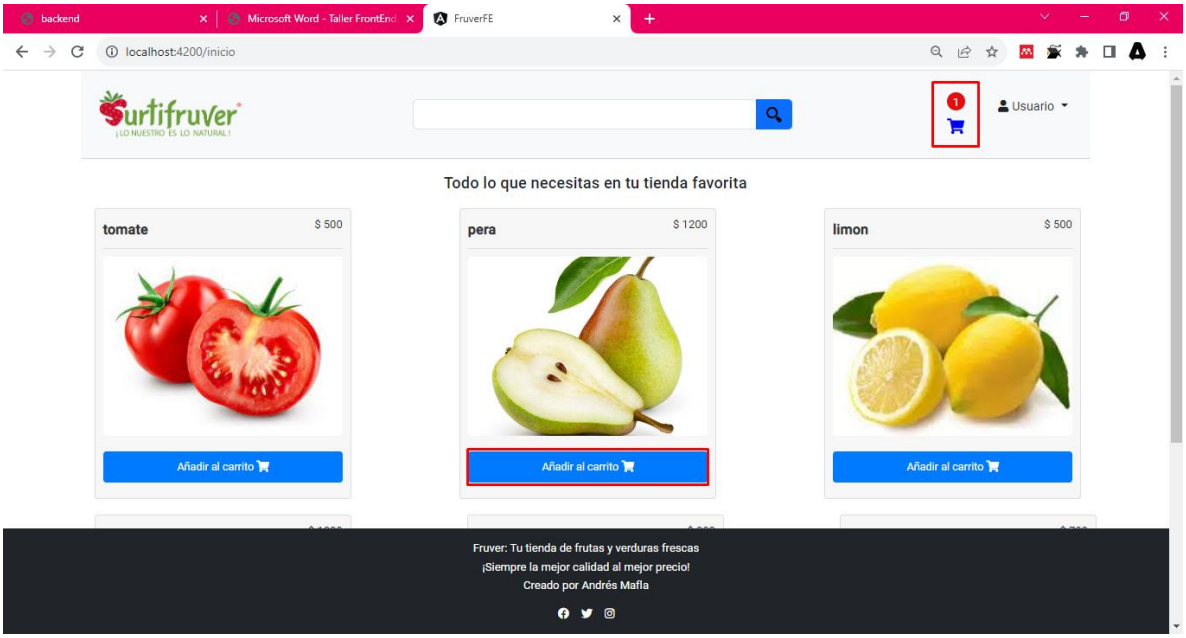
Se creo un componente Angular llamado CartComponent, que se encarga de gestionar la visualización y funcionalidad del carrito de compras de la aplicación. Utiliza el servicio CarritoService para obtener y manipular los productos en el carrito, así como calcular el total de la compra.

El componente muestra la lista de productos seleccionados mediante el observable myCart$, permitiendo al usuario modificar la cantidad de unidades de un producto y eliminarlos del carrito. También calcula el total de cada producto y el costo total del carrito.

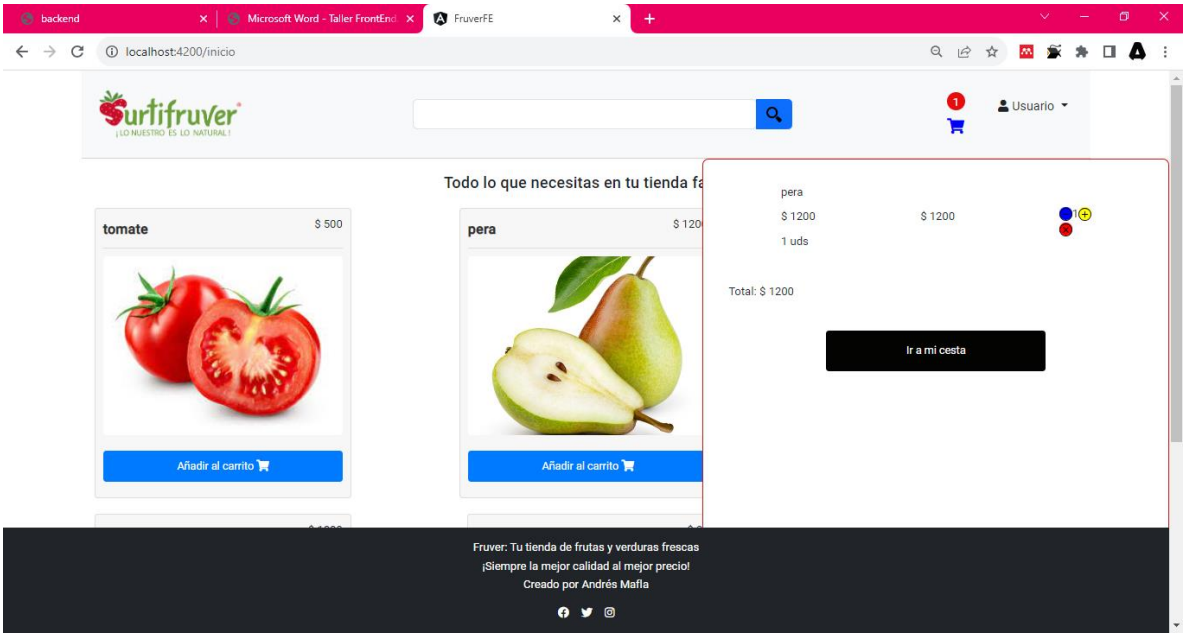
El html del componente muestra el carrito de compras. Utiliza el observable myCart$ para listar los productos y permite ajustar las cantidades, eliminar productos y calcular el costo total. Cada producto muestra su nombre, precio y cantidad. Los botones de suma y resta llaman a updateUnits(), mientras que el ícono de eliminar invoca deleteProduct(). En general, brinda una experiencia interactiva para gestionar el carrito de compras en la aplicación.



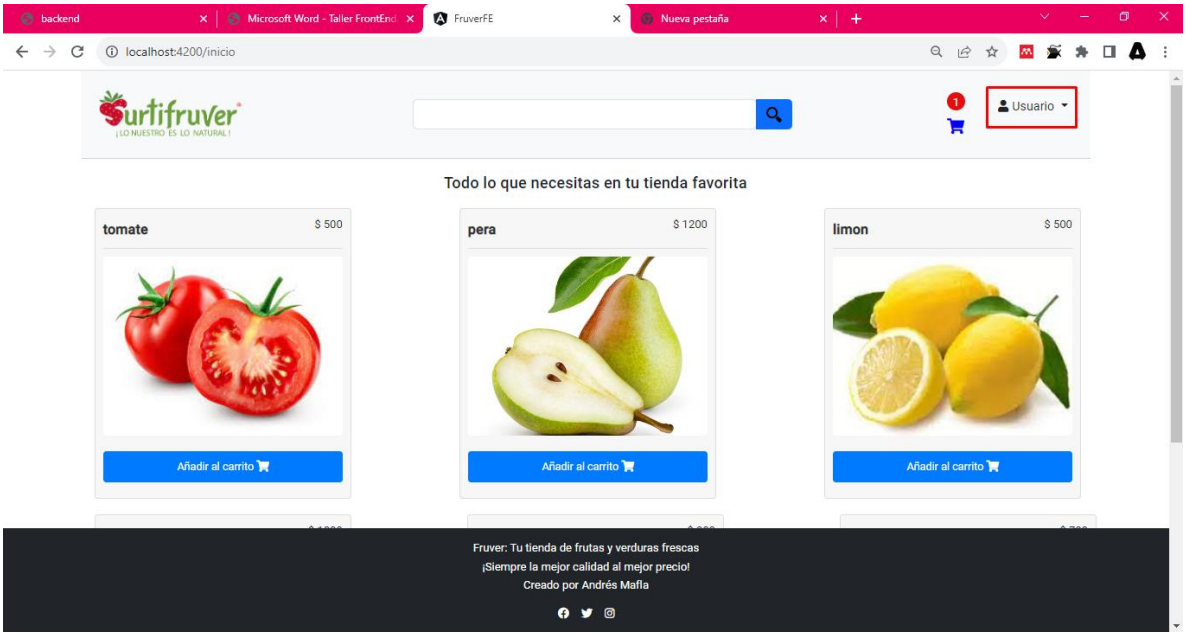
A continuación, se mostrará el resultado de la gestión del carrito de compras con diversas pruebas. Después de dar clic sobre el botón de añadir al carrito, el carrito de compras inicia un contador de los productos agregados.



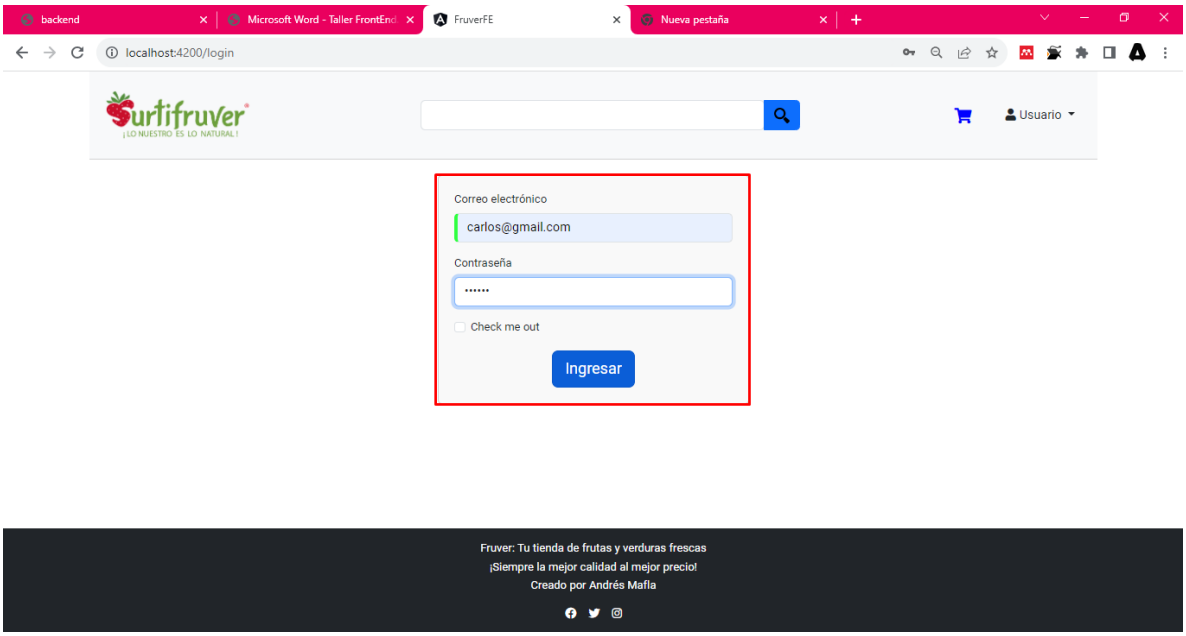
Al dar clic sobre el carrito de compras se despliega la lista de productos seleccionados donde podemos aumentar o disminuir la cantidad del producto, de la misma forma se puede quitar el producto del carrito y observar el valor total de la compra.



El aplicativo tiene la función de login en la cual accederemos a las funciones del administrador para poder gestionar los productos, clientes, administradores y ventas.



Al dar clic en iniciar sesión se nos mostrará el login donde nos pedirá el correo y la contraseña para poder ingresar.



Se creo el componente menú y dashboard los cuales serán la página inicial del administrador, el componente dashboard contendrá el menú y un espacio de trabajo donde se realizarán las diferentes tareas del manejo del Fruver.

producto.service.ts M | app-routing.module.ts | menu.component.html

FruverFE > src > app > menu > menu.component.html > div.menu

Go to component

```
<div class="menu">
    <h4>Menu de Opciones</h4>
    <hr>
    <ul>
        <li><a routerLink="/productos">Productos</a></li>
        <li><a routerLink="/clientes">Clientes</a></li>
        <li>
            <a href="#">Ventas</a>
            <ul class="submenu">
                <li><a href="#">Listar Pedidos</a></li>
                <li><a href="#">Confirmar Pedidos</a></li>
            </ul>
        </li>
        <li><a routerLink="/administradores">Administrador</a></li>
    </ul>
</div>
```

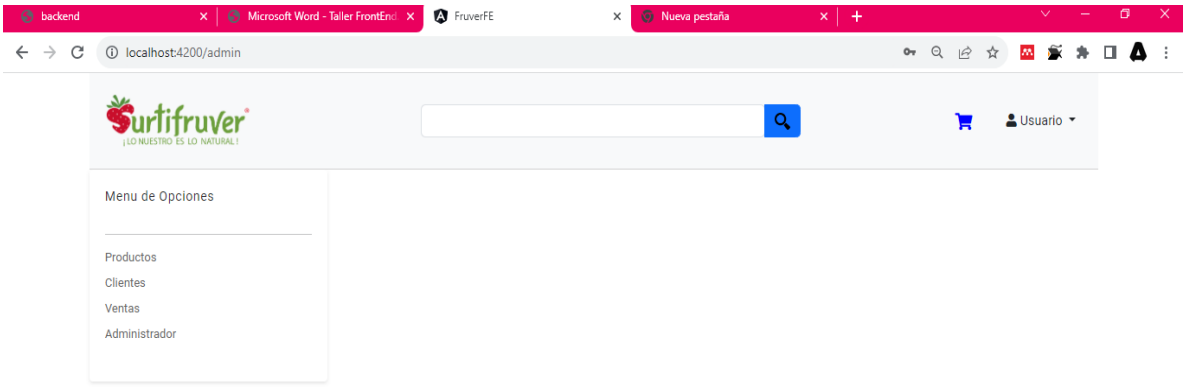
producto.service.ts M | app-routing.module.ts | dashboard.component.html

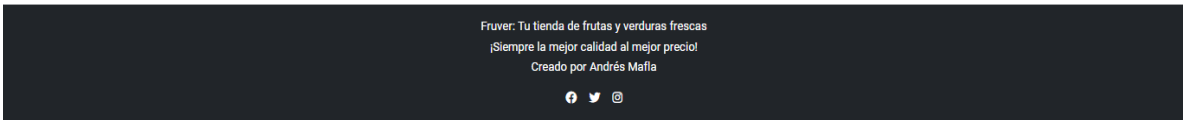
FruverFE > src > app > dashboard > dashboard.component.html > div.row

Go to component

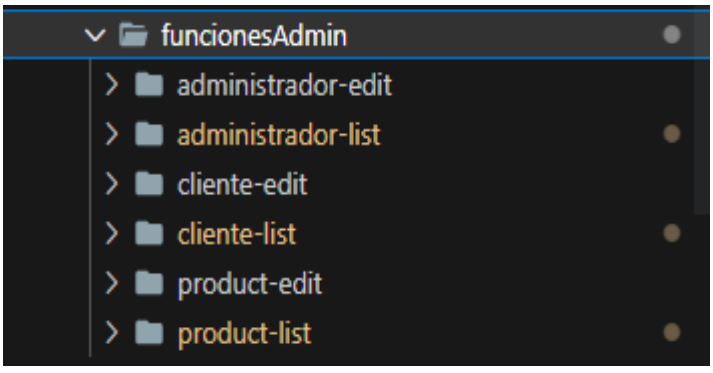
```
<div class="row">
  <div class="col-3"><app-menu></app-menu></div>
  <div class="div-9"></div>
</div>
```

De modo que al iniciar sesión el administrador se encontrará con el siguiente espacio de trabajo, donde el menú lateral tendrá las opciones de gestionar productos, clientes, ventas y administradores.

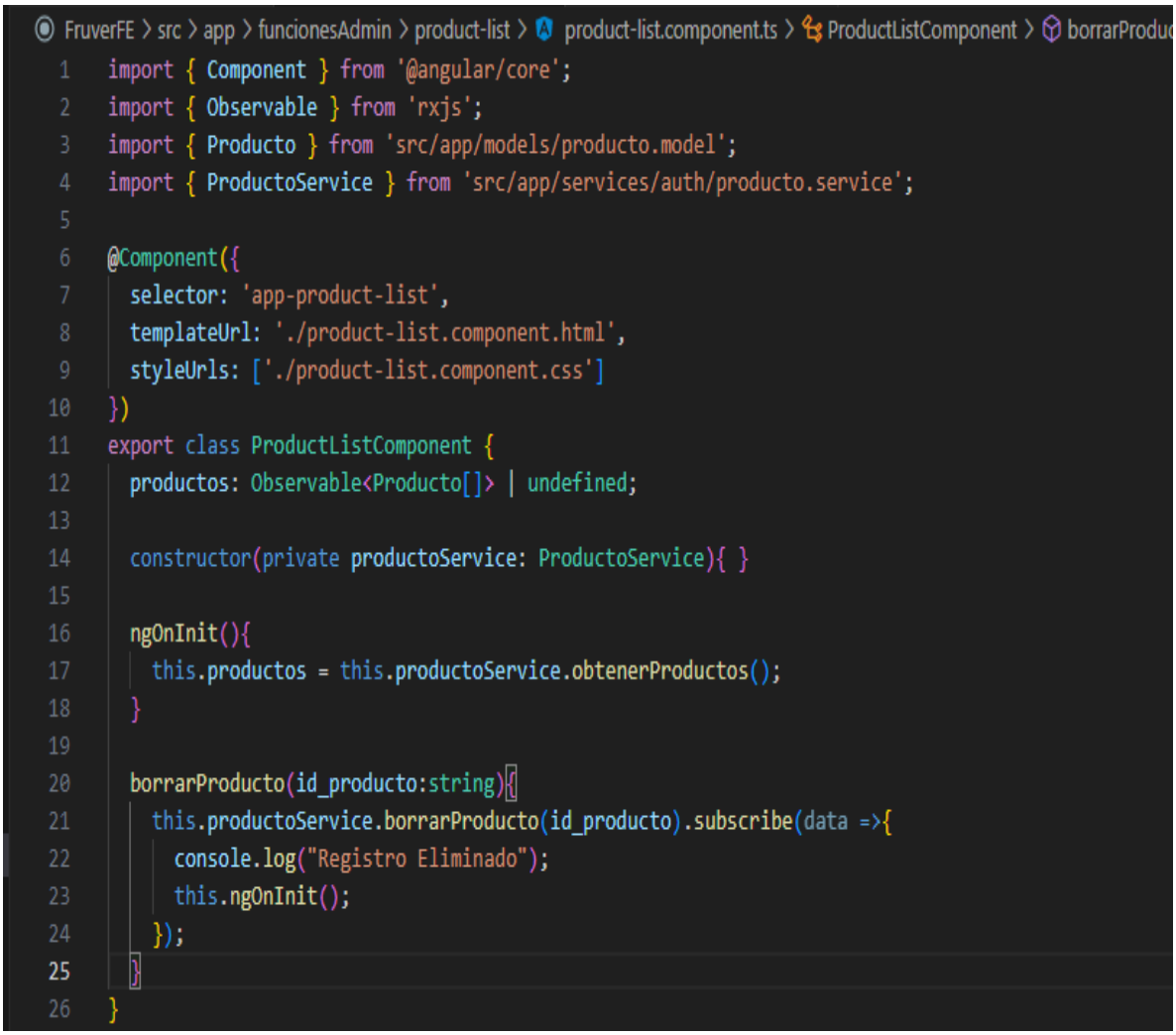




Para gestionar los productos, clientes, ventas y administradores se creó una carpeta llamada “funcionesAdmin” en la cual se crearán los diferentes componentes que ayudarán en el trabajo de esas funcionalidades.

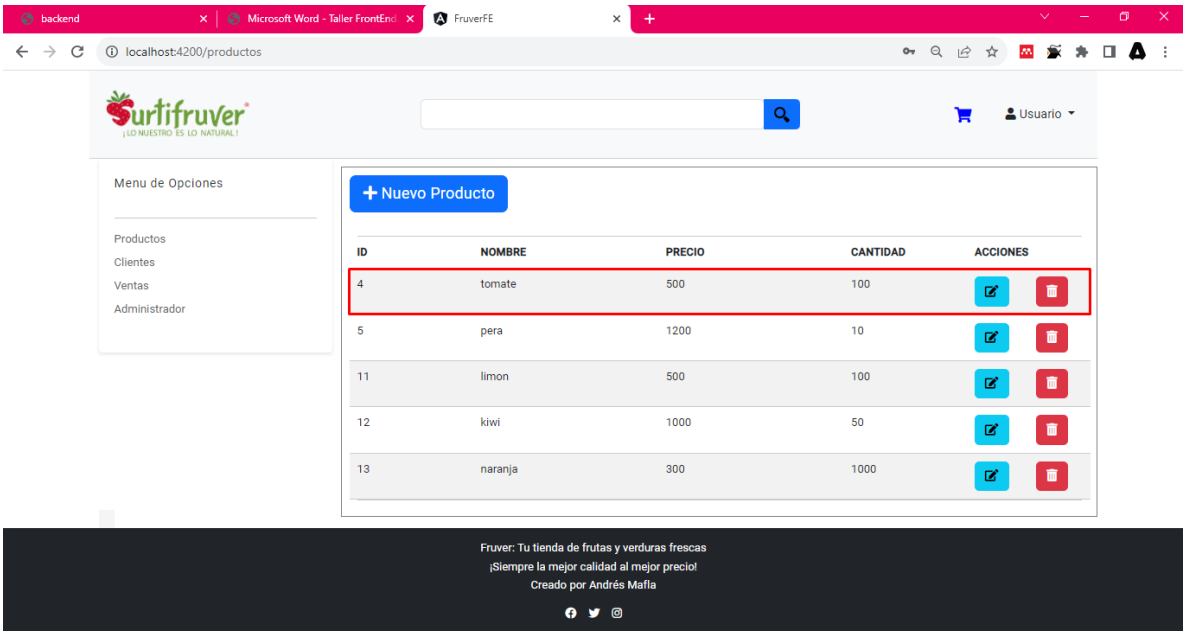


Para gestionar los productos se creó los componentes “product-list” y “product-edit”, el primer componente muestra una lista de productos y permite eliminarlos. Utiliza el servicio ProductoService para obtener y eliminar productos. Al iniciarse, carga la lista de productos. Cuando se elimina un producto, se actualiza automáticamente la lista. Es un ejemplo sencillo de cómo trabajar con componentes y servicios en Angular para gestionar datos y su visualización.

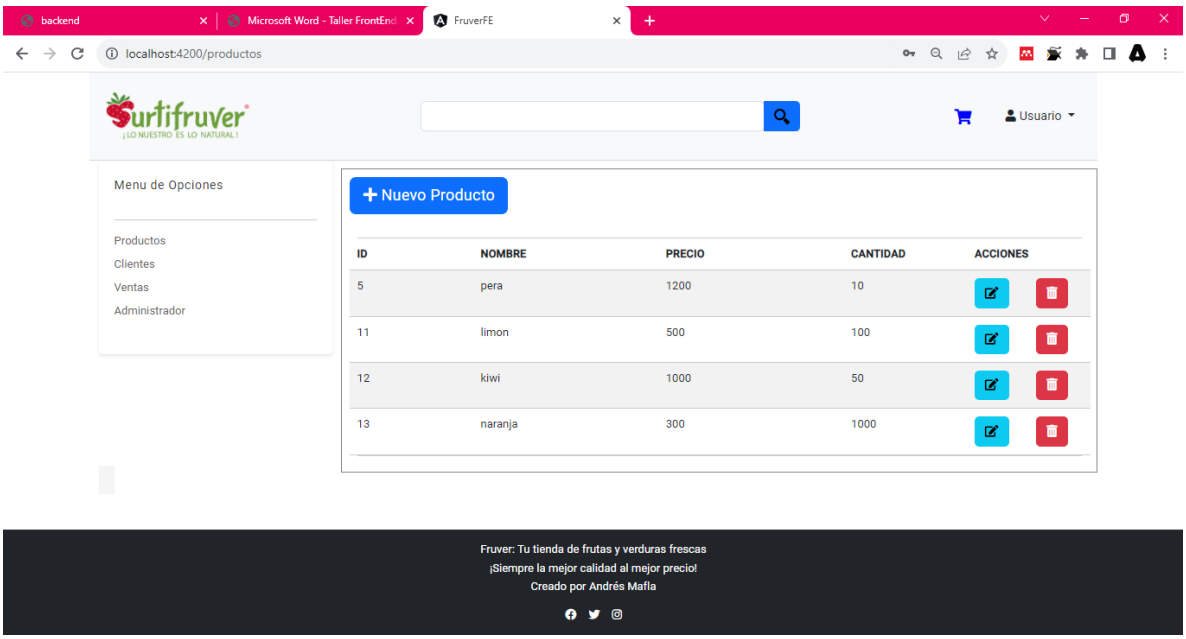


El componente html tiene un botón "Nuevo Producto" que redirige a una página para agregar un nuevo producto. Para cada producto en la tabla, se proporcionan botones de edición y eliminación para actualizar o borrar los productos. La lista de productos se carga dinámicamente utilizando la directiva *ngFor y el async pipe para obtener datos de manera asíncrona.

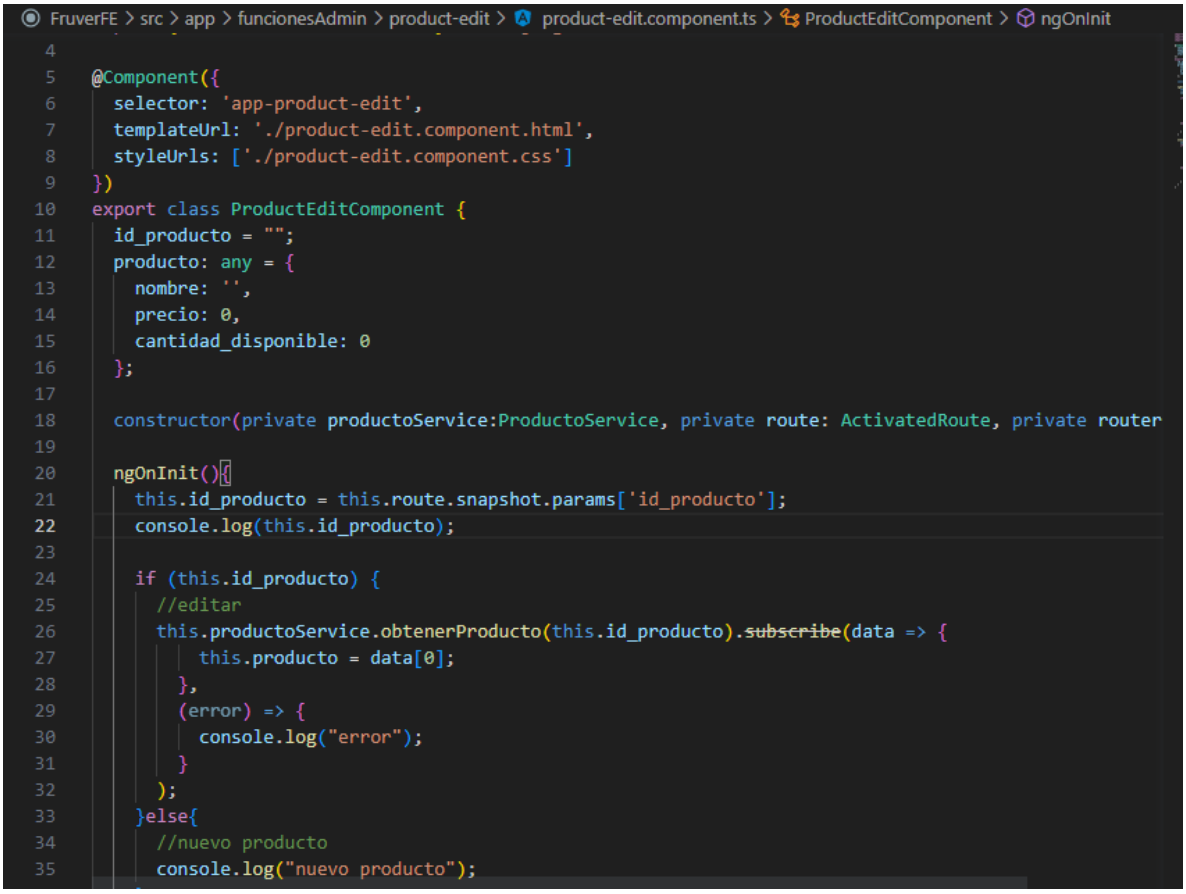
La vista principal de gestionar productos muestra un botón para agregar un nuevo producto, más abajo encontramos una tabla con todos los productos, adicionalmente se agrega una columna donde podemos encontrar los botones de editar y eliminar.



Para eliminar un producto, simplemente se da clic en el botón "Eliminar", tal como se muestra a continuación. En este ejemplo, procederemos a eliminar el producto "tomate".



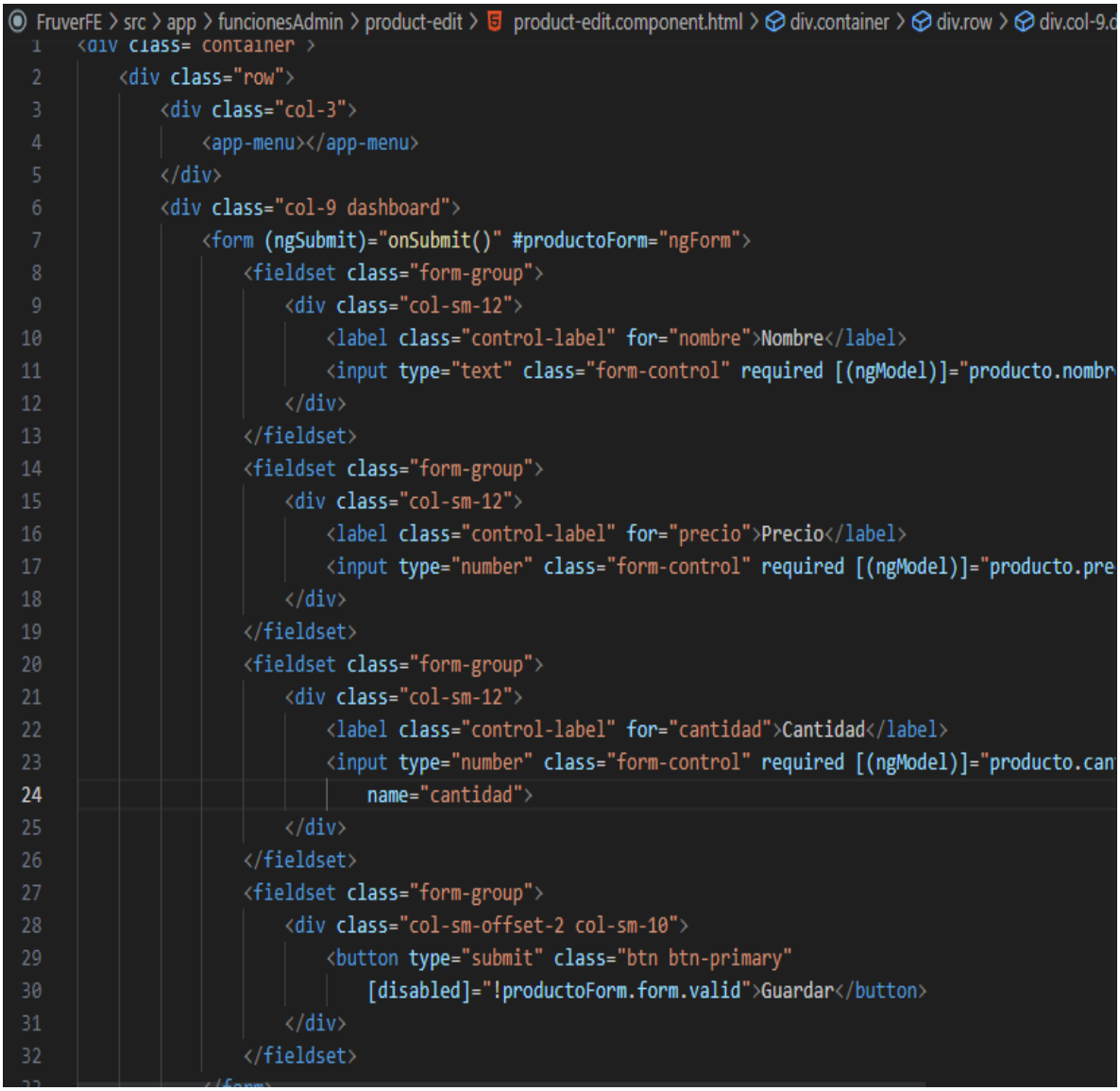
Para implementar la funcionalidad del botón editar y crear nuevo producto, se utilizó en componente de producto-edit. Aquí se implementó el método onSubmit() se activa cuando el usuario envía el formulario. Dependiendo de si el producto ya tiene un id_producto o no, se llama al método actualizarProducto() o agregarProducto() del servicio ProductoService. Estos métodos realizan solicitudes al servidor para guardar los datos del producto editado o nuevo. Si la operación se realiza con éxito, el componente redirecciona al usuario a la lista de productos.



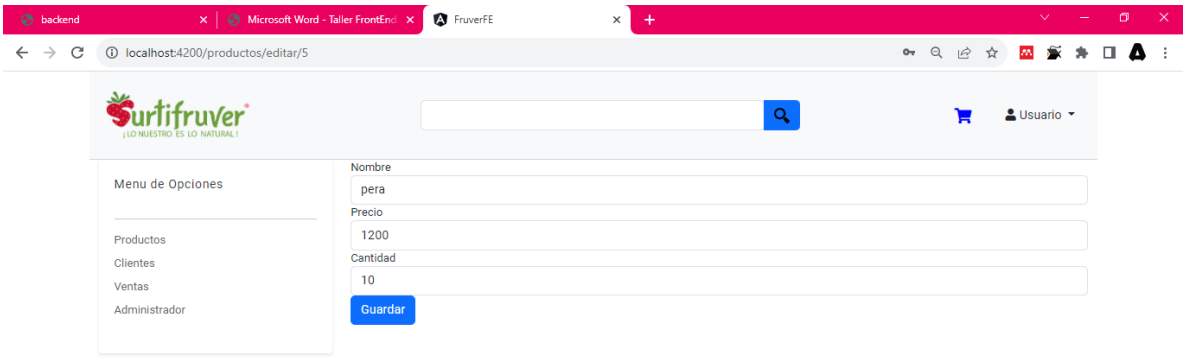
```
FruverFE > src > app > funcionesAdmin > product-edit > product-edit.component.ts > ProductEditComponent > ngOnInit
4
5   @Component({
6     selector: 'app-product-edit',
7     templateUrl: './product-edit.component.html',
8     styleUrls: ['./product-edit.component.css']
9   })
10  export class ProductEditComponent {
11    id_producto = "";
12    producto: any = {
13      nombre: '',
14      precio: 0,
15      cantidad_disponible: 0
16    };
17
18    constructor(private productoService:ProductoService, private route: ActivatedRoute, private router
19
20    ngOnInit(){
21      this.id_producto = this.route.snapshot.params['id_producto'];
22      console.log(this.id_producto);
23
24      if (this.id_producto) {
25        //editar
26        this.productoService.obtenerProducto(this.id_producto).subscribe(data => {
27            this.producto = data[0];
28          },
29          (error) => {
30            console.log("error");
31          }
32        );
33      }else{
34        //nuevo producto
35        console.log("nuevo producto");
```

```
39    onSubmit(){
40      console.log("submit realizado");
41      if (this.producto.id_producto) {
42        //viene de editar
43        this.productoService.actualizarProducto(this.producto).subscribe(
44          data => {
45            console.log(data);
46            this.router.navigate(['/productos']);
47          }
48        );
49      }else{
50        //viene de crear
51        this.productoService.agregarProducto(this.producto).subscribe(data =>{
52          this.router.navigate(['/productos']);
53        });
54      }
55    }
56  }
```

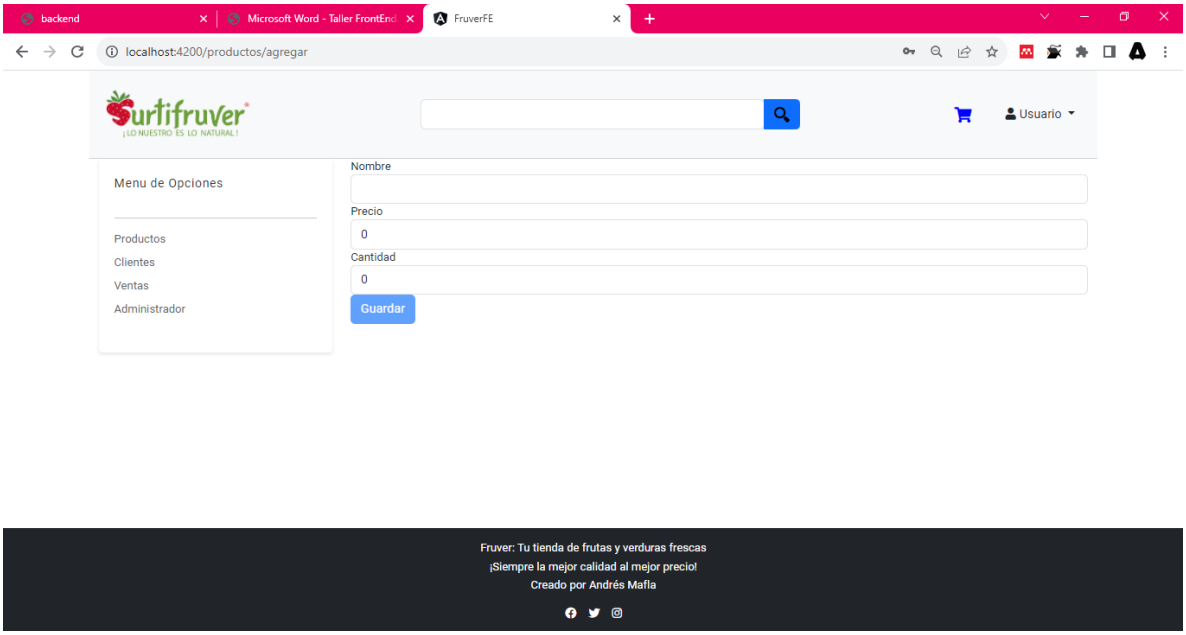
El componente “producto-edit” html presenta una vista de formulario para editar o crear productos. La página se divide en dos columnas, una muestra un menú de navegación y la otra contiene el formulario. Los campos "Nombre", "Precio" y "Cantidad" son enlazados con las propiedades del componente, manteniendo así su sincronización. El botón "Guardar" solo está habilitado si el formulario es válido. Al enviar el formulario, se activa el método onSubmit(), encargado de procesar y guardar los datos del producto. Es una implementación sencilla y funcional que brinda una experiencia intuitiva para gestionar productos en la aplicación.



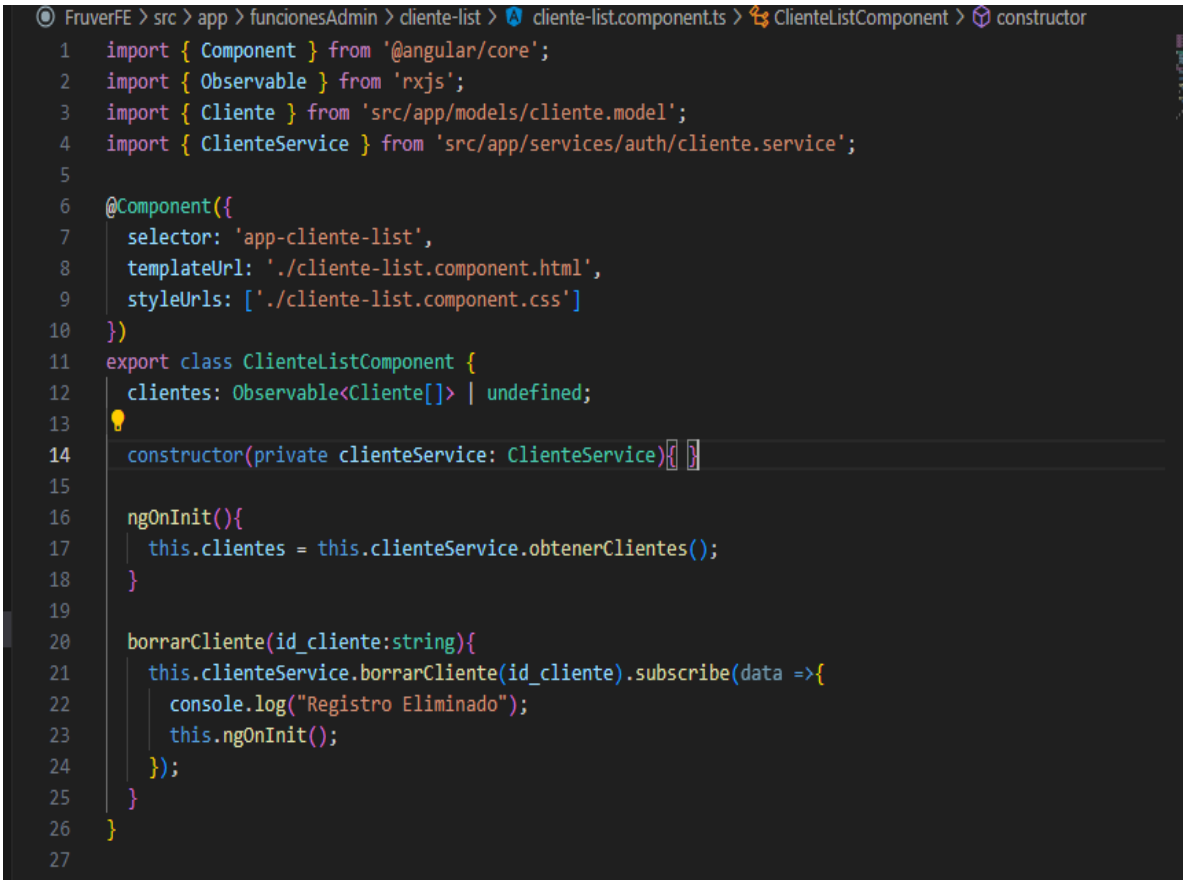
Cunado damos clic en editar se carga el formulario con la información respectivamente.



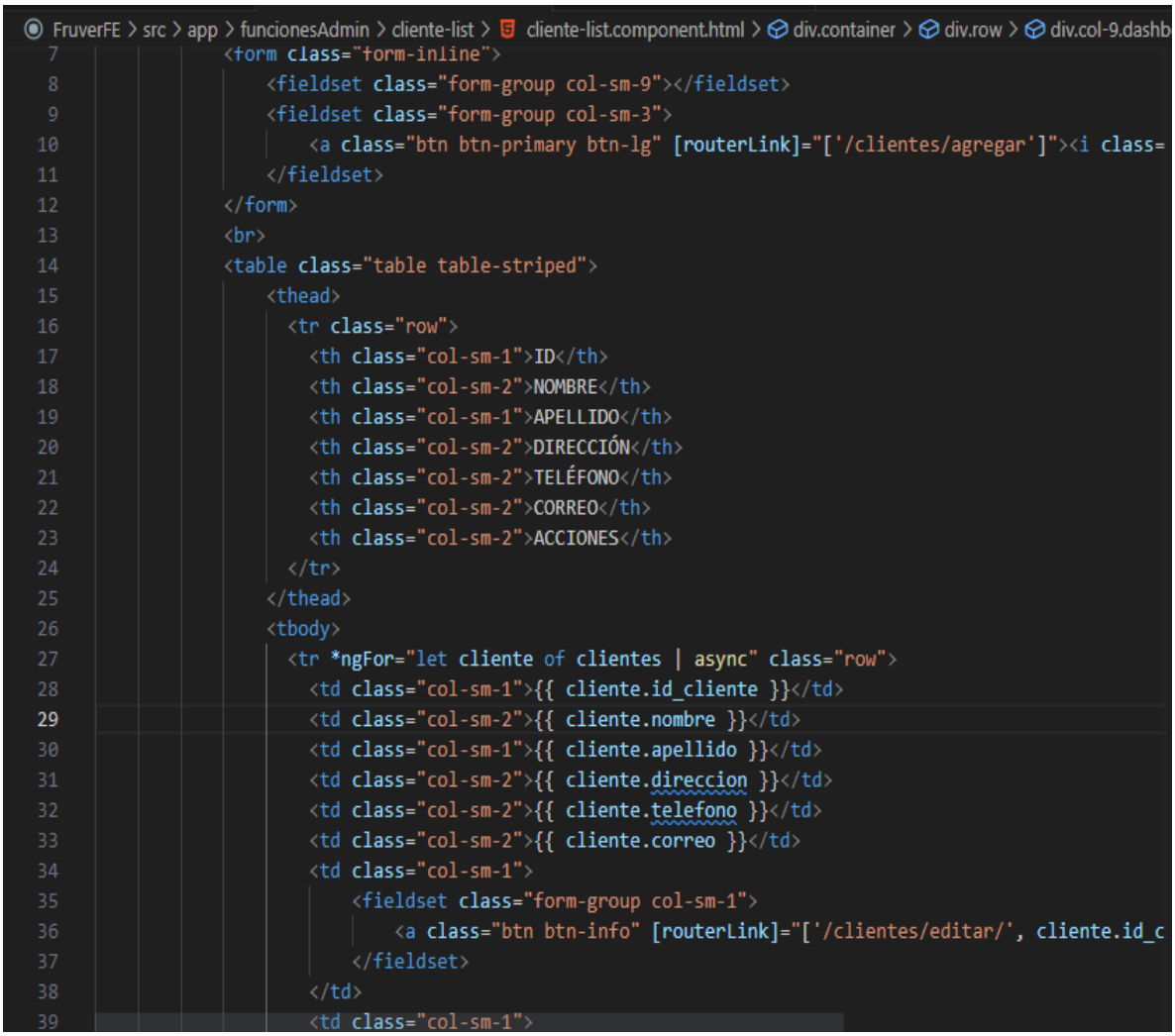
Cuando damos clic en nuevo producto, se cargar el formulario con los campos vacíos.



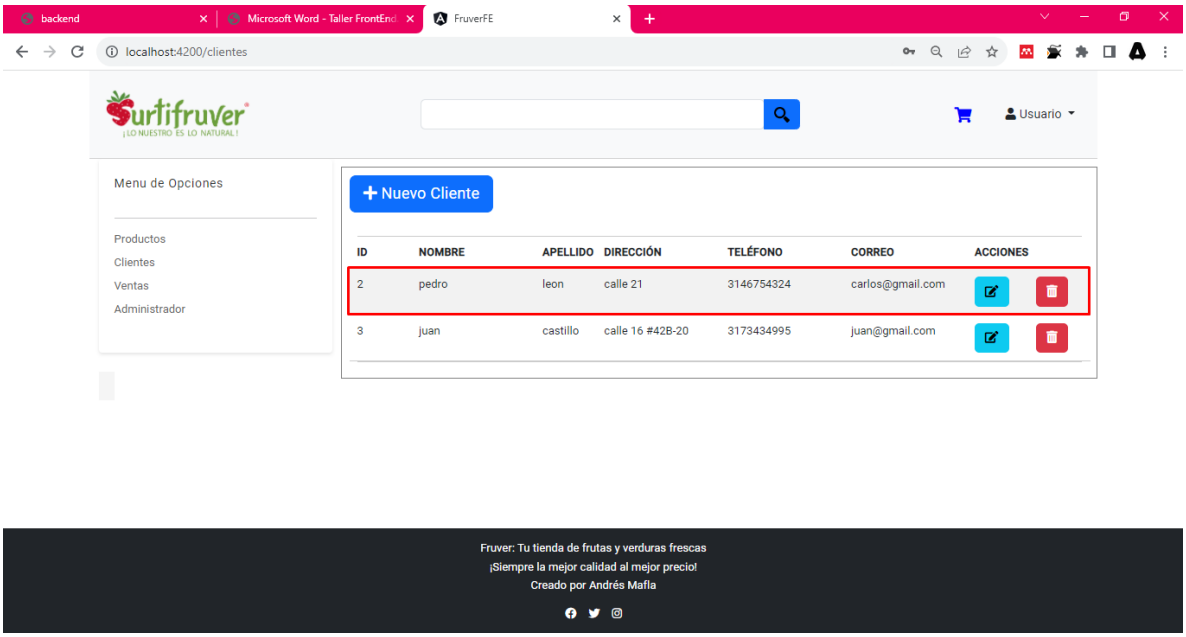
Para gestionar los productos se creó los componentes “cliente-list” y “cliente-edit”, el primer componente muestra una lista de clientes y permite eliminarlos. Utiliza el servicio ClienteService para obtener y eliminar productos. Al iniciarse, carga la lista de clientes. Cuando se elimina un cliente, se actualiza automáticamente la lista.



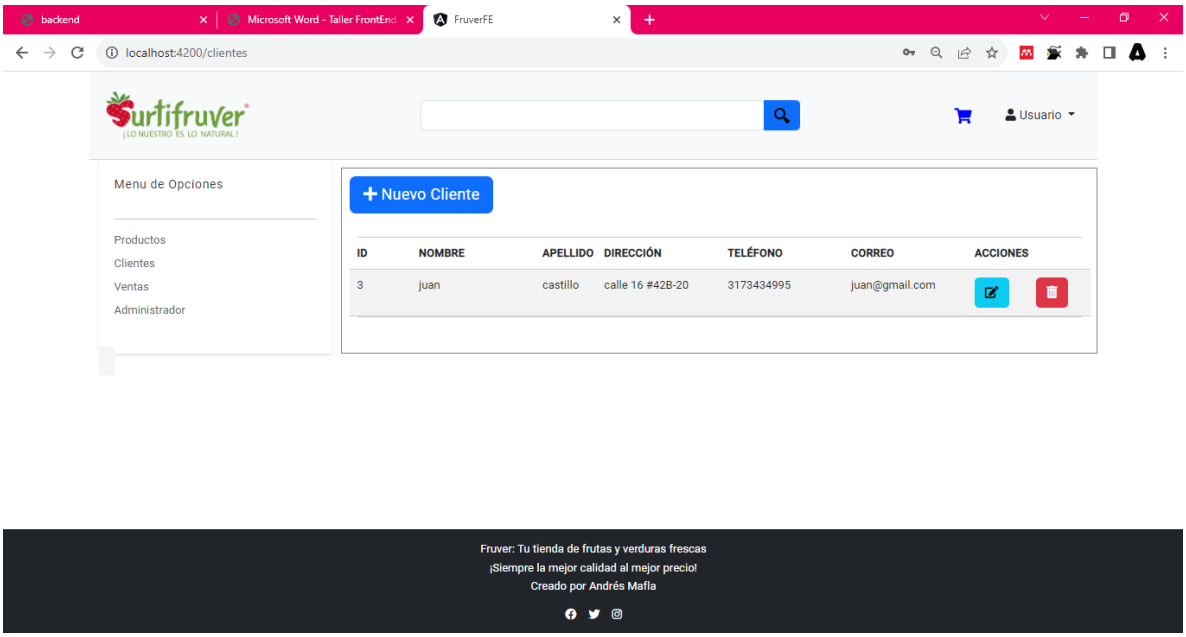
El componente html tiene un botón "Nuevo Cliente" que redirige a una página para agregar un nuevo cliente. Para cada cliente en la tabla, se proporcionan botones de edición y eliminación para actualizar o borrar los clientes. La lista se carga dinámicamente utilizando la directiva *ngFor y el async pipe para obtener datos de manera asíncrona.



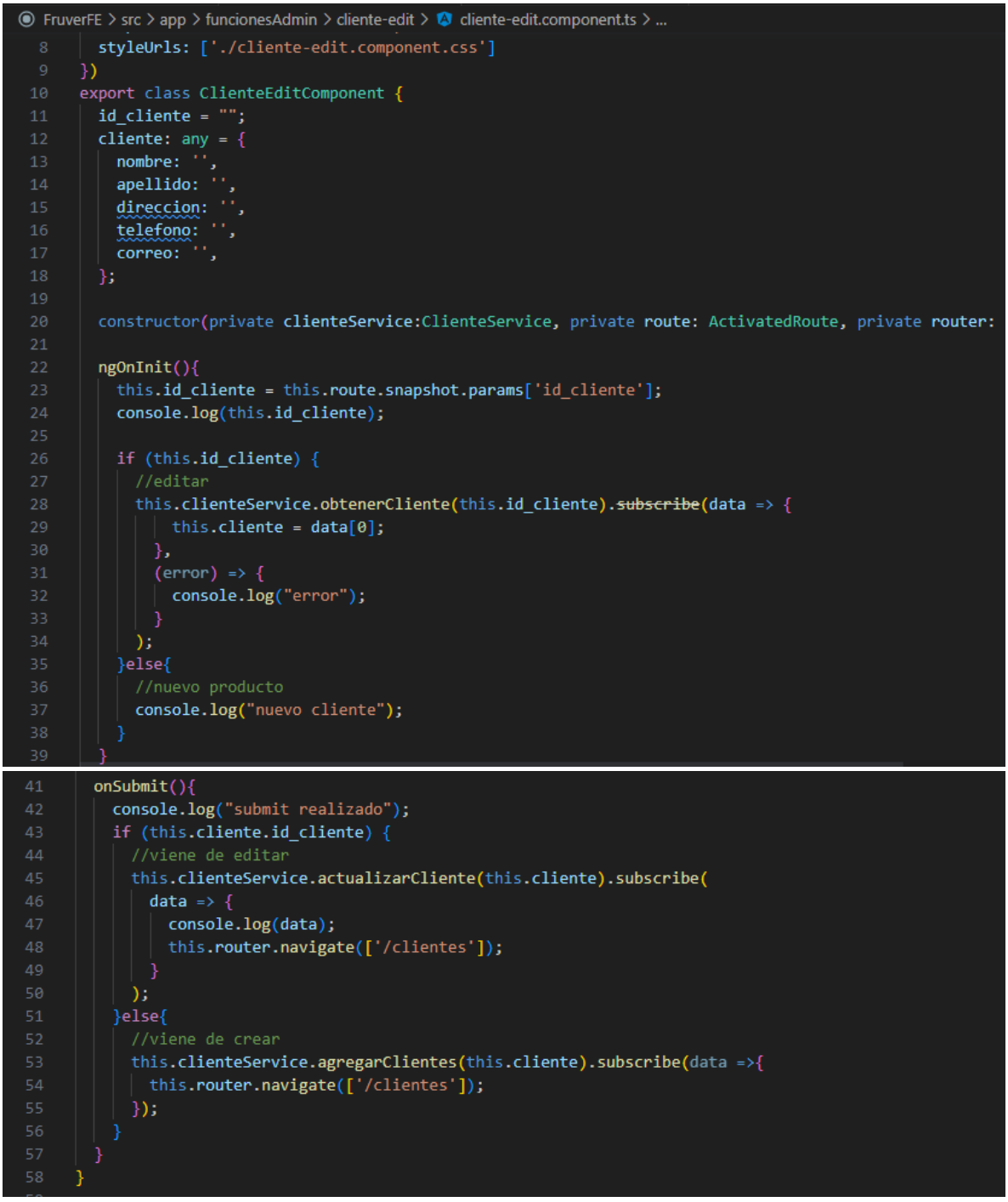
La vista principal de gestionar clientes muestra un botón para agregar un nuevo cliente, más abajo encontramos una tabla con todos los clientes, adicionalmente se agrega una columna donde podemos encontrar los botones de editar y eliminar.



Para eliminar un cliente, simplemente se da clic en el botón "Eliminar", tal como se muestra a continuación. En este ejemplo, procederemos a eliminar el cliente "pedro".

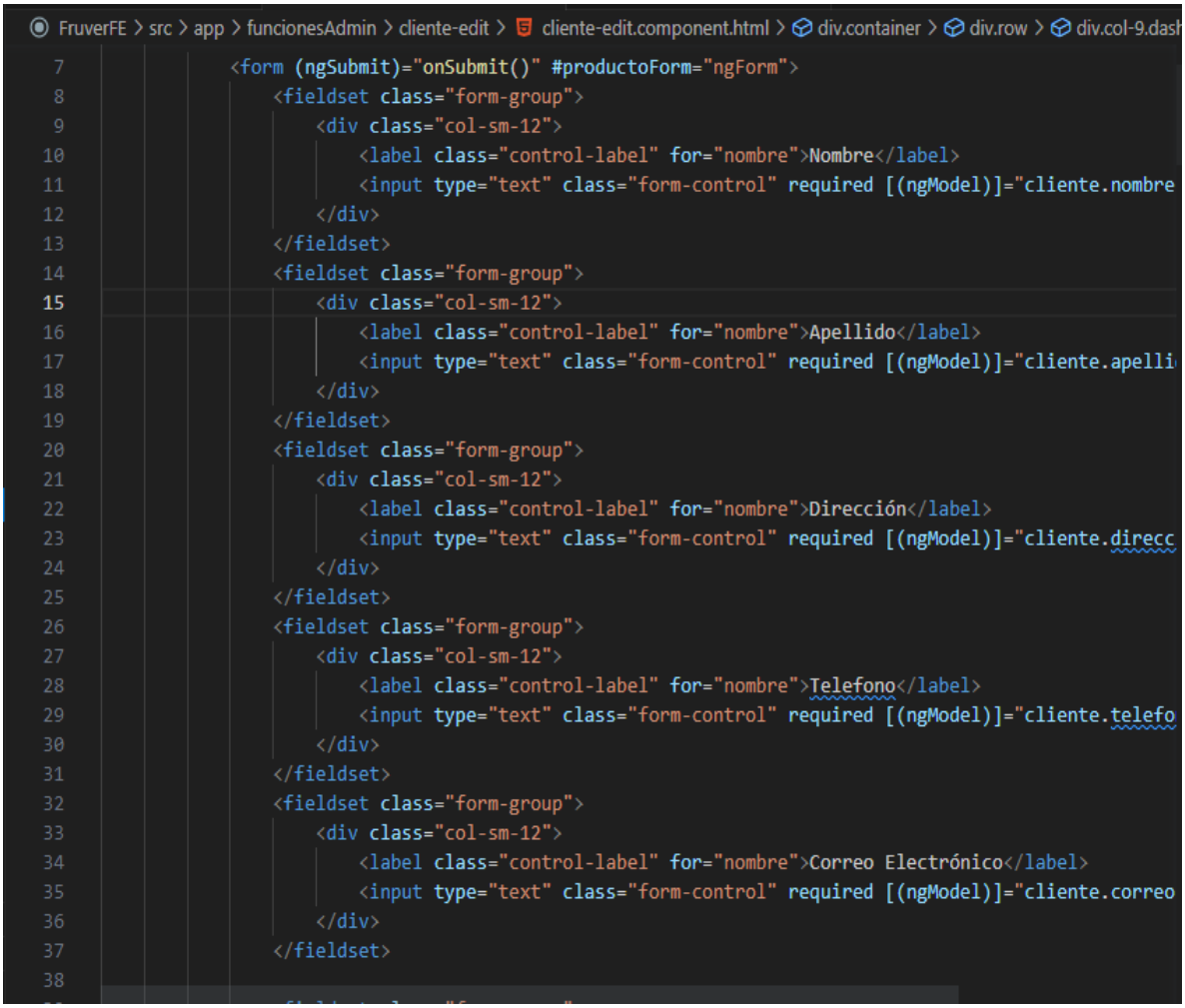


Para implementar la funcionalidad del botón editar y crear nuevo cliente, se utilizó en componente de cliente-edit. Aquí se implementó el método onSubmit() se activa cuando el usuario envía el formulario. Dependiendo de si el cliente ya tiene un id_cliente o no, se llama al método actualizarCliente() o agregarCliente() del servicio ClienteService. Estos métodos realizan solicitudes al servidor para guardar los datos del cliente editado o nuevo. Si la operación se realiza con éxito, el componente redirecciona al usuario a la lista de clientes.

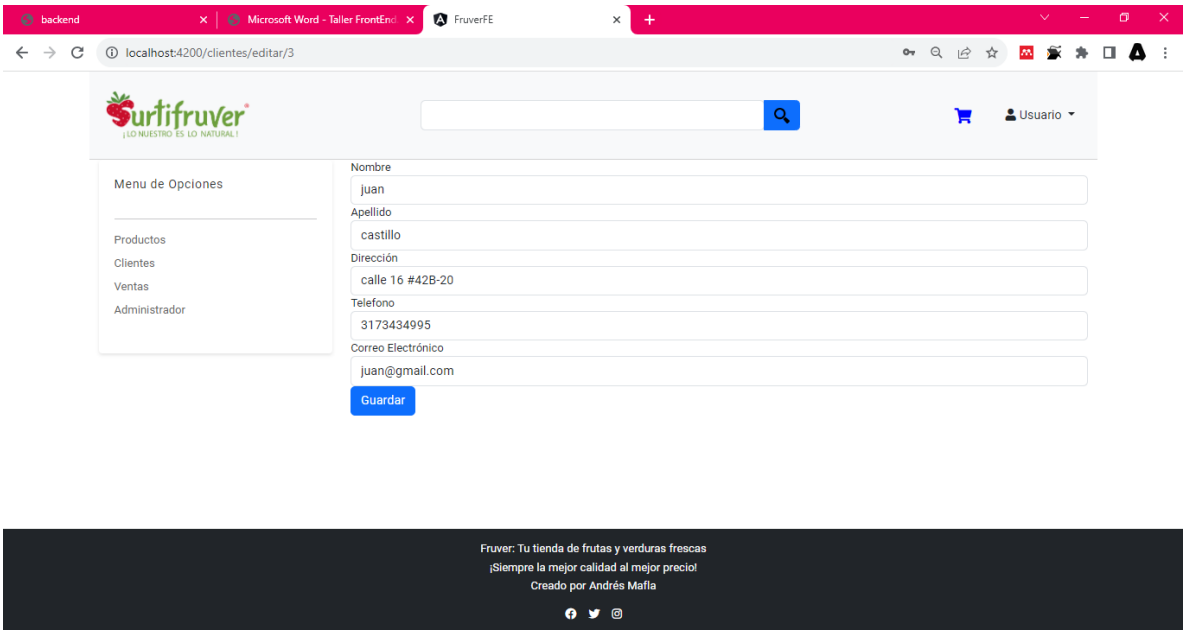


```
FruverFE > src > app > funcionesAdmin > cliente-edit > cliente-edit.component.ts > ...
  styleUrls: ['./cliente-edit.component.css']
})
export class ClienteEditComponent {
  id_cliente = "";
  cliente: any = {
    nombre: '',
    apellido: '',
    direccion: '',
    telefono: '',
    correo: '',
  };

  constructor(private clienteService:ClienteService, private route: ActivatedRoute, private router:

  ngOnInit(){
    this.id_cliente = this.route.snapshot.params['id_cliente'];
    console.log(this.id_cliente);

    if (this.id_cliente) {
      //editar
      this.clienteService.obtenerCliente(this.id_cliente).subscribe(data => {
          this.cliente = data[0];
        },
        (error) => {
          console.log("error");
        }
      );
    }else{
      //nuevo producto
      console.log("nuevo cliente");
    }
  }
  onSubmit(){
    console.log("submit realizado");
    if (this.cliente.id_cliente) {
      //viene de editar
      this.clienteService.actualizarCliente(this.cliente).subscribe(
        data => {
          console.log(data);
          this.router.navigate(['/clientes']);
        }
      );
    }else{
      //viene de crear
      this.clienteService.agregarClientes(this.cliente).subscribe(data =>{
        this.router.navigate(['/clientes']);
      });
    }
  }
}
```

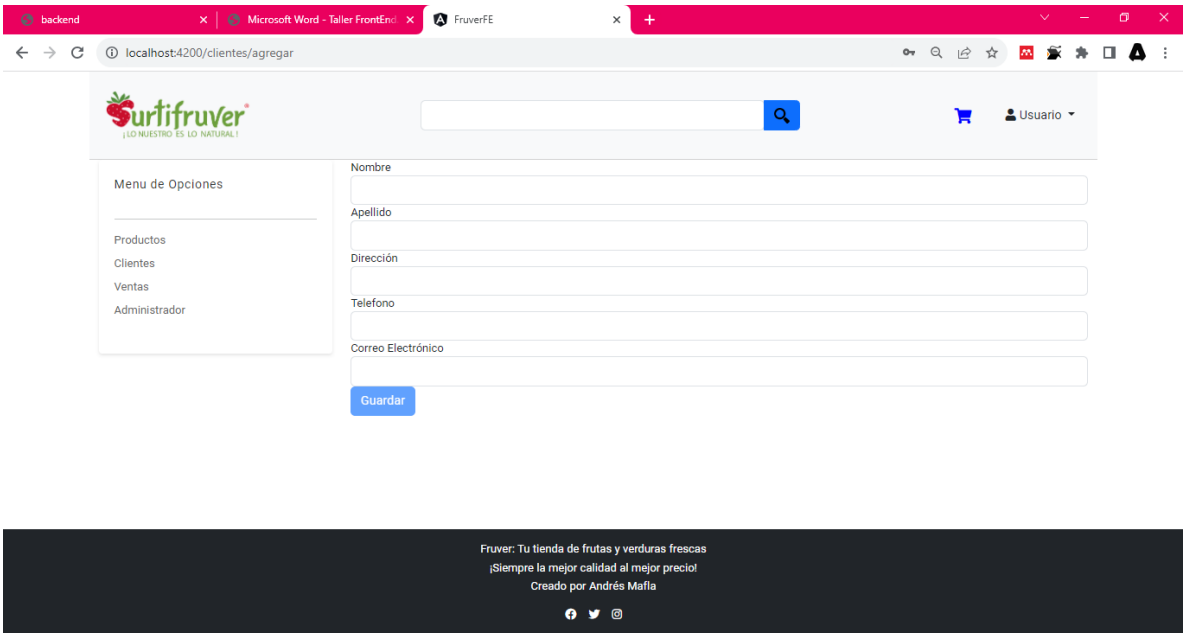
El componente “cliente-edit” html presenta una vista de formulario para editar o crear cliente. La página se divide en dos columnas, una muestra un menú de navegación y la otra contiene el formulario. Los campos "Nombre", "Apellido", "Dirección", "Teléfono" y "Correo", son enlazados con las propiedades del componente, manteniendo así su sincronización. El botón "Guardar" solo está habilitado si el formulario es válido. Al enviar el formulario, se activa el método onSubmit(), encargado de procesar y guardar los datos del cliente.



Cunado damos clic en editar se carga el formulario con la información respectivamente.



Cuando damos clic en nuevo cliente, se cargar el formulario con los campos vacíos.

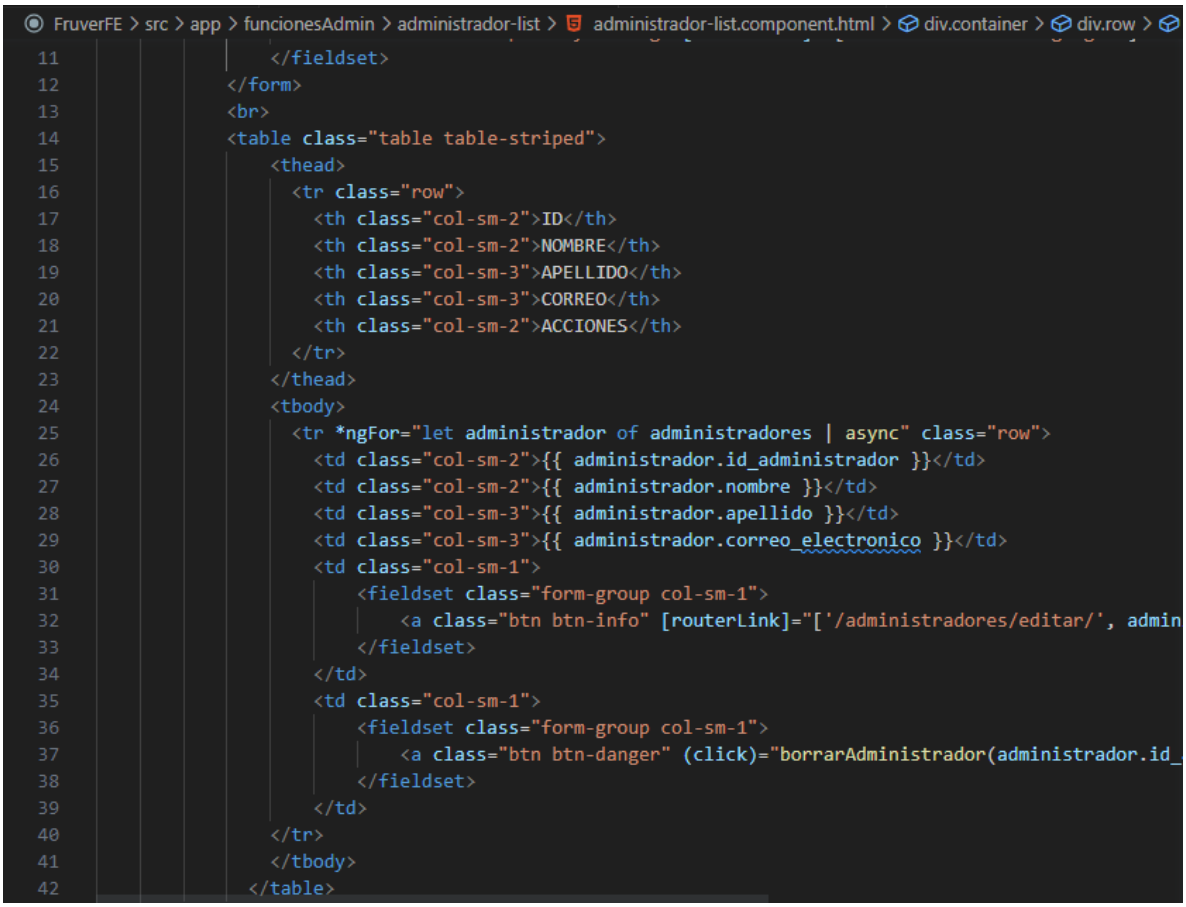


Para gestionar los administradores se creó los componentes “administrador-list” y “administrador-edit”, el primer componente muestra una lista de administradores y permite eliminarlos. Utiliza el servicio AdministradorService para obtener y eliminar administradores. Al iniciar se carga una lista. Cuando se elimina un dato, se actualiza automáticamente la lista.

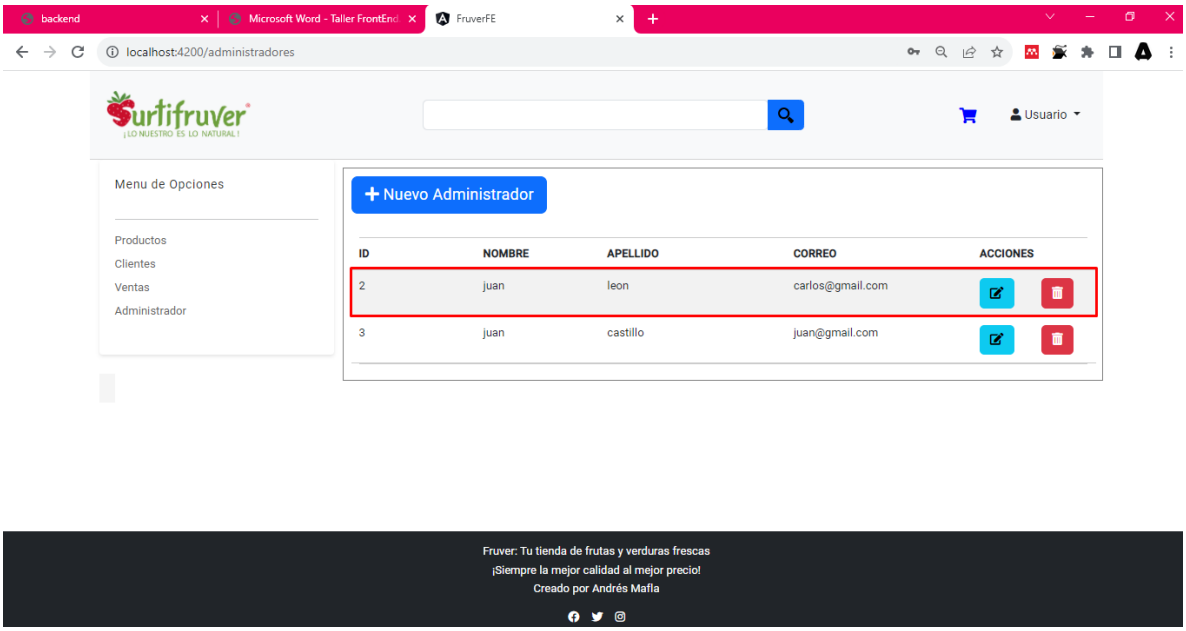
FruverFE > src > app > funcionesAdmin > administrador-list > administrador-list.component.ts > AdministradorListCompone

```ts
import { Component } from '@angular/core';
import { Observable } from 'rxjs';
import { Administrador } from 'src/app/models/administrador';
import { AdministradorService } from 'src/app/services/auth/administrador.service';

@Component({
  selector: 'app-administrador-list',
  templateUrl: './administrador-list.component.html',
  styleUrls: ['./administrador-list.component.css']
})
export class AdministradorListComponent {
  administradores: Observable<Administrador[]> | undefined;

  constructor(private administradorService: AdministradorService){ }

  ngOnInit(){
    this.administradores = this.administradorService.obtenerAdministradores();
  }

  borrarAdministrador(id_administrador:string){
    this.administradorService.borrarAdministrador(id_administrador).subscribe(data =>{
      console.log("Registro Eliminado");
      this.ngOnInit();
    });
  }
}

```

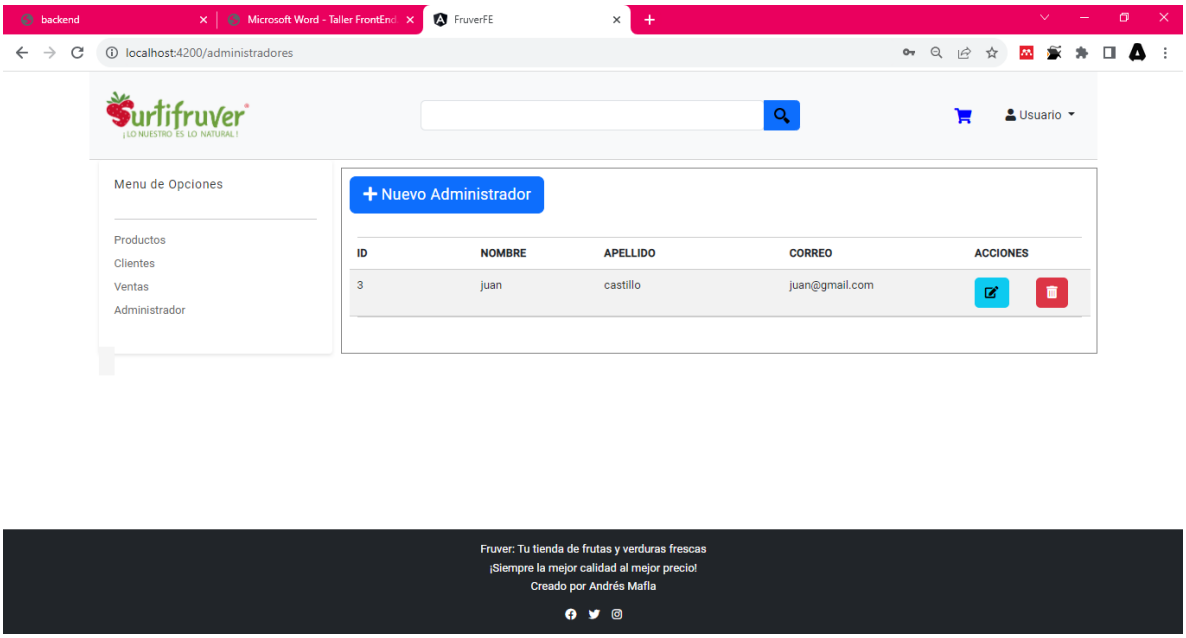
El componente html tiene un botón "Nuevo Administrador" que redirige a una página para agregar un nuevo administrador. Para cada dato en la tabla, se proporcionan botones de edición y eliminación para actualizar o borrar los administradores. La lista se carga dinámicamente utilizando la directiva *ngFor y el async pipe para obtener datos de manera asíncrona.



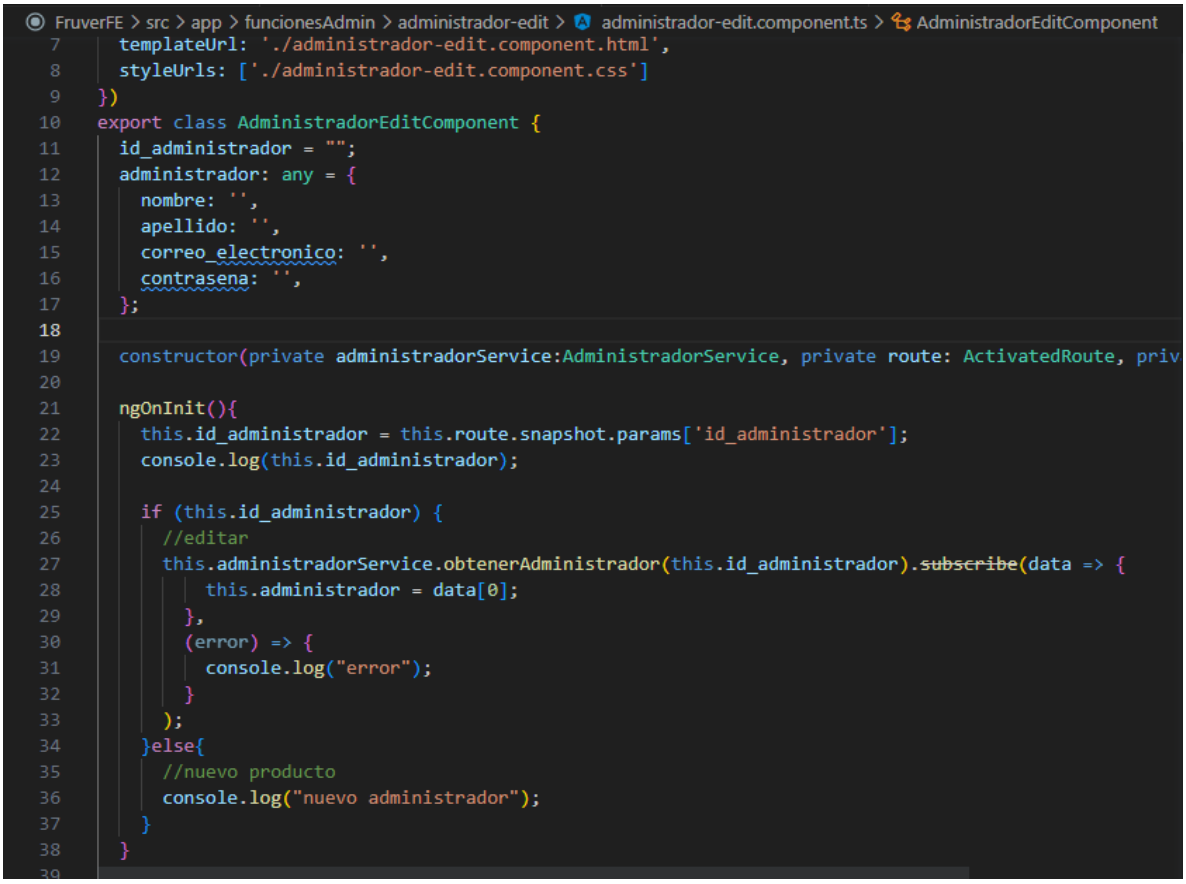
La vista principal de gestionar administradores muestra un botón para agregar un nuevo administrador, más abajo encontramos una tabla con todos los datos, adicionalmente se agrega una columna donde podemos encontrar los botones de editar y eliminar.



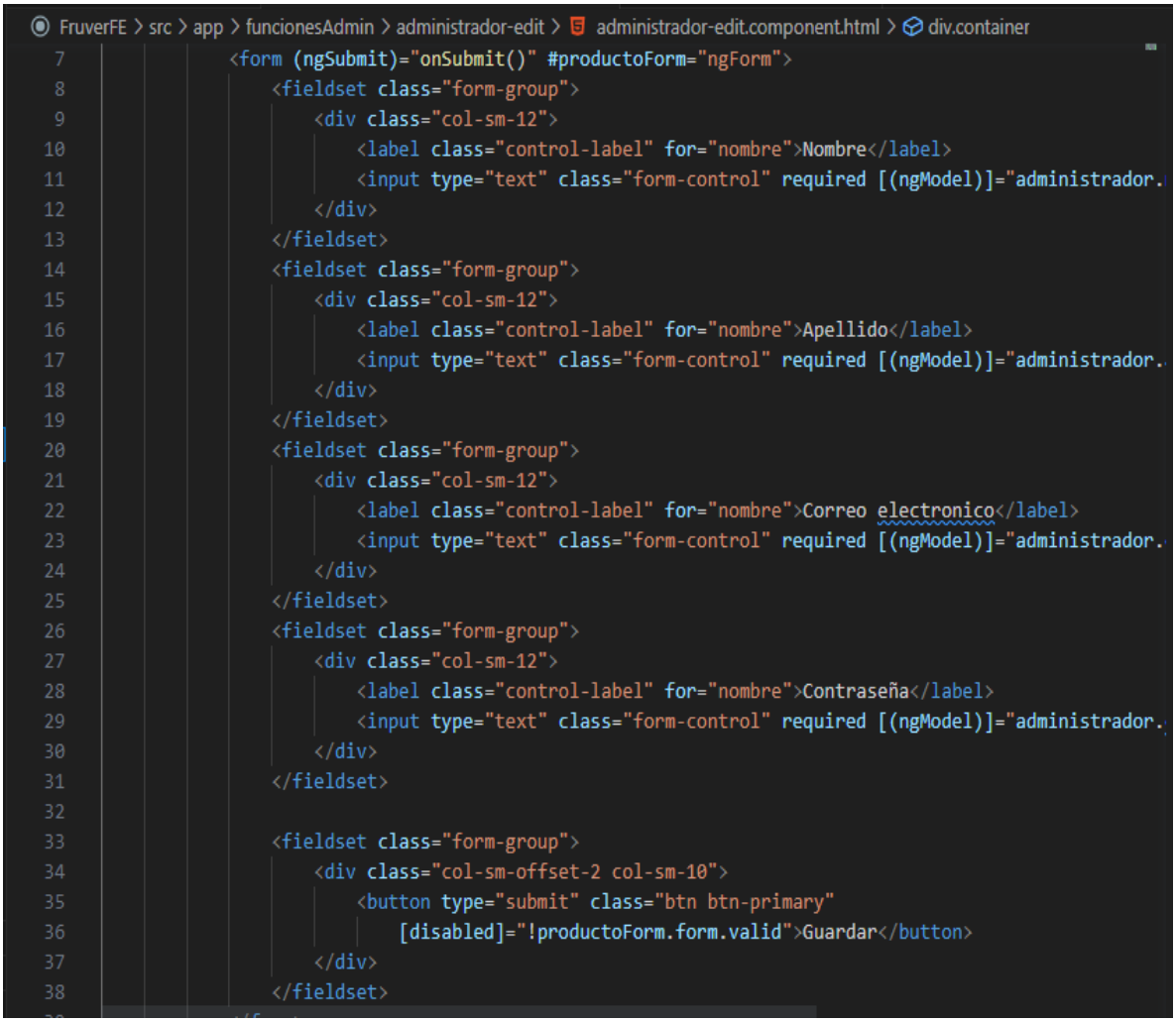
Para eliminar un administrador, simplemente se da clic en el botón "Eliminar", tal como se muestra a continuación. En este ejemplo, procederemos a eliminar el dato de "juan leon".



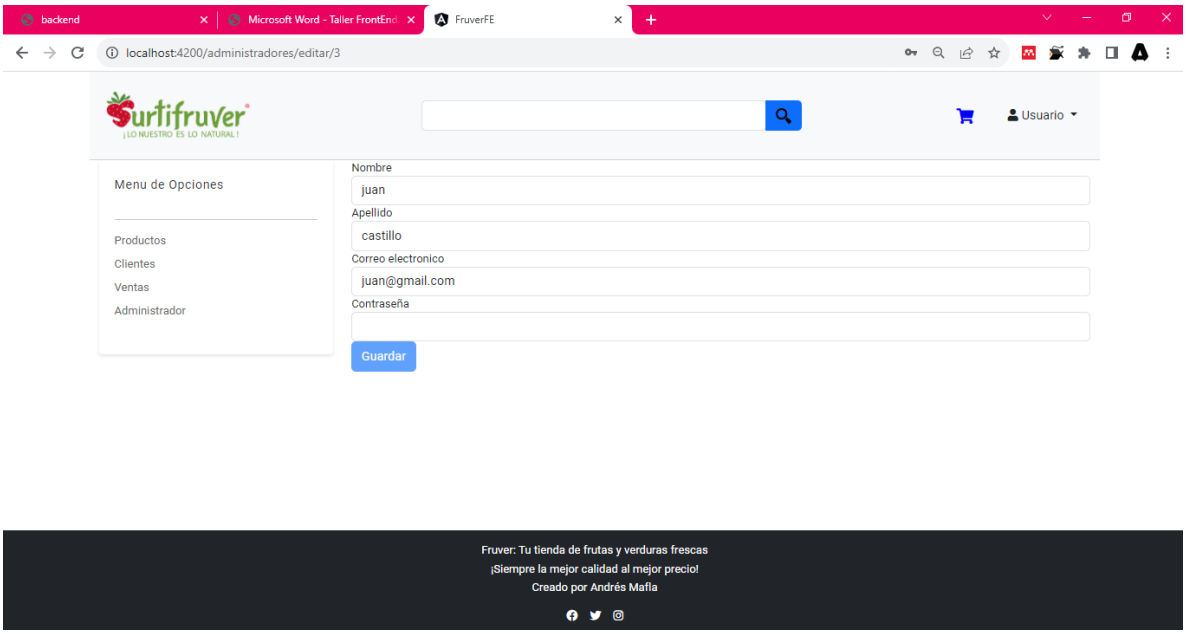
Para implementar la funcionalidad del botón editar y crear nuevo administrador, se utilizó en componente de administrador-edit. Aquí se implementó el método onSubmit() se activa cuando el usuario envía el formulario. Dependiendo de si el administrador ya tiene un id_administrador o no, se llama al método actualizarAdministrador() o agregarAdministrador() del servicio AdministradorService. Estos métodos realizan solicitudes al servidor para guardar los datos del administrador editado o nuevo. Si la operación se realiza con éxito, el componente redirecciona al usuario a la lista de administradores.



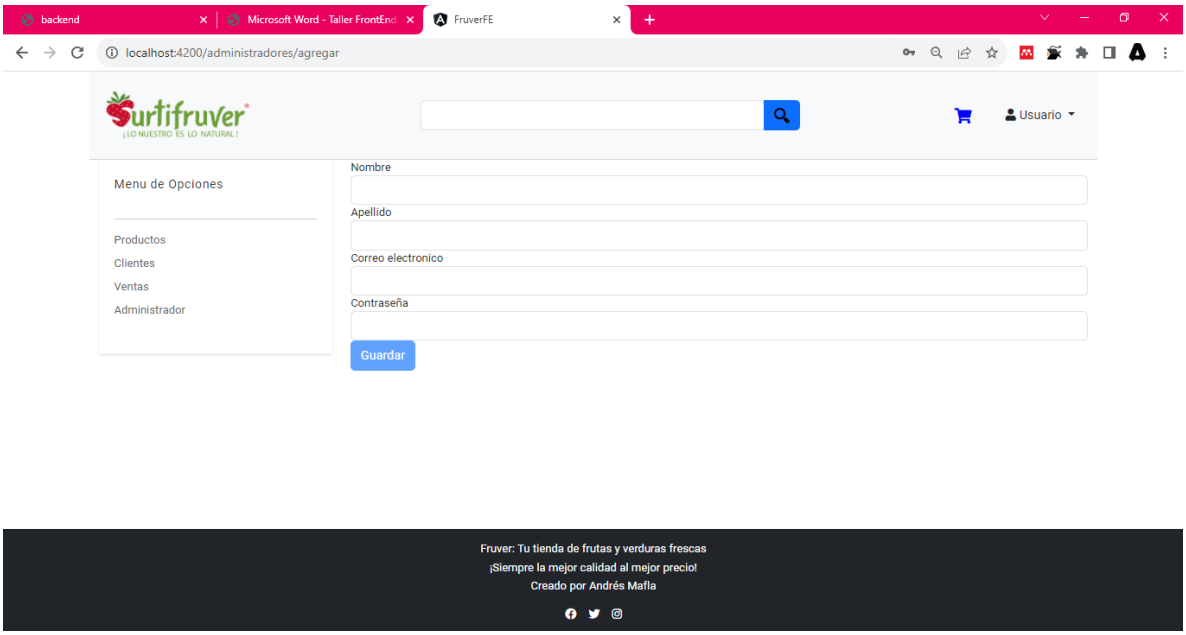
El componente “administrador-edit” html presenta una vista de formulario para editar o crear administrador. La página se divide en dos columnas, una muestra un menú de navegación y la otra contiene el formulario. Los campos "Nombre", "Apellido", "Correo” y "Contraseña", son enlazados con las propiedades del componente, manteniendo así su sincronización. El botón "Guardar" solo está habilitado si el formulario es válido. Al enviar el formulario, se activa el método onSubmit(), encargado de procesar y guardar los datos del administrador.



Cunado damos clic en editar se carga el formulario con la información respectivamente.



Cuando damos clic en nuevo cliente, se cargar el formulario con los campos vacíos.



Nota: Culminación Incompleta del Proyecto

Lamentablemente, debido a limitaciones de tiempo y conocimientos, no fue posible completar el proyecto. Agradezco su comprensión.