

TALLER UNIDAD 3 FRONTEND

BRAYAN DANIEL CERON PORTILLA

UNIVERSIDAD DE NARIÑO

1. Modelos

Los modelos utilizados en el lado FrontEnd son en esencia los mismos a los usados en el BackEnd(con algunas diferencias), esto debido a que el modelado de datos y la lógica de negocios es la misma en ambos entornos. Los modelos codificados fueron ClieteModel, ProductoModel, PedidoModel, DetallePedidoModel y PedidoEnviarModel.

Modelo Cliente

```
export class ClienteModel{
  constructor(
    public idCliente:string,
    public nombres:string,
    public apellidos:string,
    public correo:string,
    public contrasena:string,
    public rol:string){}
}
```

Modelo productos

```
export class ProductoModel {
  constructor(
    public idProducto: string,
    public nombre: string,
    public detalle:string,
    public valor: number,
    public imagen:string) { }
```

Modelo Pedido

```
import { ClienteModel } from "../cliente.model";
import { DetallePedidoModel } from "../detallepedido.model";

export class PedidoModel{
  constructor(
    public idPedido:string,
    public createdAt:string,
    public total:number,
    public procesado:Boolean,
    public idCliente:string,
    public detallePedidos:DetallePedidoModel[],
    public cliente:ClienteModel){}
}
```

Modelo detalle pedido

```
export class DetallePedidoModel{
  constructor(
    public id:string,
    public cantidadProducto:number,
    public totalProducto:number,
    public idPedido:string,
    public idProducto:string){}
}
```

Modelo enviar pedido

```
export class PedidoEnviarModel{
  constructor(
    public idProducto:string,
    public cantidadProducto:number){}
}
```

2. Servicios

Los servicios utilizados para conectar el FrontEnd con los endpoints expuestos por la api en el BackEnd, corresponden al CRUD para los modelos más importantes dentro de la lógica del negocio, es decir los modelos para productos, clientes y pedidos.

Servicio Cliente

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { ClienteModel } from '../cliente.model';
import { SERVER_BASE_URL, getHeaders } from '../shared/utils';

@Injectable({
  providedIn: 'root'
})
export class ClienteService {
  BASE_URL = SERVER_BASE_URL;
  constructor(private http: HttpClient) { }

  obtenerCliente(idCliente: string) {
    return this.http.get<ClienteModel[]>(`${this.BASE_URL}/cliente/${idCliente}`, getHeaders());
  }
  agregarCliente(cliente: ClienteModel){
    return this.http.post<string>(`${this.BASE_URL}/clientes`, cliente);
  }
  borrarCliente(idCliente: string) {
    return this.http.delete<string>(`${this.BASE_URL}/clientes/${idCliente}`, getHeaders());
  }
  actualizarCliente(cliente: ClienteModel) {
    return this.http.put<string>(`${this.BASE_URL}/clientes/${cliente.idCliente}`, cliente, getHeaders());
  }

  //AUTENTICACION

  login(correo: string, contrasena: string) {
    return this.http.post<any>(`${this.BASE_URL}/login`, { correo, contrasena }, { observe: 'response' });
  }

  verificarToken(){
    return this.http.get<any>(`${this.BASE_URL}/verificarToken`, getHeaders());
    //return this.http.get<any>(`${this.BASE_URL}/verificarToken`, getHeaders()).toPromise();
  }
  getRol(){
    return this.http.get<any>(`${this.BASE_URL}/getRol`, getHeaders())
  }
}
```

Servicio Producto

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { ProductoModel } from '../producto.model';
import { SERVER_BASE_URL, getHeaders } from '../shared/utills';

@Injectable({
  providedIn: 'root'
})
export class ProductoService {
  BASE_URL = SERVER_BASE_URL;
  constructor(private http: HttpClient) { }

  obtenerProductos() {
    return this.http.get<ProductoModel[]>(`${this.BASE_URL}/productos`, getHeaders());
  }

  obtenerProducto(idProducto: string) {
    return this.http.get<ProductoModel[]>(`${this.BASE_URL}/producto/${idProducto}`, getHeaders());
  }

  agregarProducto(producto: FormData) {
    return this.http.post<string>(`${this.BASE_URL}/productos`, producto, getHeaders());
  }

  actualizarProducto(producto: FormData) {
    return this.http.put<string>(`${this.BASE_URL}/productos/${producto.get('idProducto')}`, producto, getHeaders());
  }
  borrarProducto(idProducto: string) {
    return this.http.delete<string>(`${this.BASE_URL}/productos/${idProducto}`, getHeaders());
  }
}
```

Servicio Pedido

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { PedidoModel } from '../pedido.model';
import { PedidoEnviarModel } from '../pedidoenviar.model';
import { SERVER_BASE_URL, getHeaders } from '../shared/utills';

@Injectable({
  providedIn: 'root'
})
export class PedidoService {
  BASE_URL = SERVER_BASE_URL;
  constructor(private http: HttpClient) { }

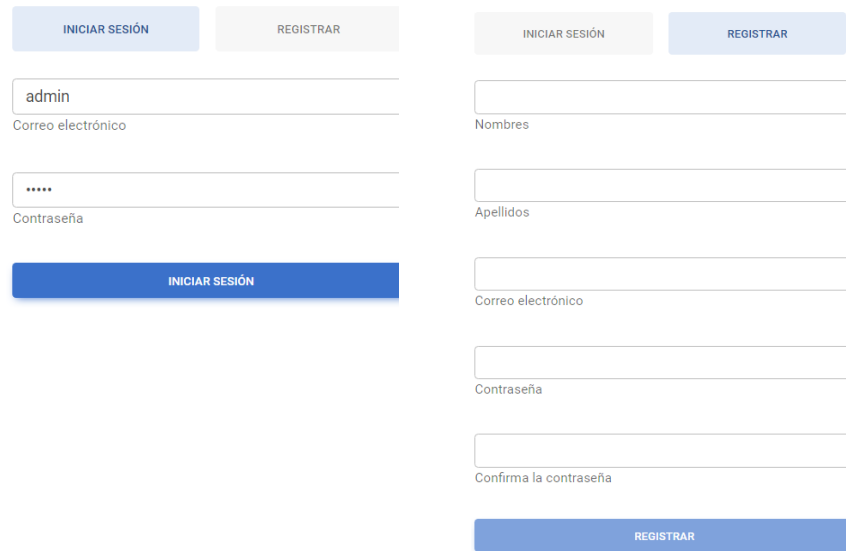
  obtenerPedido(idPedido: string) {
    return this.http.get<PedidoModel>(`${this.BASE_URL}/pedido/${idPedido}`, getHeaders());
  }
  obtenerPedidos() {
    return this.http.get<PedidoModel[]>(`${this.BASE_URL}/pedidos`, getHeaders());
  }
  obtenerPedidosCliente(idCliente: string) {
    return this.http.get<PedidoModel[]>(`${this.BASE_URL}/pedidos/cliente/${idCliente}`, getHeaders());
  }
  agregarPedido(idCliente: string, carrito: PedidoEnviarModel[]) { //idCliente debe ser el de que esta Logeado
    return this.http.post<string>(`${this.BASE_URL}/pedidos`, {idCliente: idCliente, productos: carrito}, getHeaders());
  }
  actualizarPedido(idPedido: string, carrito: PedidoEnviarModel[], procesado: boolean) { //idCliente debe ser el de que esta Logeado
    return this.http.put<string>(`${this.BASE_URL}/pedidos/${idPedido}`, {productos: carrito, procesado}, getHeaders());
  }
  borrarPedido(idPedido: string) {
    return this.http.delete<string>(`${this.BASE_URL}/pedidos/${idPedido}`, getHeaders());
  }
  procesarPedido(idPedido: string) { //idCliente debe ser el de que esta Logeado
    return this.http.put<string>(`${this.BASE_URL}/pedidos/procesar/${idPedido}`, {}, getHeaders());
  }
}
```

3. Componentes

Componente autenticación:

Este componente es el encargado de proporcionar la interfaz gráfica de usuario y lógica del lado FrontEnd, correspondiente a las funcionalidades para el inicio de sesión, autenticación y registro de usuarios

Interfaz gráfica autenticación



Lógica del componente:

Para el envío de la información se hace uso del servicio “ClienteService”, el cual mediante sus métodos “iniciarSesion” y “agregarCliente” se conecta a la api del Backend para obtener el token de acceso y para registrar un nuevo usuario respectivamente

```
export class AutenticacionComponent {
  correo: string = 'admin';
  contrasena: string = 'admin';

  nombres: string = '';
  apellidos: string = '';
  correo_registro: string = '';
  contrasena_registro: string = '';
  contrasena_registro2: string = '';

  errorLogin:string='';
  errorRegistrar:string='';

  constructor(private clienteService: ClienteService, private router: Router) { }

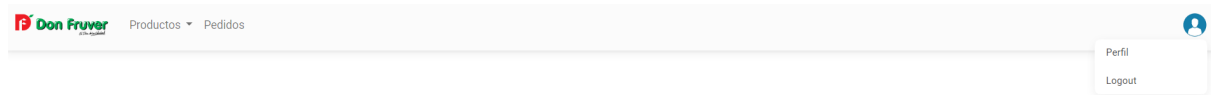
  async iniciarSesion() {
    this.clienteService.login(this.correo, this.contrasena).subscribe({
      next: data => {
        localStorage.setItem("token", data.body.token);
        this.router.navigate(['/productos']);
      },
      error: error => {this.errorLogin=error.error.msg;}
    });
  }

  registrar() {
    if (this.contrasena_registro == this.contrasena_registro2) {
      let newCliente = new ClienteModel("", this.nombres, this.apellidos, this.correo_registro, this.contrasena_registro, "u");
      this.clienteService.agregarCliente(newCliente).subscribe({
        next: data => {
          this.correo=this.correo_registro;
          this.contrasena=this.contrasena_registro;
          this.iniciarSesion(),
        },
        error: error => {this.errorRegistrar=error.error.mensaje;}
      });
    }
    else {alert("Las contraseñas no coinciden"); }
  }
}
```

Componente navbar

Este componente es el encargado de proporcionar la interfaz gráfica de usuario y lógica del lado FrontEnd, correspondiente a las funcionalidades para la navegación de la aplicación por todos los demás componentes:

Interfaz gráfica navbar



Lógica del componente:

Este componente debe proporcionar la navegabilidad por toda la aplicación, básicamente su importancia radica en habilitar o deshabilitar las opciones que permiten el acceso a las demás rutas de acuerdo al rol del usuario que ha iniciado sesión. Para ello este componente hace uso del servicio “ClienteService”, el cual permite obtener toda la información relacionada con los usuarios para posteriormente permitir o denegar el acceso a los recursos.

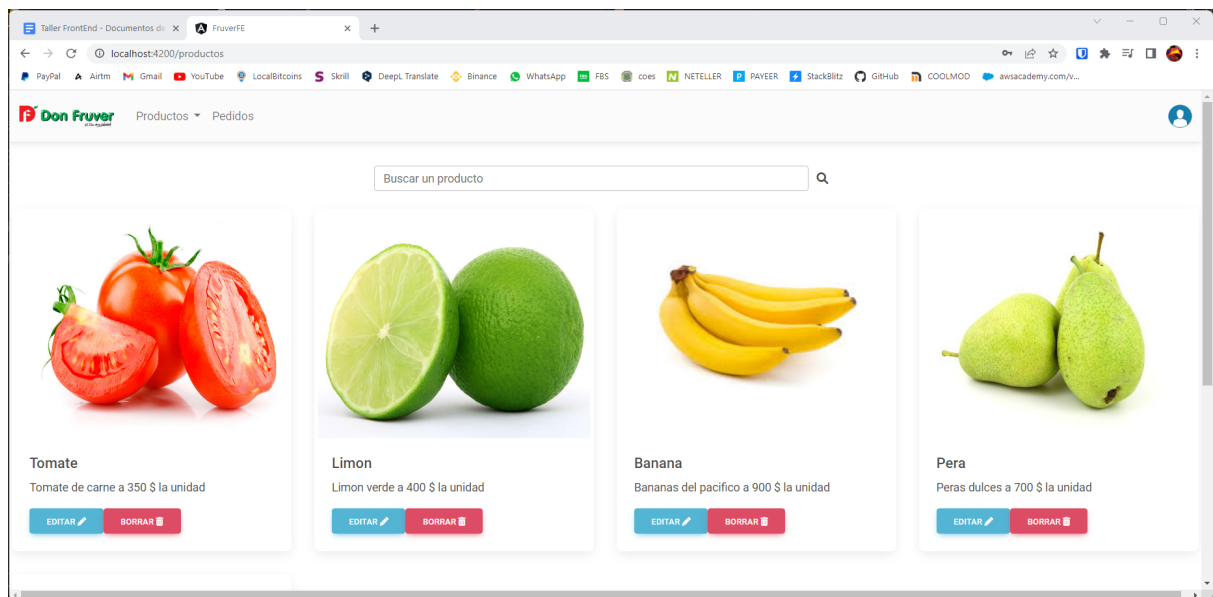
```
export class NavbarComponent implements OnInit{
  rol:string='';
  constructor(private router: Router,private clienteService:ClienteService){}
  ngOnInit() {
    //Estableciendo el rol
    this.clienteService.getRol().subscribe({
      next: data=>{this.rol=data.rol},
      error: error=>{validarRol(error);console.log(error)}
    });
  }

  cerrarSesion(){
    localStorage.removeItem("token");
    this.router.navigate(["/login"]);
  }
}
```

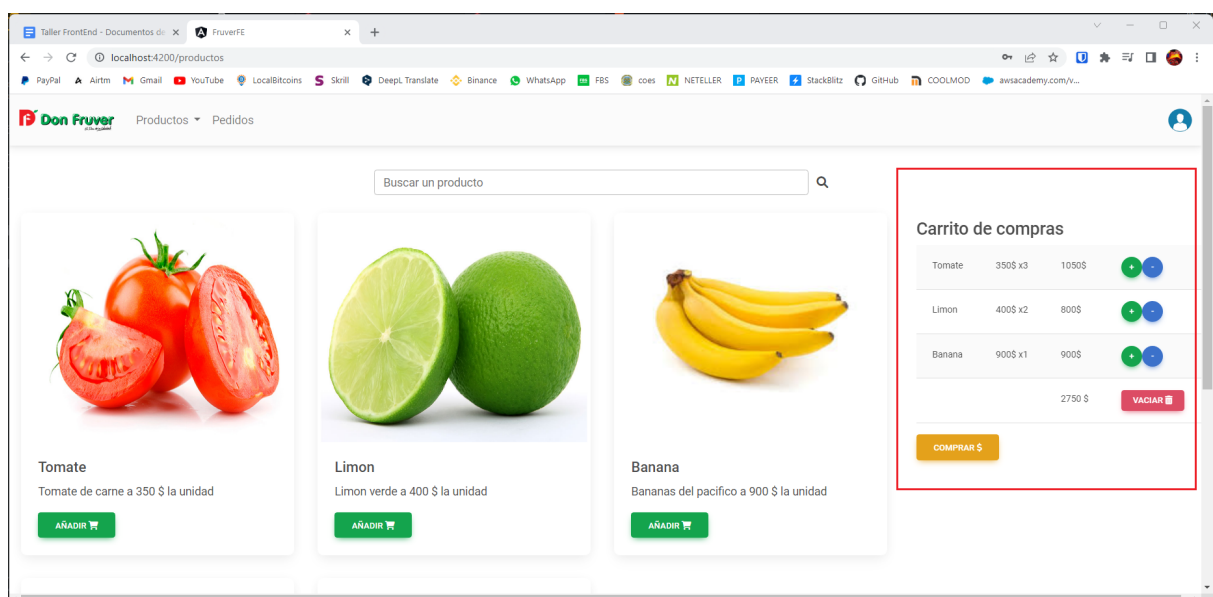
Componente lista productos

Este componente es el encargado de proporcionar la interfaz gráfica de usuario y lógica del lado FrontEnd, correspondiente a las funcionalidades para la visualización y eliminación de productos, y la generación y edición de pedidos. Las tareas que se pueden realizar en este componente varían de acuerdo al usuario que ha iniciado sesión, en ese sentido, se establece que el administrador es el único que puede editar y eliminar productos, mientras que por su parte el cliente lo único que puede hacer es generar y editar pedidos

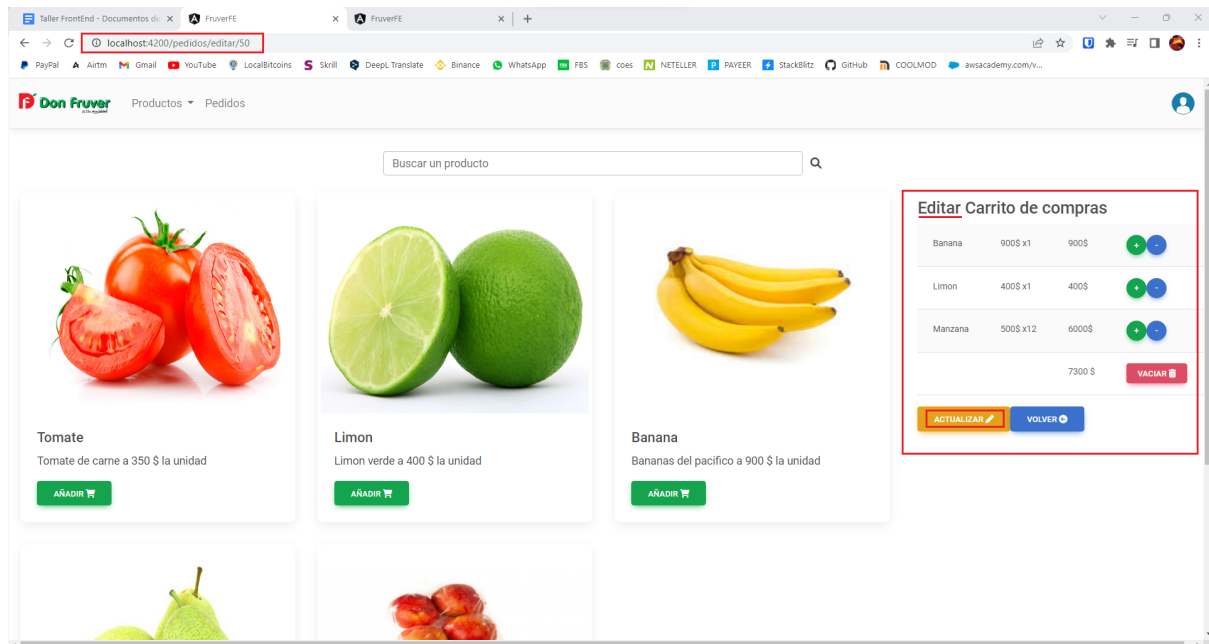
Interfaz gráfica listar productos para el rol administrador



Interfaz gráfica listar productos para el rol cliente



Interfaz gráfica listar productos para editar un pedido (rol cliente)



Lógica del componente:

Para inicializar el componente con información preprocesada implementamos la interfaz “OnInit”, en este caso usaremos esta interfaz para cargar los productos disponibles, los productos del carrito de compras almacenados en el localStorage y para obtener los datos del usuario que ha iniciado sesión. Para mostrar los productos utilizamos el servicio “ProductoService”, y mediante su método “obtenerProductos” obtenemos todos los productos que han sido registrado. Para establecer el rol de usuario utilizamos el servicio “ClienteService”, el cual mediante su método “getRol” se conecta a la api del Backend para obtener el rol y el id de un usuario, para finalmente determinar si el usuario tiene autorización para acceder al recurso.

Si el usuario que inicia sesion es el administrador, solamente es necesario carga los productos registrados(habilitando las opciones del rol administrador), si por el contrario el usuario que inicio sesion es un cliente, hay que determinar en primer lugar si el componente será utilizado para crear o editar un pedido; esto lo determinamos mediante el paso o no pasó del id de un pedido en la url. En el caso de que el componente se use para crear un pedido se cargará el carrito de compras vacío, caso contrario se cargará en el carrito los productos registrados en ese pedido.


```

ngOnInit() {
  this.productos = this.productoService.obtenerProductos();
  this.productos.subscribe({error: error => {validarAutorizacion(error);console.log(error)}});

  this.idPedido = this.route.snapshot.params['idPedido'];

  //Estableciendo el rol
  this.clienteService.getRol().subscribe({
    next: data=>{
      this.rol=data.rol;
      this.idCliente = data.idCliente

      if(this.idPedido){//Editar Pedido
        this.pedidoService.obtenerPedido(this.idPedido).subscribe({
          next: data=>{
            if(!data){this.router.navigate(["~/pedidos"]);return;} //validando que el pedido exista o no este vacio

            this.pedidoProcesado=data.procesado;
            let carritoEditar: any[] = [];
            data.detallePedidos.forEach((element: any) => {
              let nuevoRegistro = {
                idProducto: element.idProducto,
                nombre: element.producto.nombre,
                valor: element.producto.valor,
                modelo: new PedidoEnviarModel(element.idProducto, element.cantidadProducto)
              }
              carritoEditar.push(nuevoRegistro);
            });

            localStorage.setItem("carritoEditar",JSON.stringify(carritoEditar));
            localStorage.getItem("carritoEditar")?this.carrito=JSON.parse(localStorage.getItem('carritoEditar')):this.carrito;
            this.calcluarTotalCarrito();
            if(this.idCliente!=data.idCliente){this.router.navigate(["~/pedidos"]);return;} //validando que el pedido pertenezca al cliente
          },
          error: error=>{validarAutorizacion(error);}
        });
      }
    }
  });
  else{//Listar productos
    localStorage.getItem('carrito'+this.idCliente)?this.carrito=JSON.parse(localStorage.getItem('carrito'+this.idCliente)):this.carrito;
    this.calcluarTotalCarrito();
  }
},
error: error=>{validarRol(error);}
});
}

```

Las funcionalidades para el carrito de compras solamente se implementan en el FrontEnd, lo que implica que la información sólo permanece activa en el localStorage del navegador, y solamente cuando el cliente concreta la compra se envía la información del pedido al Backend. Entre las funcionalidades del carrito de compras se encuentra las opciones para añadir un nuevo producto al carrito, aumentar o reducir en uno la cantidad de un producto ya añadido, calcular total de la compra, vaciar el carrito y persistir el carrito en el localStorage

```

anadirAlCarrito(idProducto: string, nombreProducto: string, valor: number,) {
  let b: Boolean = false;
  this.carrito.forEach(p => {
    if (p.idProducto == idProducto) {this.masUno(idProducto);b = true;}
  });
  if (b) return;

  let nuevoRegistro = {
    idProducto: idProducto,
    nombre: nombreProducto,
    valor: valor,
    modelo: new PedidoEnviarModel(idProducto, 1)
  }
  this.carrito.push(nuevoRegistro);
  this.saveCarrito();
}

```

```

masUno(idProducto: string) {
  this.carrito.forEach(p => {
    if (p.idProducto == idProducto) {p.modelo.cantidadProducto++;}
  });
  this.saveCarrito();
}

```

```

menosUno(idProducto: string) {
  let index: number = 0;
  this.carrito.forEach(p => {
    if (p.idProducto == idProducto) {
      if ((p.modelo.cantidadProducto - 1) > 0) {p.modelo.cantidadProducto--;}
      else {this.carrito.splice(index, 1);}
    }
    index++;
  });
  this.saveCarrito();
}

```

```

calcularTotalCarrito() {
  this.totalCarrito = 0;
  this.carrito.forEach(element => {
    this.totalCarrito += element.valor * element.modelo.cantidadProducto;
  });
}

```

```

vaciarCarrito() {
  this.carrito = [];
  this.totalCarrito = 0;
  this.saveCarrito();
}

```

```

saveCarrito(){
  if(!this.idPedido){localStorage.setItem("carrito"+this.idCliente,JSON.stringify(this.carrito));}
  this.calcularTotalCarrito();
}

```

La funcionalidad para eliminar un producto(disponible solamente para el rol administrador) hace uso del servicio “ProductoService”, el cual mediante su método “borrarProducto” se conecta a la api del Backend para borrar un producto mediante un id que se le debe pasar como parámetro.

```
setProductoBorrar(idProducto: string) {this.idProductoBorrar=idProducto;}
borrarProducto() {
  this.productoService.borrarProducto(this.idProductoBorrar).subscribe({
    next: data => {this.ngOnInit();},
    error: error => {
      if(error.status==500){alert(error.error.mensaje);return;}
      else{validarAutorizacion(error);}
    }
  });
}
```

Para formalizar el pedido y confirmar la compra, el componente hace uso del servicio “PedidoService”, al cual se le debe proporcionar el id del cliente que hace el pedido y una instancia en forma de vector del modelo “PedidoEnviarModel”. La función de este modelo es simplemente facilitar una estructura donde únicamente se especifique el id del producto junto con la cantidad comprada, para de esta forma enviar por la red solamente la información necesaria del pedido.

```
comprar() { //crear pedido
  let dataEnviar: PedidoEnviarModel[] = [];
  this.carrito.forEach(element => {dataEnviar.push(element.modelo)});
  this.pedidoService.agregarPedido(this.idCliente,dataEnviar).subscribe({
    next: data => {this.vaciarCarrito();},
    error: error => { validarAutorizacion(error);}
  });
}
```

La funcionalidad para actualizar un pedido (disponible solamente para el rol cliente y para pedidos que únicamente están en estado pendiente) hace uso del servicio “PedidoService”, el cual mediante su método “actualizarPedido” se conecta a la api del Backend para actualizar un pedido. Para utilizar este método es necesario establecer como parámetros el id del pedido a modificar, el vector del modelo “PedidoEnviarModel” con los nuevos cambios (semejante a como se hizo cuando se registró el pedido) y el estado del pedido.

```
actualizarPedido(){
  let dataEnviar: PedidoEnviarModel[] = [];
  this.carrito.forEach(element => {dataEnviar.push(element.modelo)});
  this.pedidoService.actualizarPedido(this.idPedido,dataEnviar,false).subscribe({
    next: data => {this.router.navigate(["/pedidos"]);},
    error: error => {
      if(error.status==409){alert(error.error.mensaje);return;}
      validarAutorizacion(error);
    }
  });
}
```

La funcionalidad para editar un producto realmente no se maneja en este componente, por lo que lo único que hace el botón “EDITAR” es redireccionar al componente “editar productos” el cual realiza toda la lógica para esa operación

Componente editar productos

Este componente es el encargado de proporcionar la interfaz gráfica de usuario y lógica del lado FrontEnd, correspondiente a las funcionalidades para el registro de nuevos productos y para la edición de las propiedades de los productos ya existentes. Las tareas que se pueden realizar en este componente hacen parte de las actividades del rol administrador y solo pueden ser usadas por este rol

Interfaz gráfica para registrar producto

The screenshot shows a web browser window with the URL `localhost:4200/productos/agregar`. The page title is "Registrar Producto". The form contains the following fields:

- Nombre:
- Detalle:
- Valor:
- Archivo: Ninguno archivo selec.

Below the fields is a blue button labeled "ENVIAR". The browser's address bar shows the URL, and the top navigation bar includes the "Don Fruver" logo and links to "Productos" and "Pedidos".

Interfaz gráfica para editar producto

The screenshot shows a web browser window with the URL `localhost:4200/productos/editar/1`. The page title is "Editar Producto". The form contains the following fields:

- Nombre:
- Detalle:
- Valor:
- Archivo: tomate.jpg

Below the fields is a blue button labeled "ENVIAR". The browser's address bar shows the URL, and the top navigation bar includes the "Don Fruver" logo and links to "Productos" and "Pedidos".

Lógica del componente:

Este componente es utilizado tanto para editar como para crear un producto, por lo que antes de hacer la carga del componente se debe establecer para qué va ser utilizado; esto lo determinamos mediante el paso o no pasó del id de un producto en la url. En el caso de que el componente se use para crear un producto se cargará el formulario vacío, caso contrario se cargará en el formulario las propiedades del producto a editar.

Para inicializar el componente con información preprocesada implementamos la interfaz “OnInit”, en este caso usaremos esta interfaz para saber cómo utilizaremos el componente (crear o editar un producto) y para cargar la información de un producto (mediante el servicio “ProductoService”) en caso de que se lo utilice para editar.

```
ngOnInit() {
  this.idProducto = this.route.snapshot.params['idProducto'];

  if (this.idProducto) { //Viene de Editar
    this.encabezado="Editar Producto";
    this.productoService.obtenerProducto(this.idProducto).subscribe({
      next: data => {this.producto = data[0];},
      error: error => {validarAutorizacion(error); /*console.log(error);*/}
    });
  }

  this.clienteService.getRol().subscribe({ //Estableciendo el rol
    next: data => {
      this.rol = data.rol;
      if (this.rol == "user") { this.router.navigate(['/productos']); },
      error: error => { validarRol(error); /*console.log(error);*/ }
    });
  }
}
```

Para el envío de la información tanto como para crear o editar un producto se utiliza en este caso el objeto FormData, esto es debido a que tenemos que adjuntar archivos y no podemos mandarlo en formato json. Para hacer efectivo el envío de la información se hace uso del servicio “ProductoService”, utilizando sus métodos para “actualizarProducto” y “agregarProducto” según sea el caso

```
onArchivoSeleccionado(e:any){this.archivoSeleccionado=e.target.files[0];/*console.log(e);*/}
onSubmit() {
  let fd = new FormData();
  fd.append('nombre', this.producto.nombre);
  fd.append('detalle', this.producto.detalle);
  fd.append('valor', this.producto.valor.toString());
  fd.append('file',this.archivoSeleccionado);

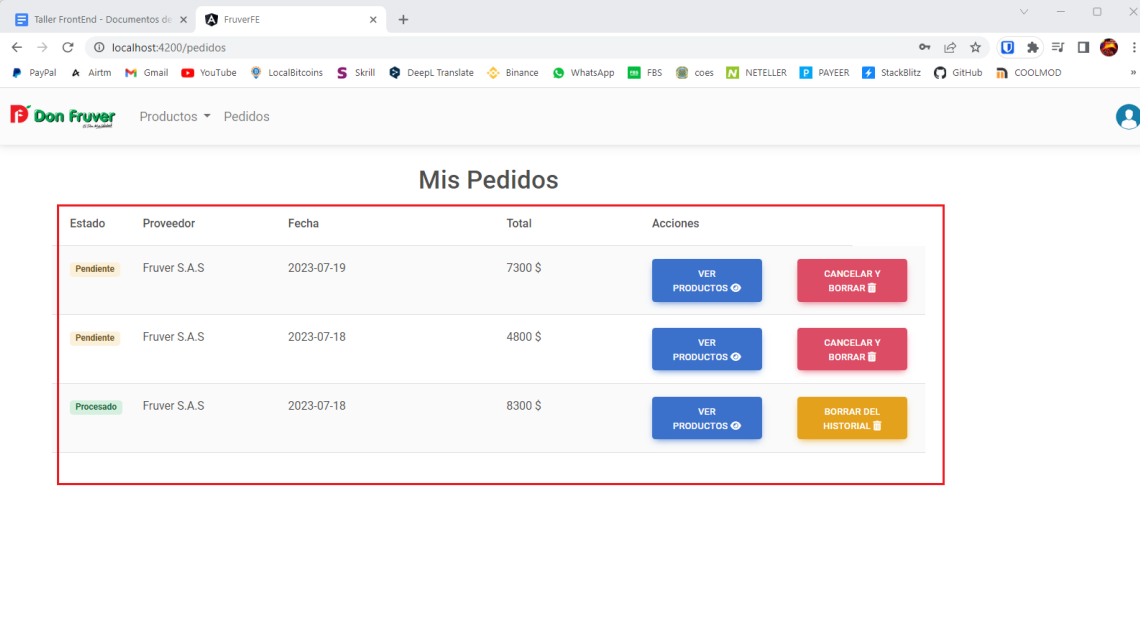
  if(!this.archivoSeleccionado) {alert("No se ha seleccionado ningun archivo"); return;}

  if (this.producto.idProducto) { //Viene de Editar
    fd.append('idProducto',this.idProducto);
    this.productoService.actualizarProducto(fd).subscribe({
      next: data => {this.router.navigate(['/productos']);},
      error: error => { validarAutorizacion(error) }});
  }
  else { //nuevo producto
    this.productoService.agregarProducto(fd).subscribe({
      next: data => {this.router.navigate(['/productos']);},
      error: error => { validarAutorizacion(error)}});
  }
}
```

Componente listar pedidos

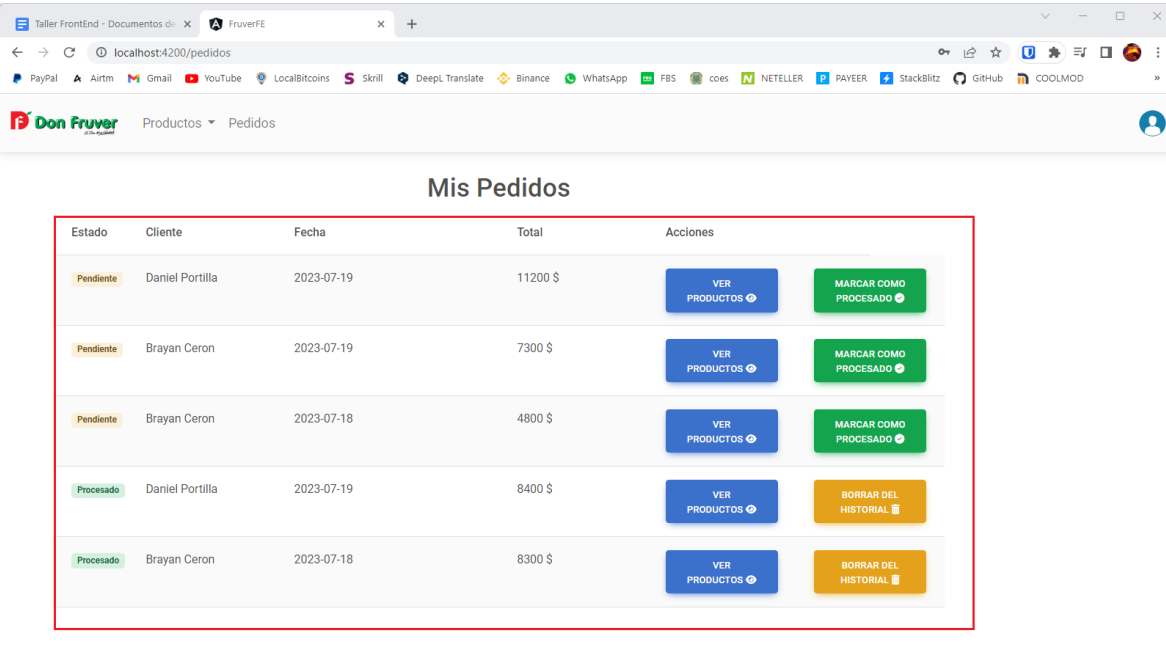
Este componente es el encargado de proporcionar la interfaz gráfica de usuario y lógica del lado FrontEnd, correspondiente a las funcionalidades para la visualización, edición, y eliminación de pedidos. Las tareas que se pueden realizar en este componente varían de acuerdo al contexto y al usuario que ha iniciado sesión, en ese sentido, se establece que el administrador es el único que puede procesar un pedido, y solo después de haberlo procesado puede borrar del historial dicho pedido. Por su parte el cliente además de ser el único que puede realizar un pedido, puede también cancelar o eliminar dicho pedido antes y después de que el administrador haya procesado el pedido

Interfaz gráfica listar pedidos para el rol usuario



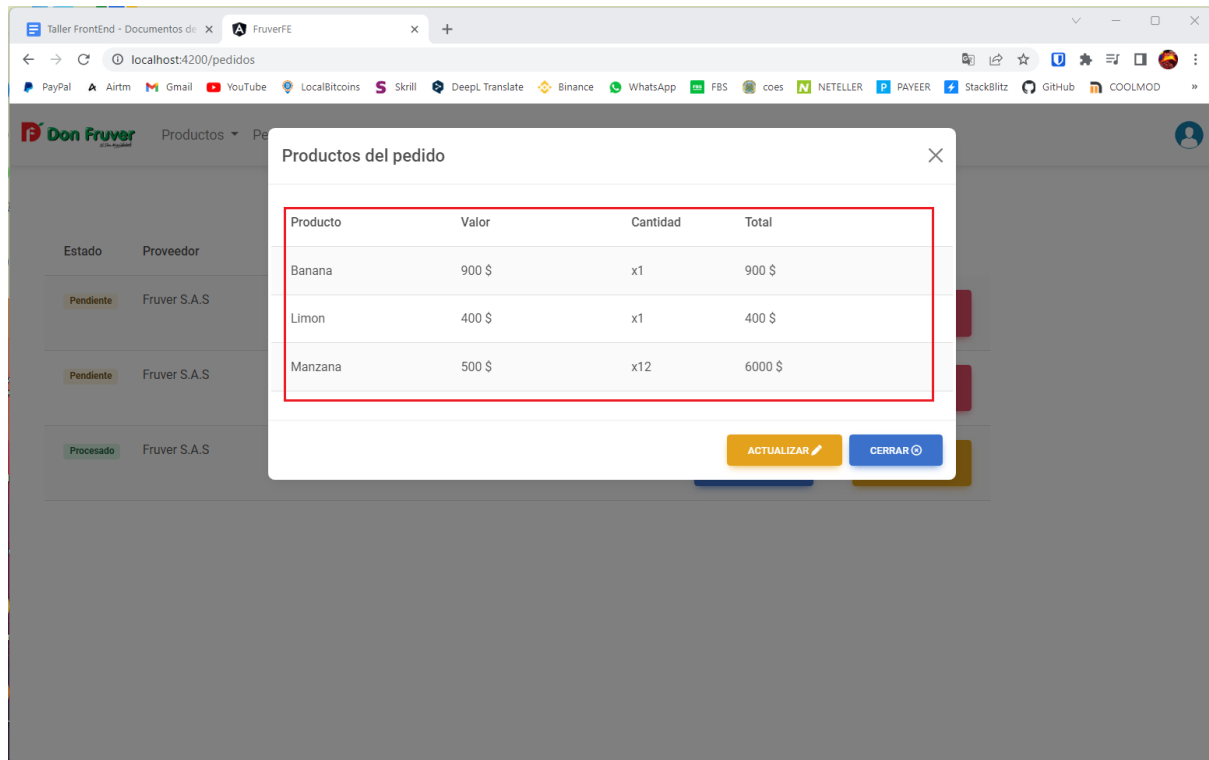
Estado	Proveedor	Fecha	Total	Acciones
Pendiente	Fruver S.A.S	2023-07-19	7300 \$	<button>VER PRODUCTOS</button> <button>CANCELAR Y BORRAR</button>
Pendiente	Fruver S.A.S	2023-07-18	4800 \$	<button>VER PRODUCTOS</button> <button>CANCELAR Y BORRAR</button>
Procesado	Fruver S.A.S	2023-07-18	8300 \$	<button>VER PRODUCTOS</button> <button>BORRAR DEL HISTORIAL</button>

Interfaz gráfica listar pedidos para el rol administrador



Estado	Cliente	Fecha	Total	Acciones
Pendiente	Daniel Portilla	2023-07-19	11200 \$	<button>VER PRODUCTOS</button> <button>MARCAR COMO PROCESADO</button>
Pendiente	Brayan Ceron	2023-07-19	7300 \$	<button>VER PRODUCTOS</button> <button>MARCAR COMO PROCESADO</button>
Pendiente	Brayan Ceron	2023-07-18	4800 \$	<button>VER PRODUCTOS</button> <button>MARCAR COMO PROCESADO</button>
Procesado	Daniel Portilla	2023-07-19	8400 \$	<button>VER PRODUCTOS</button> <button>BORRAR DEL HISTORIAL</button>
Procesado	Brayan Ceron	2023-07-18	8300 \$	<button>VER PRODUCTOS</button> <button>BORRAR DEL HISTORIAL</button>

Interfaz gráfica para ver productos de un pedido



Para inicializar el componente con información preprocesada implementamos la interfaz “OnInit”, en este caso usaremos esta interfaz para obtener los pedidos registrados, y para establecer sobre cada uno de estos las acciones que se puede hacer dependiendo el usuario que ha iniciado sesión. Para mostrar los pedidos utilizamos el servicio “PedidoService”, y mediante sus métodos “obtenerPedidos” y “obtenerPedidosCliente” podemos obtener todos los pedidos que han sido registrados o los pedidos por cliente. Para establecer las acciones que pueden realizarse según el rol del usuario, utilizamos el servicio “ClienteService”, el cual mediante su método “getRol” se conecta a la api del Backend para obtener el rol y el id de un usuario, para finalmente determinar si el usuario tiene autorización para acceder al recurso.

```
ngOnInit() {  
  this.clienteService.getRol().subscribe({  
    next: data => {  
      this.rol = data.rol;  
      this.idCliente = data.idCliente;  
  
      if (this.rol == "admin") {  
        this.pedidos = this.pedidoService.obtenerPedidos();  
        this.pedidos.subscribe({error: error => validarAutorizacion(error)});  
      }  
      else if (this.rol == "user") {  
        this.pedidos = this.pedidoService.obtenerPedidosCliente(this.idCliente);  
        this.pedidos.subscribe({error: error => validarAutorizacion(error)});  
      }  
    },  
    error: error => { validarRol(error); console.log(error) }  
  });  
}
```

La funcionalidad para procesar un pedido hace también uso del servicio “PedidoService”, el cual mediante su método “procesarPedidos” se conecta a la api del Backend para marcar como “procesado” un pedido. Esta acción también envía un correo electrónico de confirmación con todos los detalles de la compra al correo del cliente

```
setIdPedidoProcesar(idPedido: string){this.idPedidoProcesar=idPedido;}
procesarPedido() {
  if(this.idPedidoProcesar==""){ alert("Error, No se puede completar la operación");}
  this.pedidoService.procesarPedido(this.idPedidoProcesar).subscribe({
    next: data => {this.ngOnInit();/*console.Log("Registro Procesado");*/},
    error: error => {
      if(error.status==409){alert(error.error.mensaje);return;}
      validarAutorizacion(error);
    }});
}
```

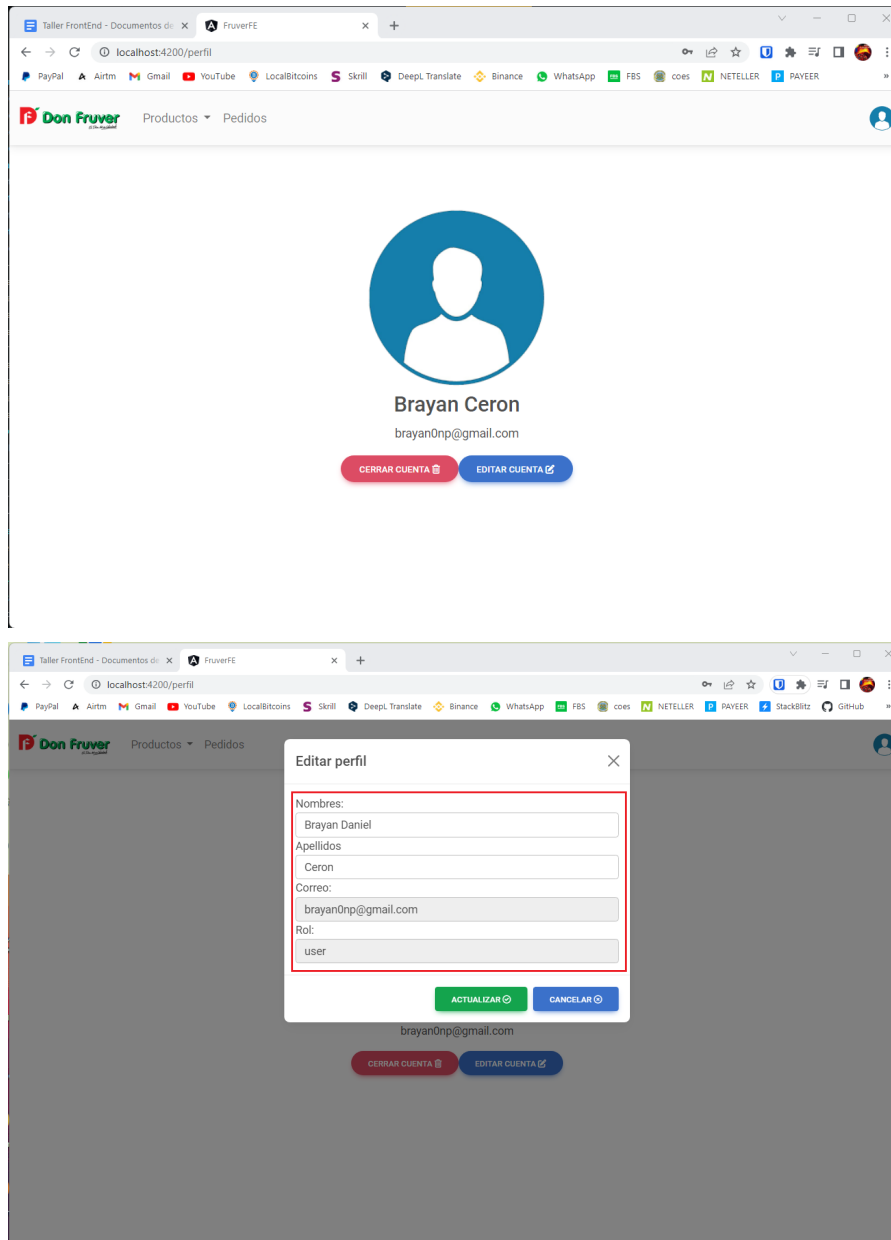
La funcionalidad para eliminar o cancelar un pedido hace de igual forma uso del servicio “PedidoService”, el cual mediante su método “borrarPedidos” se conecta a la api del Backend para borrar permanente la información de un pedido

```
setIdPedidoBorrar(idPedido: string){this.idPedidoBorrar=idPedido; }
borrarPedido() {
  if(this.idPedidoBorrar==""){ alert("Error, No se puede completar la operación");}
  this.pedidoService.borrarPedido(this.idPedidoBorrar).subscribe({
    next: data => {this.ngOnInit();/*console.Log("Registro Eliminado");*/},
    error: error => {
      validarAutorizacion(error);
    }});
}
```

La funcionalidad para editar un pedido realmente no se maneja en este componente, por lo que lo único que hace el botón “ACTUALIZAR” es redireccionar al componente “lista productos” el cual realiza toda la lógica para esa operación

Componente perfil

Este componente es el encargado de proporcionar la interfaz gráfica de usuario y lógica del lado FrontEnd, correspondiente a las funcionalidades para la visualización, edición y eliminación de la información de los usuarios.



Lógica del componente:

Para permitir la visualización, eliminación y modificación de la información de los usuarios este componente hace uso del servicio "ClienteService", el cual mediante sus métodos "obtenerCliente", "borrarCliente" y "actualizarCliente" se conecta a la api del Backend para obtener, borrar y actualizar la información de los usuarios

Para inicializar el componente con información preprocesada implementamos la interfaz “OnInit”, en este caso usaremos esta interfaz para obtener la información del usuario que ha iniciado sesión. En primer lugar validamos que el usuario tenga permisos para acceder a este recursos, esto lo hacemos mediante el servicio “ClienteService” y su método “getRol”. Finalmente para traer la información del usuario utilizamos este mismo servicio, pero en este caso utilizamos el método “obtenerCliente” para obtener toda la información personal de un usuario

```
ngOnInit(): void {
  this.clienteService.getRol().subscribe({
    next: async data => {
      this.rol = data.rol;
      this.idCliente = data.idCliente;

      this.cliente=await this.clienteService.obtenerCliente(this.idCliente);
      this.cliente.subscribe({
        next: data =>{
          this.nombres=data[0].nombres; this.edit_nombres= this.nombres;
          this.apellidos=data[0].apellidos; this.edit_apellidos=this.apellidos;
          this.correo=data[0].correo;
        },
        error: error => {validarAutorizacion(error); /*console.log(error);*/}
      });
      error: error => { validarRol(error); /*console.log(error);*/ }
    });
}
```

La funcionalidad para editar la información de un usuario hace uso del servicio “ClienteService”, el cual mediante su método “actualizarCliente” se conecta a la api del Backend para actualizar la información de un usuario. Cabe señalar, que para esta funcionalidad, solo se permite modificar el nombre y el apellido de un usuario, esto debido a que la demás información debe permanecer inalterable

```
actualizarClient(){
  if (this.nombres==" || this.apellidos == ""){alert("los campos no pueden estar vacíos");return;}
  let clienteModificado=new ClienteModel(this.idCliente,this.nombres,this.apellidos,"","");
  this.clienteService.actualizarCliente(clienteModificado).subscribe({
    next: data => {this.ngOnInit();},
    error: error => { validarAutorizacion(error); /*console.log(error);console.log(error.message);*/}
  });
}
```

La funcionalidad para eliminar un usuario (No disponible para el rol administrador) hace uso del servicio “ClienteService”, el cual mediante su método “borrarCliente” se conecta a la api del Backend para borrar un cliente mediante un id que se le debe pasar como parámetro

```
borrarCliente() {
  this.clienteService.borrarCliente(this.idCliente).subscribe({
    next: data => {
      localStorage.removeItem('token');
      this.router.navigate(['/login']);
    },
    error: error => {
      if(error.status==500){alert(error.error.mensaje);return;}
      else{validarAutorizacion(error);}
    }
  });
}
```

Autenticación

Para el sistema de autenticación y la protección de las rutas del lado Frontend (archivo app-routing.module.ts), se implementó un sencillo guard.

```
const routes: Routes = [
  { path: 'login', component: AutenticacionComponent, canActivate:[logiAgainnGuard]},
  { path: 'productos', component: ListaProductosComponent, canActivate:[loginGuard] },
  { path: 'productos/editar/:idProducto', component: EditarProductosComponent, canActivate:[loginGuard] },
  { path: 'productos/agregar', component: EditarProductosComponent, canActivate:[loginGuard] },
  { path: 'pedidos', component: ListarPedidosComponent, canActivate:[loginGuard] },
  { path: 'pedidos/editar/:idPedido', component: ListaProductosComponent, canActivate:[loginGuard] },
  { path: 'perfil', component: PerfilComponent, canActivate:[loginGuard] },
  { path: '**', redirectTo: '/productos', pathMatch: 'full' }
];

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

Este guard hace una validación muy básica, verifica en primera instancia que ya exista un token almacenado en el localStorage, si es así, se verifica la validez del token mediante el método “verificarToken” del servicio “ClienteService”; si cualquiera de las condiciones anteriores no se cumplen, se procede a redirigir a la ruta para hacer login. El guard válida también, que si ya existe un token se prohíba el acceso a la ruta para hacer login, esto debido a que no se debe permitir iniciar sesión si ya existe una sesión activa en el navegador

```
import { inject } from '@angular/core';
import { Router } from '@angular/router';
import { ClienteService } from '../shared/cliente.service';
import { lastValueFrom } from 'rxjs';
import { cierreDeSesion } from '../shared/utils';

const loginGuard = async () => {
  let clienteService = inject(ClienteService);
  const router = inject(Router);
  if (localStorage.getItem('token')) {
    try{
      let res = await lastValueFrom(clienteService.verificarToken());
      if (res.status=="200") {return true;}
      else {
        cierreDeSesion();
        return false;
      }
    }
    catch(error){
      cierreDeSesion();
      return false
    }
  }
  else {
    router.navigate(['/login']);
    return false;
  }
}

const logiAgainnGuard = () => {
  const router = inject(Router);
  if (localStorage.getItem('token')) {
    router.navigate(['/productos']);
    return false;
  }
  else {return true;}
}

export { loginGuard, logiAgainnGuard }
```

Finalmente, para evitar repetir demasiado código y hacer más legible el programa, se separaron las funciones más frecuentemente utilizadas en un nuevo archivo llamado `utils.ts`. Las funciones de “`validarAutorizacion`” y “`validarRol`” sirven para validar la autorización en la api del Backend, y rectifican principalmente la integridad y expiración del token. La función “`getHeaders`” se encarga simplemente de establecer el token que llevará la cabecera `authorization` en la petición `http`, este token posteriormente es utilizado en el Backend para determinar si dicha petición tiene acceso o no al recurso solicitado. Por último la función “`cierreDeSesion`” se encarga de destruir el token de autenticación y de mostrar un mensaje cuando la sesión ha terminado.

```
import { HttpResponse, HttpHeaders } from "@angular/common/http";

let SERVER_BASE_URL: string = 'http://localhost:3000';

const validarAutorizacion=(error: HttpResponse)=> {
  if (error.status == 401) {//Es porque se cerro la sesion
    cierreDeSesion();
    return;
  }
  alert("Error, No se pudo completar la operación, inténtelo mas tarde"); //Es un error desconocido
}

const validarRol=(error: HttpResponse) =>{
  if (error.status == 400 || error.status == 401 || error.status == 404) {
    cierreDeSesion();
    //return;
  }
}

const getHeaders= () =>{
  return {
    headers: new HttpHeaders({
      'Authorization': localStorage.getItem('token')!
    })
  };
}

const cierreDeSesion=()=>{
  alert("La sesión ha terminado, todos los cambios realizados no seran guardados. Por favor vuelva a hacer login nuevamente");
  localStorage.removeItem('token');
  window.location.reload();
}

export { SERVER_BASE_URL, validarAutorizacion, validarRol, getHeaders,cierreDeSesion }
```