

**Diplomado de actualización en nuevas tecnologías para el desarrollo de
software**

Taller Unidad 3 Frontend

EDISON CAMILO ROSERO ACHICANOY

UNIVERSIDAD DE NARIÑO

INGENIERIA DE SISTEMAS

2023

Para el desarrollo del proyecto frontend utilizando el framework Angular, es necesario crear componentes que separen las funciones según el tipo de usuario que ingrese a la aplicación web. A continuación, definimos los roles que se utilizarán:

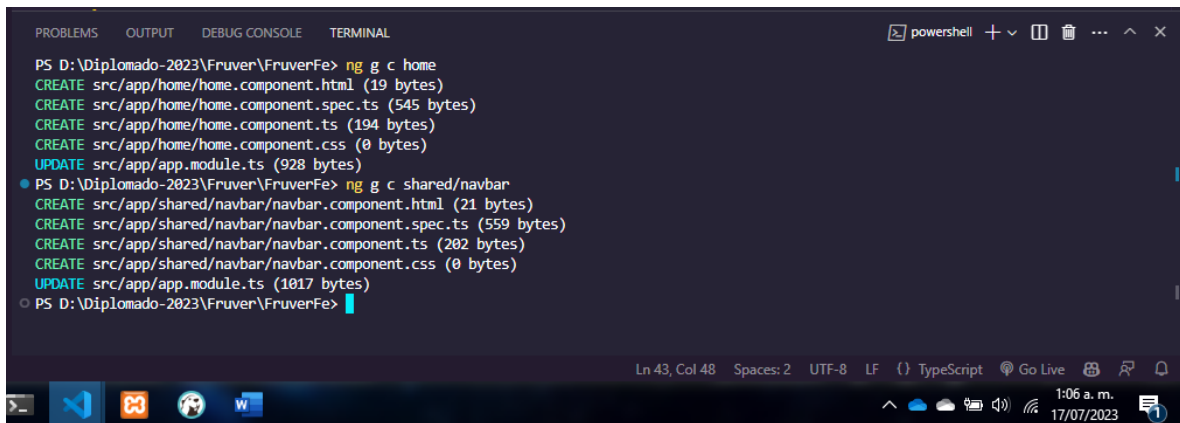
- **GUEST:** Este tipo de usuario corresponde a aquellos que no cuentan con una cuenta registrada en la aplicación.
- **USER:** Los usuarios registrados en la base de datos serán considerados como usuarios regulares.
- **ADMIN:** Este rol corresponde a los administradores de la aplicación.

Es importante destacar que cada uno de estos roles tendrá diferentes privilegios y acceso a funcionalidades específicas de la aplicación. Se usarán directivas estructurales como `*ngIf` para mostrar u ocultar elementos en función del rol del usuario.

Para la creación de componentes en angular se usa el comando “**ng generate component <nombre >**” o su contracción “**ng g c <nombre>**”

Se utilizarán los siguientes componentes en el proyecto:

- **Home:** Esta es la vista predeterminada que muestra una lista de productos disponibles.
- **DetalleProducto:** Esta vista muestra información detallada sobre un producto específico.
- **EditarProducto:** Esta vista permite la gestión de productos, como agregar, editar o eliminar productos de la base de datos.
- **ListarPedidos:** Esta vista se utiliza para la gestión de compras. Dependiendo del rol del usuario, se activarán funciones específicas para administrar los pedidos.
- **Navbar:** Este componente representa la barra de navegación superior en la aplicación. Contiene elementos como opciones de inicio de sesión y búsqueda.
- **Sidebar:** Este componente es una barra de navegación lateral que proporciona acceso rápido a los diferentes componentes y funcionalidades de la aplicación.



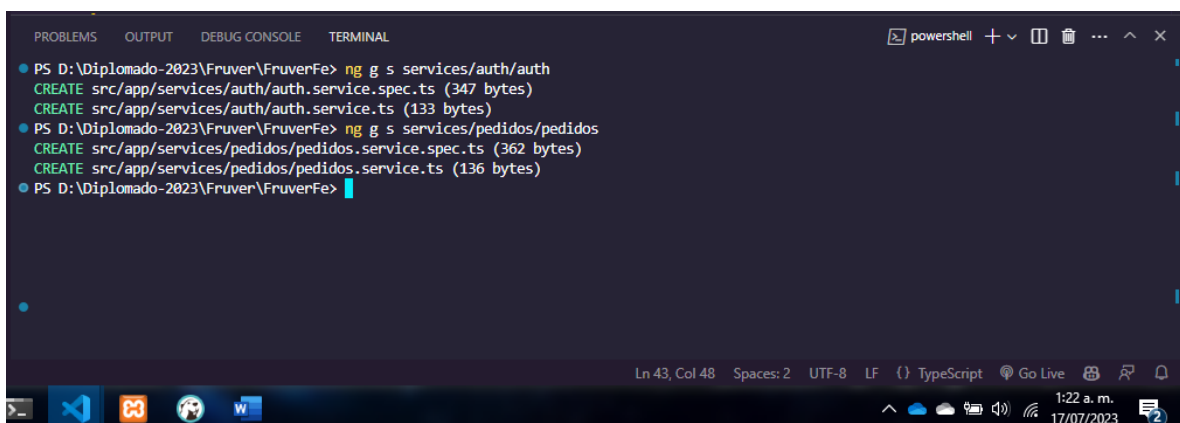
```
PS D:\Diplomado-2023\Fruver\FruverFe> ng g c home
CREATE src/app/home/home.component.html (19 bytes)
CREATE src/app/home/home.component.spec.ts (545 bytes)
CREATE src/app/home/home.component.ts (194 bytes)
CREATE src/app/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (928 bytes)
PS D:\Diplomado-2023\Fruver\FruverFe> ng g c shared/navbar
CREATE src/app/shared/navbar/navbar.component.html (21 bytes)
CREATE src/app/shared/navbar/navbar.component.spec.ts (559 bytes)
CREATE src/app/shared/navbar/navbar.component.ts (202 bytes)
CREATE src/app/shared/navbar/navbar.component.css (0 bytes)
UPDATE src/app/app.module.ts (1017 bytes)
PS D:\Diplomado-2023\Fruver\FruverFe>
```

También es necesario crear los servicios, cuya función es separar la lógica relacionada con los datos, la comunicación con el servidor y otras operaciones comunes, así evitar la duplicación de código y promover la reutilización.

Para la creación de servicios en angular se usa el comando “**ng generate service <nombre >**” o su contracción “**ng g s <nombre>**”

Se usarán los siguientes servicios que establecerán la comunicación con el servidor:

- Auth: Este servicio estará relacionado con la autenticación de usuarios.
- Productos: Se encargará de la gestión de productos, como obtener la lista de productos, crear nuevos productos, actualizar información de productos, etc.
- Pedidos: será responsable de la gestión de pedidos, como realizar pedidos, obtener el historial de pedidos, actualizar estados de pedidos, etc.
- ProductosPedidos: Este servicio se utilizará para gestionar los productos asociados a los pedidos, como modificar la cantidad, etc.



```
PS D:\Diplomado-2023\Fruver\FruverFe> ng g s services/auth/auth
CREATE src/app/services/auth/auth.service.spec.ts (347 bytes)
CREATE src/app/services/auth/auth.service.ts (133 bytes)
PS D:\Diplomado-2023\Fruver\FruverFe> ng g s services/pedidos/pedidos
CREATE src/app/services/pedidos/pedidos.service.spec.ts (362 bytes)
CREATE src/app/services/pedidos/pedidos.service.ts (136 bytes)
PS D:\Diplomado-2023\Fruver\FruverFe>
```

En Angular también es común crear modelos que representan las entidades de tu aplicación y se utilizan para mapear los datos recibidos desde el servidor o para estructurar los datos antes de enviarlos al servidor.

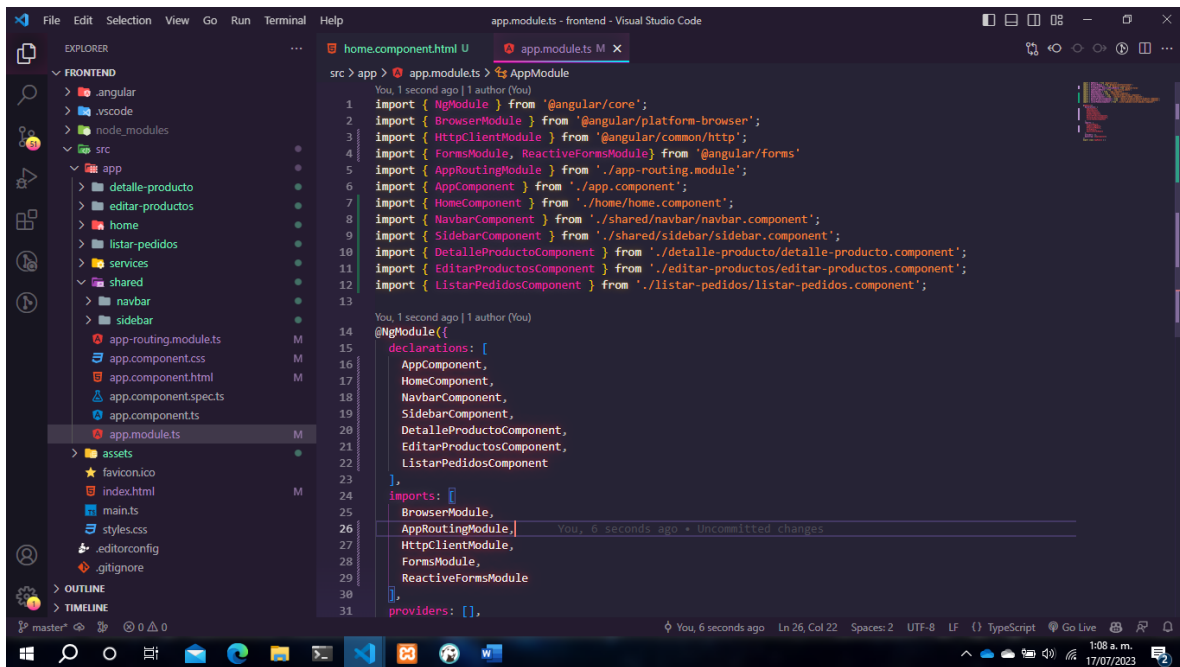
En esta ocasión, crearemos nuevos archivos TypeScript que definirán los modelos de las entidades en la base de datos del proyecto backend. Los archivos se ubicarán dentro de su servicio correspondiente para mantener un orden adecuado.

```
src > app > services > pedidoProducto > PedidosProd.model.ts > PedidoProductoModel
1  export class PedidoProductoModel {
2      constructor(
3          public idPP: number,
4          public idPedido: number,
5          public idProducto: number,
6          public cantidad: number
7      ) {}
8  }
9  }
```

```
src > app > services > pedidos > pedido.model.ts > PedidoModel
1  export class PedidoModel{
2      constructor(
3          public idPedido: string,
4          public idCliente: string,
5          public fechaSolicitud: string,
6          public estado: string,
7      ){}
8  }
9  }
```

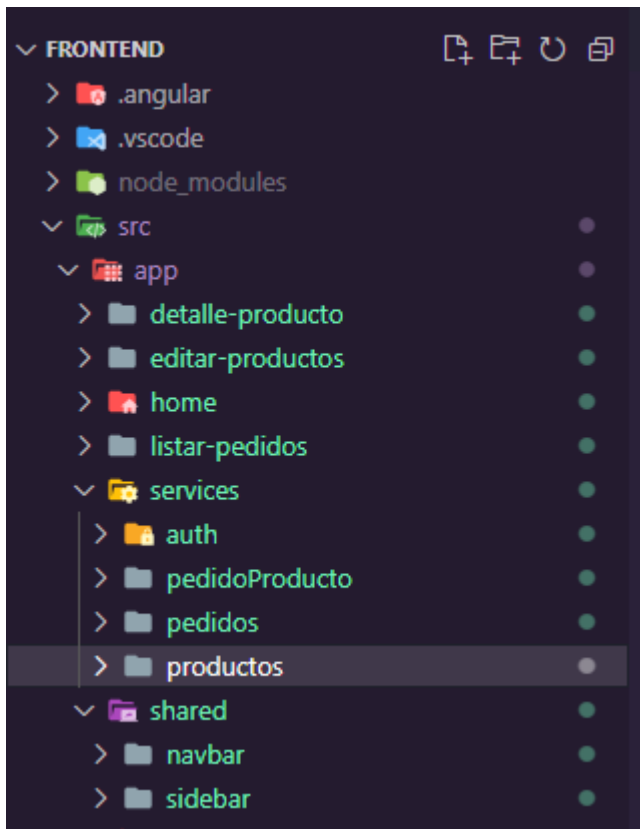
```
src > app > services > productos > producto.model.ts > ProductoModel
1 export class ProductoModel {
2     constructor(
3         public idProducto: string,
4         public nombre: string,
5         public descripcion: string,
6         public precio: number,
7         public imagen: string,
8     ) {}
9 }
```

Ya creados los componentes es necesario agregarlos en el archivo app.module.ts, de esta forma se asegura que estén disponibles en toda la aplicación y se puedan utilizar en otras partes de esta, como en las plantillas de otros componentes o en las rutas definidas en el enrutador.

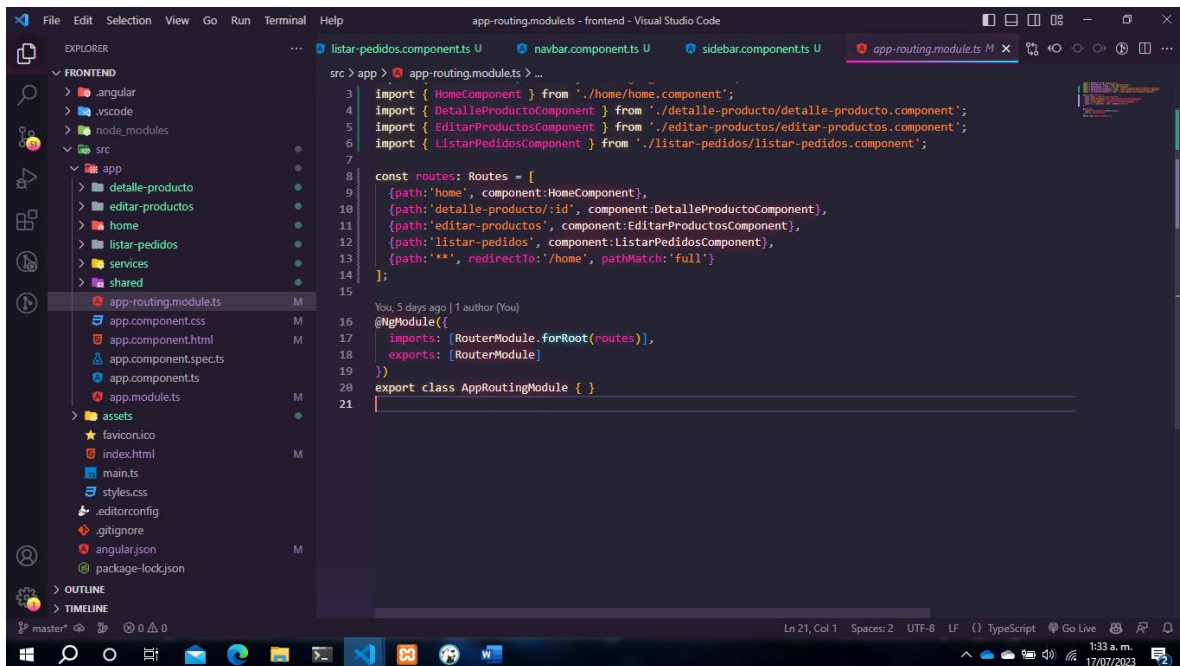


```
src > app > app.module.ts > AppModule
You, 1 second ago | 1 author (You)
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { HttpClientModule } from '@angular/common/http';
4 import { FormsModule, ReactiveFormsModule } from '@angular/forms';
5 import { AppRoutingModule } from './app-routing.module';
6 import { AppComponent } from './app.component';
7 import { HomeComponent } from './home/home.component';
8 import { NavbarComponent } from './shared/navbar/navbar.component';
9 import { SidebarComponent } from './shared/sidebar/sidebar.component';
10 import { DetalleProductoComponent } from './detalle-producto/detalle-producto.component';
11 import { EditarProductosComponent } from './editar-productos/editar-productos.component';
12 import { ListarPedidosComponent } from './listar-pedidos/listar-pedidos.component';
13
14 You, 1 second ago | 1 author (You)
15 @NgModule({
16     declarations: [
17         AppComponent,
18         HomeComponent,
19         NavbarComponent,
20         SidebarComponent,
21         DetalleProductoComponent,
22         EditarProductosComponent,
23         ListarPedidosComponent
24     ],
25     imports: [
26         BrowserModule,
27         AppRoutingModule,
28         HttpClientModule,
29         FormsModule,
30         ReactiveFormsModule
31     ],
32     providers: [],
33 })
```

Esta es la estructura del proyecto.



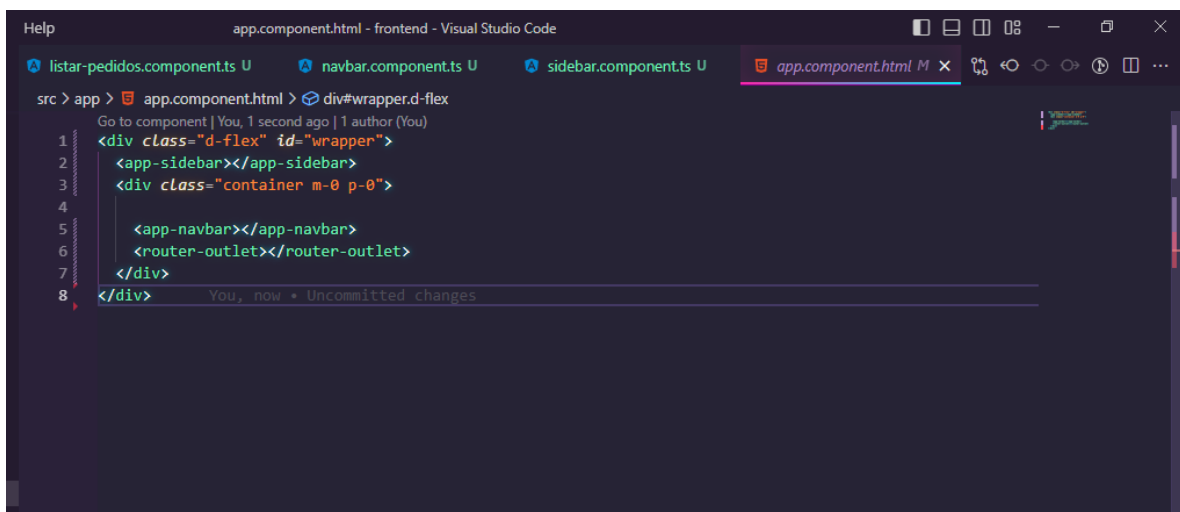
En el archivo `app-routing.module.ts` especificamos que cadena de texto está relacionada con un componente. Este archivo define y configura el enrutamiento de la aplicación Angular. Este archivo es responsable de determinar qué componentes deben mostrarse en función de las rutas o URLs específicas a las que acceda el usuario.



```
src > app > app-routing.module.ts > ...
3 import { HomeComponent } from './home/home.component';
4 import { DetalleProductoComponent } from './detalle-producto/detalle-producto.component';
5 import { EditarProductosComponent } from './editar-productos/editar-productos.component';
6 import { ListarPedidosComponent } from './listar-pedidos/listar-pedidos.component';
7
8 const routes: Routes = [
9   {path: 'home', component: HomeComponent},
10  {path: 'detalle-producto/:id', component: DetalleProductoComponent},
11  {path: 'editar-productos', component: EditarProductosComponent},
12  {path: 'listar-pedidos', component: ListarPedidosComponent},
13  {path: '**', redirectTo: '/home', pathMatch: 'full'}
14 ];
15
16 @NgModule({
17   imports: [RouterModule.forRoot(routes)],
18   exports: [RouterModule]
19 })
20 export class AppRoutingModule { }
21
```

En el archivo `app.component.html`, elimina todo el contenido existente y agrega la etiqueta `<router-outlet>`. Esta etiqueta se encargará de renderizar los componentes correspondientes a las rutas definidas en el archivo de enrutamiento. También puedes utilizar esta plantilla para incluir componentes que siempre deben ser visibles en todas las rutas de la aplicación.

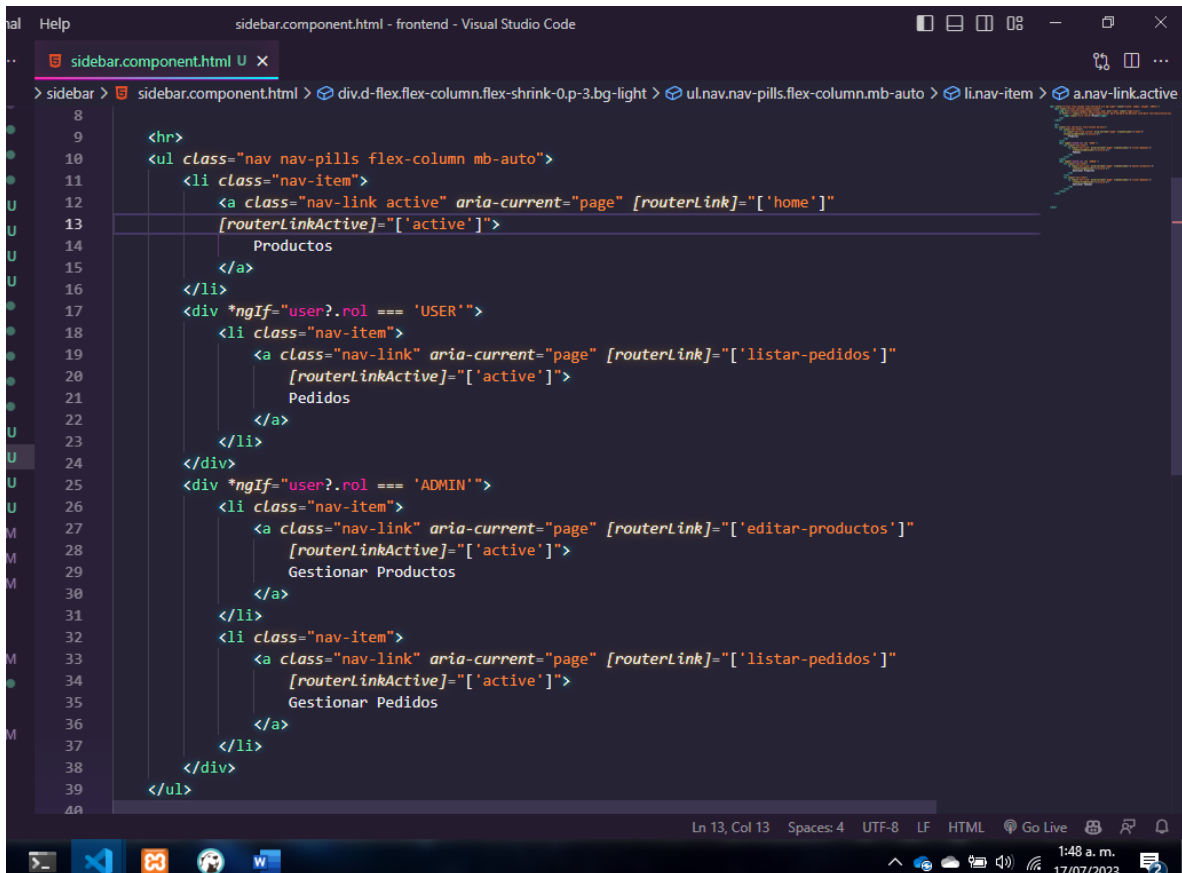
En este caso, el componente `navbar` y `sidebar` siempre serán visibles en todos los componentes.



```
src > app > app.component.html > div#wrapper.d-flex
Go to component | You, 1 second ago | 1 author (You)
1 <div class="d-flex" id="wrapper">
2   <app-sidebar></app-sidebar>
3   <div class="container m-0 p-0">
4     <app-navbar></app-navbar>
5     <router-outlet></router-outlet>
6   </div>
7 </div>
8 </div> You, now • Uncommitted changes
```

Ya definidas las rutas podemos hacer uso de la directiva [routerLink] para establecer enlaces de navegación entre diferentes rutas en la aplicación.

Usamos la directiva ngIf para habilitar funciones dependiendo del tipo de usuario.



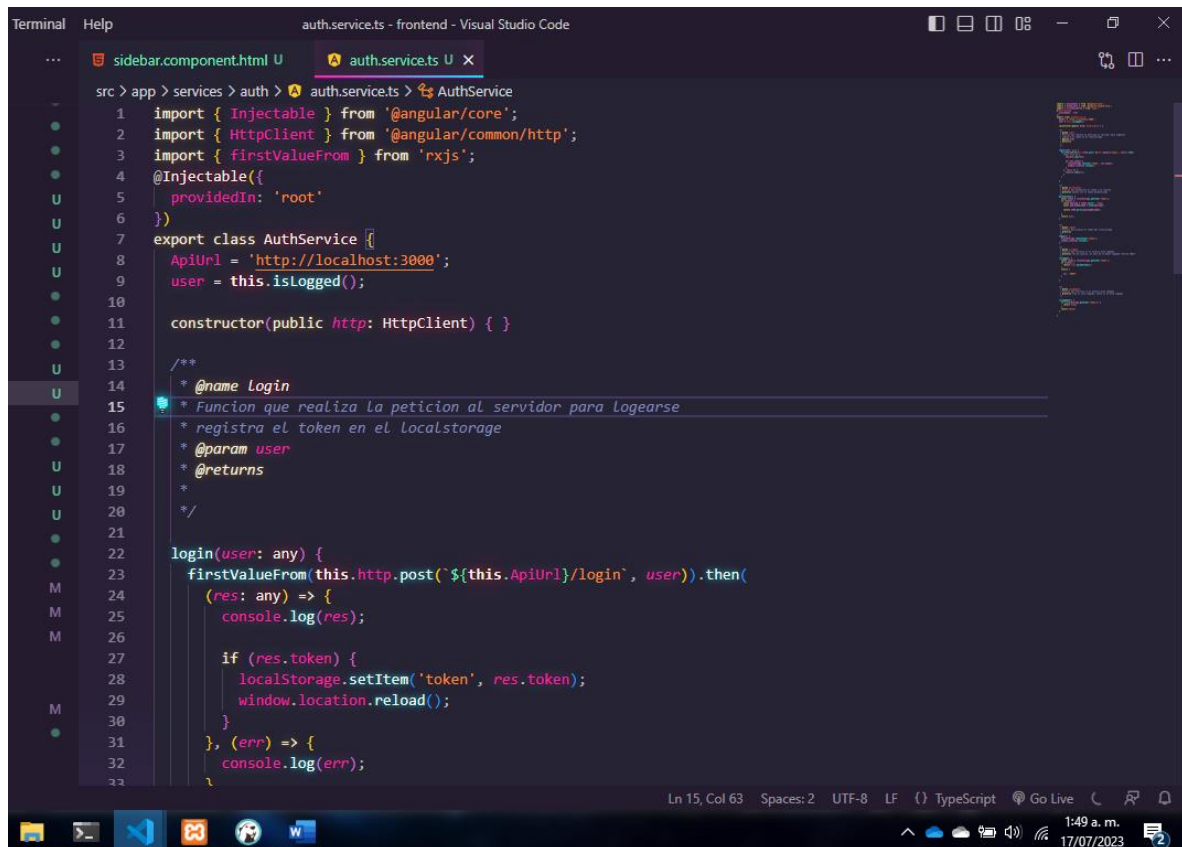
```
8
9
10 <hr>
11 <ul class="nav nav-pills flex-column mb-auto">
12   <li class="nav-item">
13     <a class="nav-link active" aria-current="page" [routerLink]="['home']"
14       [routerLinkActive]="['active']">
15       Productos
16     </a>
17   </li>
18   <div *ngIf="user?.rol === 'USER'">
19     <li class="nav-item">
20       <a class="nav-link" aria-current="page" [routerLink]="['listar-pedidos']"
21         [routerLinkActive]="['active']">
22         Pedidos
23       </a>
24     </li>
25   </div>
26   <div *ngIf="user?.rol === 'ADMIN'">
27     <li class="nav-item">
28       <a class="nav-link" aria-current="page" [routerLink]="['editar-productos']"
29         [routerLinkActive]="['active']">
30         Gestionar Productos
31       </a>
32     </li>
33     <li class="nav-item">
34       <a class="nav-link" aria-current="page" [routerLink]="['listar-pedidos']"
35         [routerLinkActive]="['active']">
36         Gestionar Pedidos
37       </a>
38     </li>
39   </div>
40 </ul>
```


Modificaremos el proyecto anterior (backend) para implementar las funciones de autenticación.

```
controllers > clientes.controller.js > login
You, 1 second ago | 1 author (You)
1 import Models from '../models/index.js';
2 import jwt from 'jsonwebtoken';
3
4 const login = async (req, res) => {
5   try {
6     let { email, password } = req.body;
7     let cliente = await Models.Cliente.findOne({
8       attributes: ['idCliente', 'identificacion', 'nombre', 'apellido', 'email', 'rol'],
9       where: {
10         email: email,
11         contrasena: password
12       }
13     });
14
15     if (cliente) {
16       console.log(cliente);
17       let data = JSON.stringify(cliente);
18       let token = jwt.sign(data, 'secretKey');
19       res.status(200).send({ token });
20     } else {
21       res.status(401).json({ message: 'Credenciales incorrectas' });
22     }
23   } catch (error) {
24     console.log(error);
25     res.status(500).json({ message: 'Error en el servidor' });
26   }
27 }
28
29
```

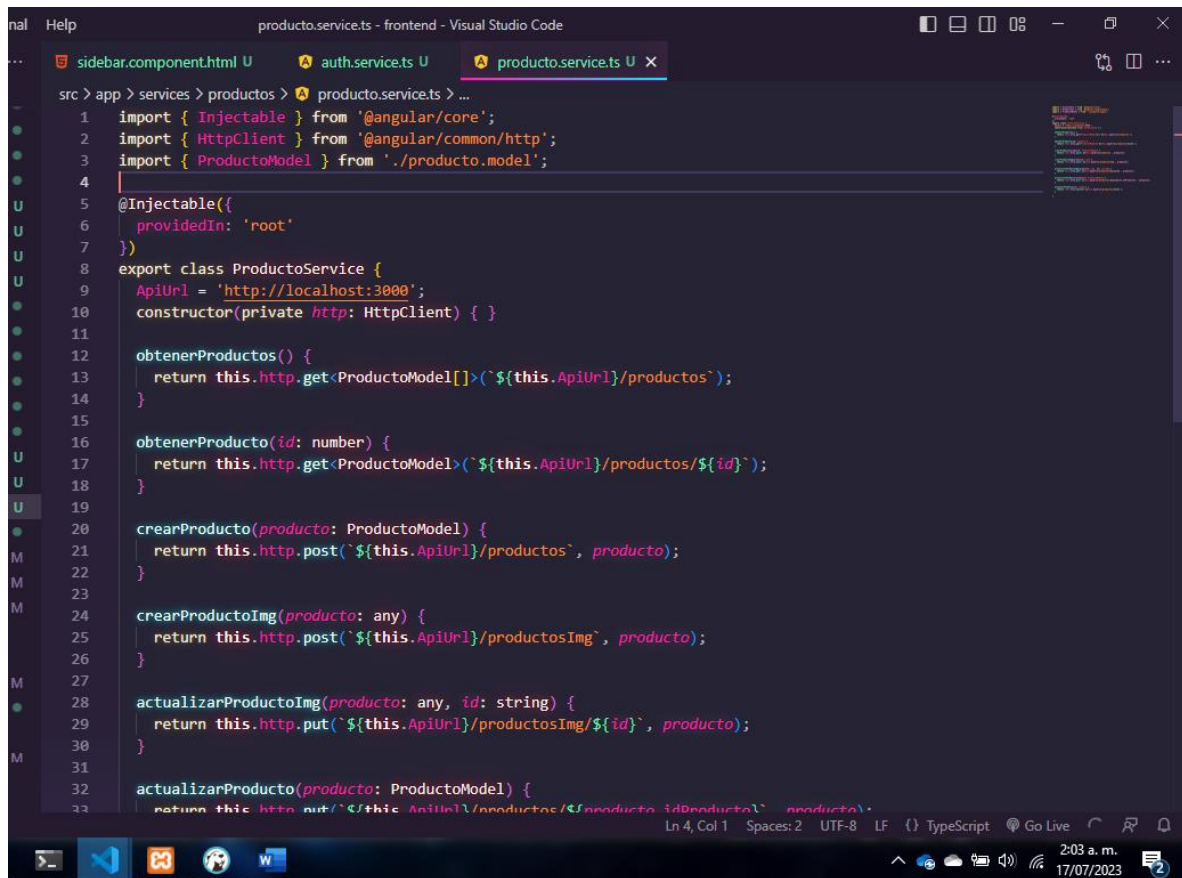
En los servicios de Frontend se establecerá la conexión con el servidor.

En el servicio de autenticación (Auth), existirá el método login que obtendrá del servidor un token, el cual será almacenado en el Local Storage. Además, el servicio puede incluir otras funciones como cerrar sesión, decodificar el token, entre otras.



```
src > app > services > auth > auth.service.ts > AuthService
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { firstValueFrom } from 'rxjs';
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class AuthService {
8   ApiUrl = 'http://localhost:3000';
9   user = this.isLoggedIn();
10
11   constructor(public http: HttpClient) { }
12
13   /**
14    * @name Login
15    * Funcion que realiza la peticion al servidor para logearse
16    * registra el token en el localStorage
17    * @param user
18    * @returns
19    */
20   login(user: any) {
21     firstValueFrom(this.http.post(`${this.ApiUrl}/login`, user)).then(
22       (res: any) => {
23         console.log(res);
24
25         if (res.token) {
26           localStorage.setItem('token', res.token);
27           window.location.reload();
28         }
29       }, (err) => {
30         console.log(err);
31       }
32     );
33   }
34 }
```

En el servicio de productos, se usa el modelo de productos creado anteriormente y se implementarán los métodos para gestionar los datos relacionados con los productos en el servidor. Estos métodos permitirán realizar operaciones como obtener la lista de productos, crear un nuevo producto, actualizar información de un producto existente y eliminar un producto.



```
src > app > services > productos > producto.service.ts > ...
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { ProductoModel } from '../producto.model';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class ProductoService {
9   apiUrl = 'http://localhost:3000';
10   constructor(private http: HttpClient) { }
11
12   obtenerProductos() {
13     return this.http.get<ProductoModel[]>(`${this.apiUrl}/productos`);
14   }
15
16   obtenerProducto(id: number) {
17     return this.http.get<ProductoModel>(`${this.apiUrl}/productos/${id}`);
18   }
19
20   crearProducto(producto: ProductoModel) {
21     return this.http.post(`${this.apiUrl}/productos`, producto);
22   }
23
24   crearProductoImg(producto: any) {
25     return this.http.post(`${this.apiUrl}/productosImg`, producto);
26   }
27
28   actualizarProductoImg(producto: any, id: string) {
29     return this.http.put(`${this.apiUrl}/productosImg/${id}`, producto);
30   }
31
32   actualizarProducto(producto: ProductoModel) {
33     return this.http.put(`${this.apiUrl}/productos/${producto.idProducto}`, producto);
34   }
35 }
```

Ln 4, Col 1 Spaces: 2 UTF-8 LF {} TypeScript Go Live 2:03 a.m. 17/07/2023

En el servicio de pedidos, no se utiliza un modelo específico debido a que las consultas realizadas al servidor API involucran relaciones complejas entre diferentes entidades en la base de datos. Estas consultas pueden incluir instrucciones para obtener información relacionada.

The screenshot shows the Visual Studio Code editor with the file `pedido.service.ts` open. The code defines an Angular service `PedidoService` that interacts with a REST API. It includes imports for `Injectable` and `HttpClient` from Angular. The service is decorated with `@Injectable({ providedIn: 'root' })`. The `ApiUrl` is set to `'http://localhost:3000'`. The service implements several methods: `obtenerPedidos` (GET /pedidos), `obtenerPedido(id: number)` (GET /pedidos/{id}), `obtenerPedidosCliente(id: number)` (GET /pedidosCliente/{id}), `crearPedido(pedido: any)` (POST /pedidos), `actualizarPedido(pedido: any)` (PUT /pedidos/{pedido.idPedido}), and `eliminarPedido(id: number)` (DELETE /pedidos/{id}).

```
src > app > services > pedidos > pedido.service.ts > PedidoService > obtenerPedidos
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class PedidoService {
8   ApiUrl = 'http://localhost:3000';
9   constructor(private http: HttpClient) { }
10
11   obtenerPedidos() {
12     return this.http.get(`${this.ApiUrl}/pedidos`);
13   }
14
15   obtenerPedido(id: number) {
16     return this.http.get(`${this.ApiUrl}/pedidos/${id}`);
17   }
18
19   obtenerPedidosCliente(id: number) {
20     return this.http.get(`${this.ApiUrl}/pedidosCliente/${id}`);
21   }
22
23   crearPedido(pedido: any) {
24     return this.http.post(`${this.ApiUrl}/pedidos`, pedido);
25   }
26
27   actualizarPedido(pedido: any) {
28     return this.http.put(`${this.ApiUrl}/pedidos/${pedido.idPedido}`, pedido);
29   }
30
31   eliminarPedido(id: number) {
32     return this.http.delete(`${this.ApiUrl}/pedidos/${id}`);
33   }
34 }
```

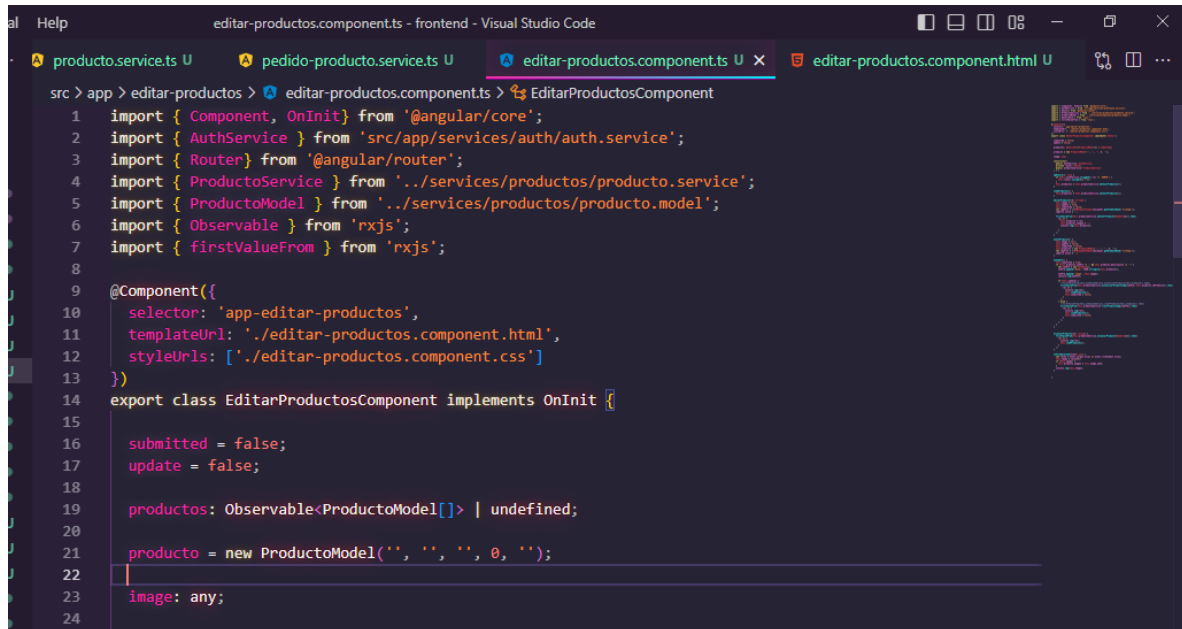
Servicio pedido-producto.

The screenshot shows the Visual Studio Code editor with the file `pedido-producto.service.ts` open. The code defines an Angular service `PedidoProductoService` that interacts with a REST API. It includes imports for `Injectable`, `PedidoProductoModel`, and `HttpClient`. The service is decorated with `@Injectable({ providedIn: 'root' })`. The `ApiUrl` is set to `'http://localhost:3000'`. The service implements several methods: `obtenerPedidosProducto` (GET /pedidosProd), `obtenerPedidoProducto(id: number)` (GET /pedidosProd/{id}), `obtenerPedidosProductoPedido(id: number)` (GET /pedidosProd/{id}), `crearPedidoProducto(pedidoProducto: PedidoProductoModel)` (POST /pedidosProd), and `actualizarPedidoProducto(pedidoProducto: PedidoProductoModel)` (PUT /pedidosProd/{pedidoProducto.idPP}).

```
src > app > services > pedidoProducto > pedido-producto.service.ts > PedidoProductoService
1 import { Injectable } from '@angular/core';
2 import { PedidoProductoModel } from './PedidosProd.model';
3 import { HttpClient } from '@angular/common/http';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class PedidoProductoService {
9   ApiUrl = 'http://localhost:3000';
10   constructor(private http: HttpClient) { }
11
12   obtenerPedidosProducto() {
13     return this.http.get<PedidoProductoModel[]>(`${this.ApiUrl}/pedidosProd`);
14   }
15
16   obtenerPedidoProducto(id: number) {
17     return this.http.get<PedidoProductoModel>(`${this.ApiUrl}/pedidosProd/${id}`);
18   }
19
20   obtenerPedidosProductoPedido(id: number) {
21     return this.http.get<PedidoProductoModel[]>(`${this.ApiUrl}/pedidosProd/${id}`);
22   }
23
24   crearPedidoProducto(pedidoProducto: PedidoProductoModel) {
25     return this.http.post<PedidoProductoModel>(`${this.ApiUrl}/pedidosProd`, pedidoProducto);
26   }
27
28   actualizarPedidoProducto(pedidoProducto: PedidoProductoModel) {
29     return this.http.put<PedidoProductoModel>(`${this.ApiUrl}/pedidosProd/${pedidoProducto.idPP}`, pedidoProducto);
30   }
31
32 }
33 }
```

Uso de NgModel

En el archivo typescript del componente editar-producto se crea un objeto de tipo producto.



```
src > app > editar-productos > editar-productos.component.ts > EditarProductosComponent
1 import { Component, OnInit } from '@angular/core';
2 import { AuthService } from 'src/app/services/auth/auth.service';
3 import { Router } from '@angular/router';
4 import { ProductoService } from '../services/productos/producto.service';
5 import { ProductoModel } from '../services/productos/producto.model';
6 import { Observable } from 'rxjs';
7 import { firstValueFrom } from 'rxjs';
8
9 @Component({
10   selector: 'app-editar-productos',
11   templateUrl: './editar-productos.component.html',
12   styleUrls: ['./editar-productos.component.css']
13 })
14 export class EditarProductosComponent implements OnInit {
15
16   submitted = false;
17   update = false;
18
19   productos: Observable<ProductoModel[]> | undefined;
20
21   producto = new ProductoModel('', '', '', 0, '');
22
23   image: any;
24 }
```

En la plantilla HTML del componente, se usa la propiedad producto y ngModel para vincular los datos del formulario con el modelo.

Se agregan condiciones adicionales que deshabilitan el botón de guardar para no permitir el ingreso de datos nulos en los campos de nombre, descripción y cantidad. De esta forma, el formulario solo se enviará si estos campos están completos.


```

48 <form (ngSubmit)="onSubmit()" class="formProd" enctype="multipart/form-data" #form="ngForm">
49   <div class="col-auto">
50     <label for="nPrd" class="form-label">Nombre</label>
51     <input type="text" class="form-control" id="nPrd" [(ngModel)]="producto.nombre" name="nombre" #
52   </div>
53   <div class="col-auto">
54     <label for="about" class="form-label">Descripción</label>
55     <textarea type="text" class="form-control" id="about" [(ngModel)]="producto.descripcion"
56       name="descripcion" #descripcion="ngModel" required>
57   </div>
58   <div class="col-auto">
59     <label for="precio" class="form-label">Precio</label>
60     <input type="number" class="form-control" id="precio" [(ngModel)]="producto.precio"
61       name="precio" #precio="ngModel" value="0" required>
62   </div>
63   <div class="col-auto">
64     <div class="col-auto">
65       <label for="img" class="form-label">Imagen</label>
66       <input type="hidden" [(ngModel)]="producto.imagen" name="imagen" #imagen="ngModel">
67     <input type="file" class="form-control" #fileImg
68       (change)="onFileSelected($event)" id="fileImg" name="fileImg" accept="image/*">
69   </div>
70   <div class="col-auto my-3 d-flex justify-content-end gap-3">
71     <button type="submit" class="btn btn-outline-success mb-3" [class.disabled]="form.invalid">Guar
72     <button type="button" class="btn btn-outline-danger mb-3" data-bs-dismiss="modal"
73       aria-label="Close">Cancelar</button>
74   </div>
75 </form>
76 </div>
77 </div>
78 </div>
79

```

Se crearon nuevos métodos en el servidor para el almacenamiento de imágenes, usando la librería Multer que se utiliza para el manejo de formularios con multipart/form-data, para la carga de archivos en una aplicación web.

Archivo route.js

```

7 You, 3 minutes ago • Uncommitted changes
8 import multer from 'multer';
9
18 // productos
19 router.get('/productos', productosController.getProductos);
20 router.get('/productos/:id', productosController.getProductos);
21 router.post('/productos', productosController.postProducto);
22 router.post('/productosImg', multer().single('image'), productosController.postProductoImg);
23 router.put('/productosImg/:id', multer().single('image'), productosController.putProductoImg);
24 router.put('/productos/:id', productosController.putProducto);
25 router.delete('/productos/:id', productosController.deleteProducto);
26
27 // pedidos

```

Productos.controller.js

```
const postProductoImg = async (req, res) => {
  try {
    let t = await sequelize.transaction();
    let body = JSON.parse(req.body.data);
    let img = req.file;

    let producto = await Models.Producto.create(body, { transaction: t });
    let update = await Models.Producto.findByPk(producto.idProducto, { transaction: t });

    // set name image with id of producto
    let nameImg = `${producto.idProducto}.jpg`;
    update.imagen = nameImg;
    if (img) {
      // save image
      fs.writeFileSync(`./public/media/${nameImg}`, img.buffer);
    }

    await update.save({ transaction: t });
    await t.commit();

    res.status(200).json(producto);
  } catch (error) {
    await t.rollback();
    res.status(500).json(error);
    console.log(error);
  }
}
```

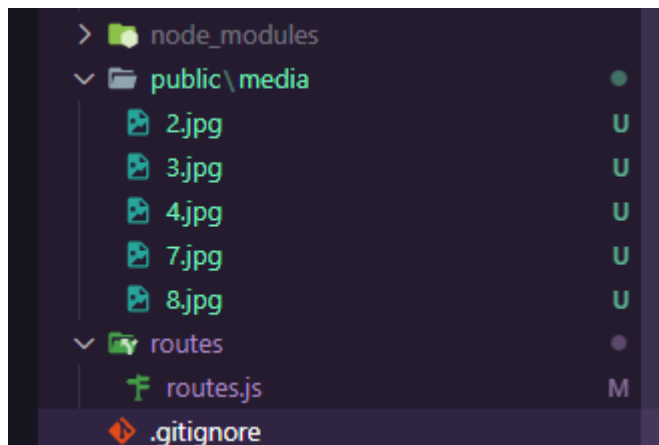
```
73 const putProductoImg = async (req, res) => {
74   try {
75     console.log(req.body, req.file);
76     let body = JSON.parse(req.body.data);
77     let img = req.file;
78     console.log(body, img);
79     let t = await sequelize.transaction({ autocommit: false });
80     let idProducto = req.params.id;
81     let producto = await Models.Producto.findByPk(idProducto, { transaction: t });
82     producto = await producto.update(body, { transaction: t });
83
84     // set name image with id of producto
85     let nameImg = `${producto.idProducto}.jpg`;
86     producto.imagen = nameImg;
87     if (img) {
88       // save image
89       fs.writeFileSync(`./public/media/${nameImg}`, img.buffer);
90     }
91
92     await producto.save({ transaction: t });
93     await t.commit();
94
95     res.status(200).json(producto);
96
97   } catch (error) {
98     await t.rollback();
99     res.status(500).json(error);
100    console.log(error);
101  }
102 }
```

App.js

```
● You, 4 minutes ago | 1 author (You)
M 1 import express from 'express';
M 2 import routes from './routes/routes.js';
M 3 import sequelize from './Database/database.js';
M 4 import cors from 'cors';
M 5 import path from 'path';
6 import { fileURLToPath } from 'url';
7
8
```

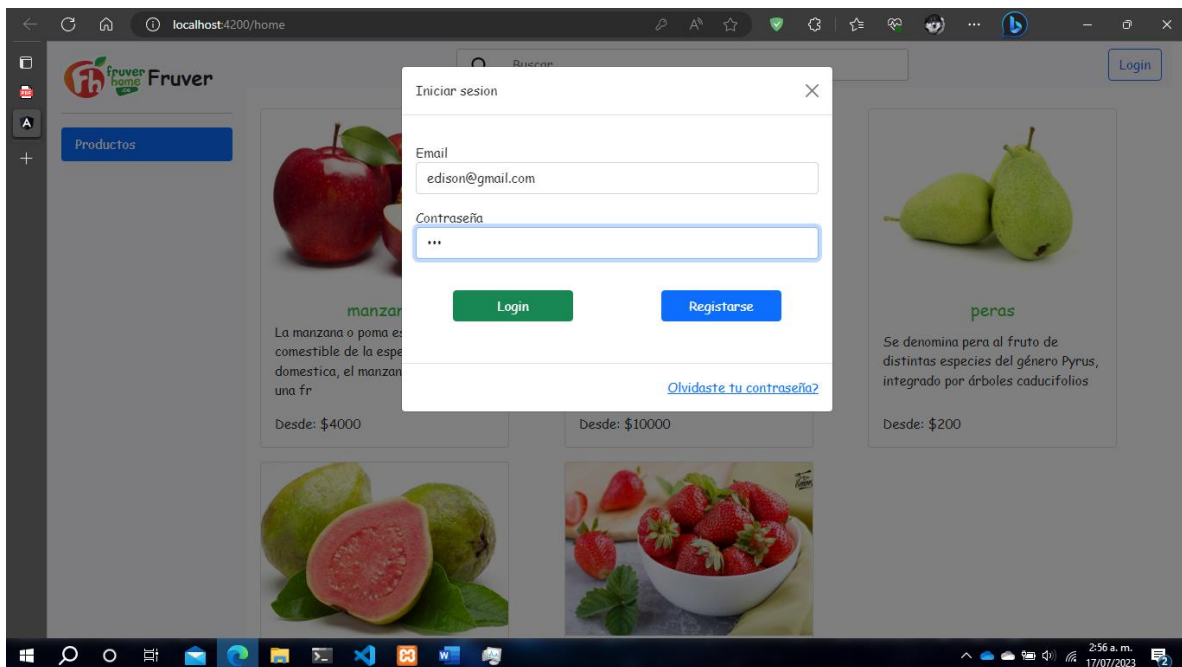
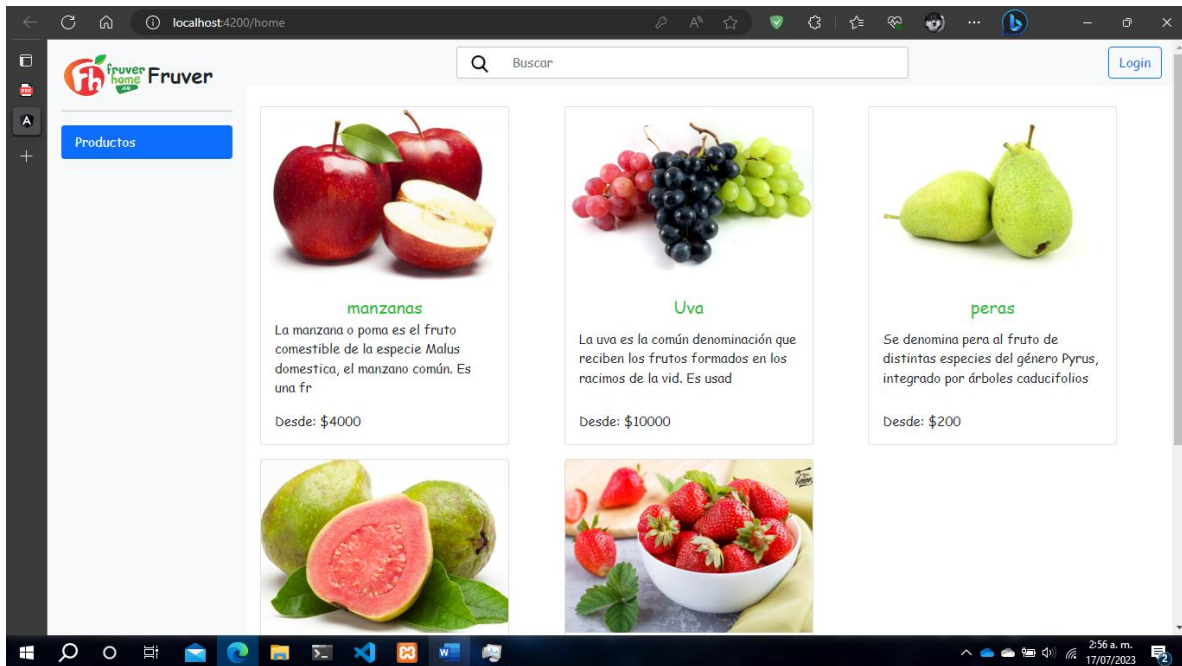
```
10
11
12 const app = express();
13 const port = 3000;
14 const __filename = fileURLToPath(import.meta.url);
15 const __dirname = path.dirname(__filename);
16
17 app.use(cors());
18 app.set('port', port);
19 app.use(express.json());
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 You, 5 minutes ago • Uncommitted changes
23
```

Las imágenes se guardarán en el servidor en la carpeta “public/media” con el nombre del id del producto al que pertenecen para evitar conflicto.



Interfaz de usuario.

Se hace uso de la biblioteca Bootstrap para los estilos, además de unos estilos propios.




fb fruver home

Productos

Pedidos

Buscar

Edison Camilo




manzanas

La manzana o poma es el fruto comestible de la especie Malus domestica, el manzano común. Es una fr

Desde: \$4000

1 Añadir al carrito




Uva

La uva es la común denominación que reciben los frutos formados en los racimos de la vid. Es usad

Desde: \$10000

1 Añadir al carrito





peras

Se denomina pera al fruto de distintas especies del género Pyrus, integrado por árboles caducifolios

Desde: \$200

1 Añadir al carrito





Explorador de archivos

fb fruver home



Productos

Pedidos

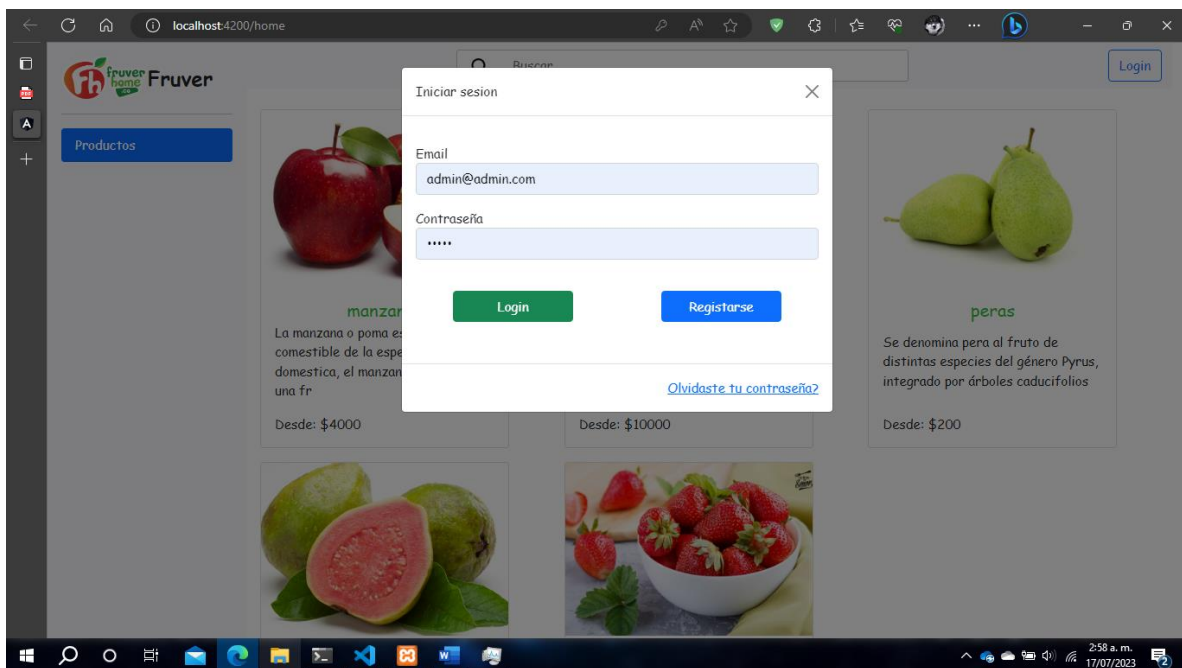
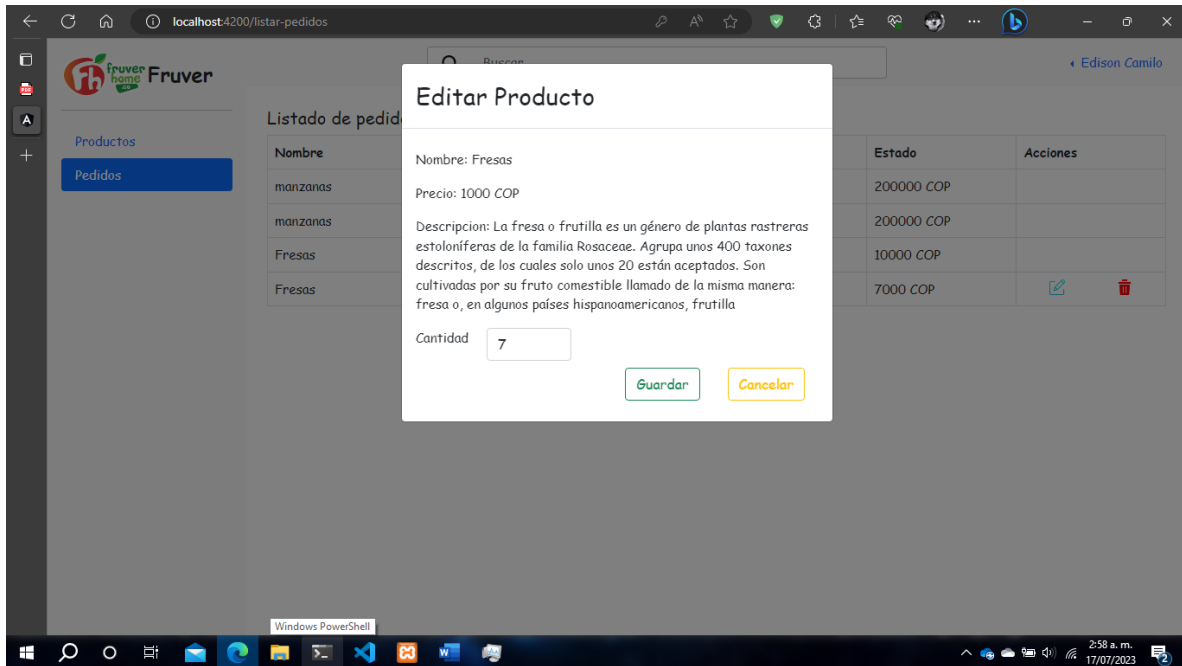
Buscar

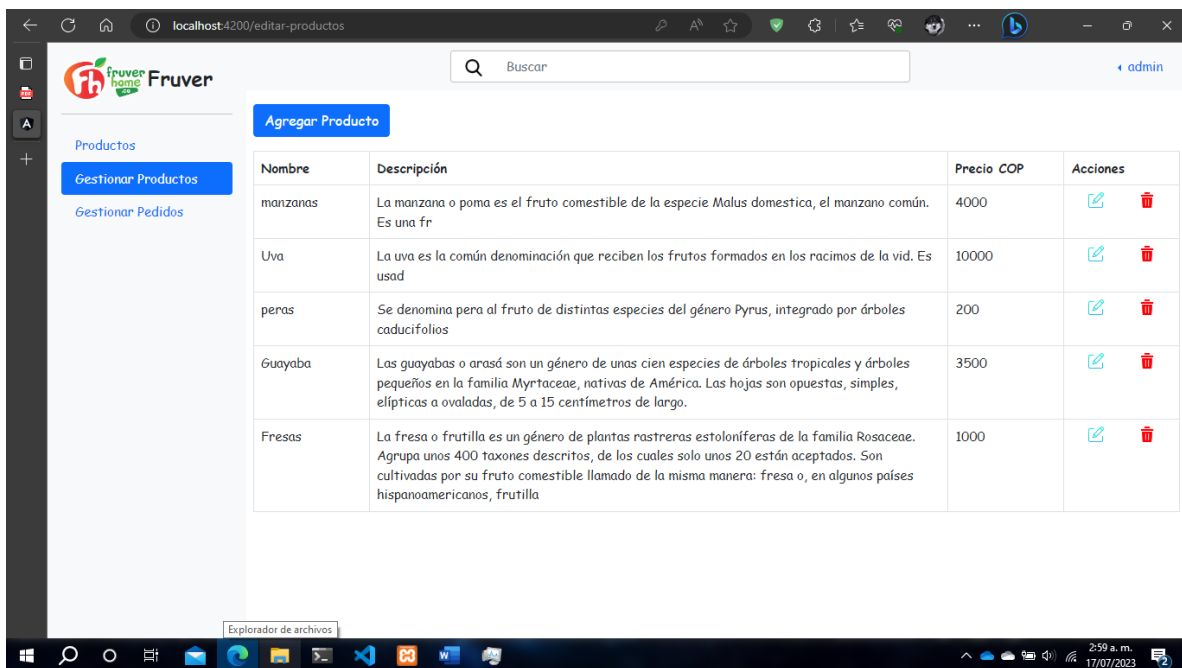
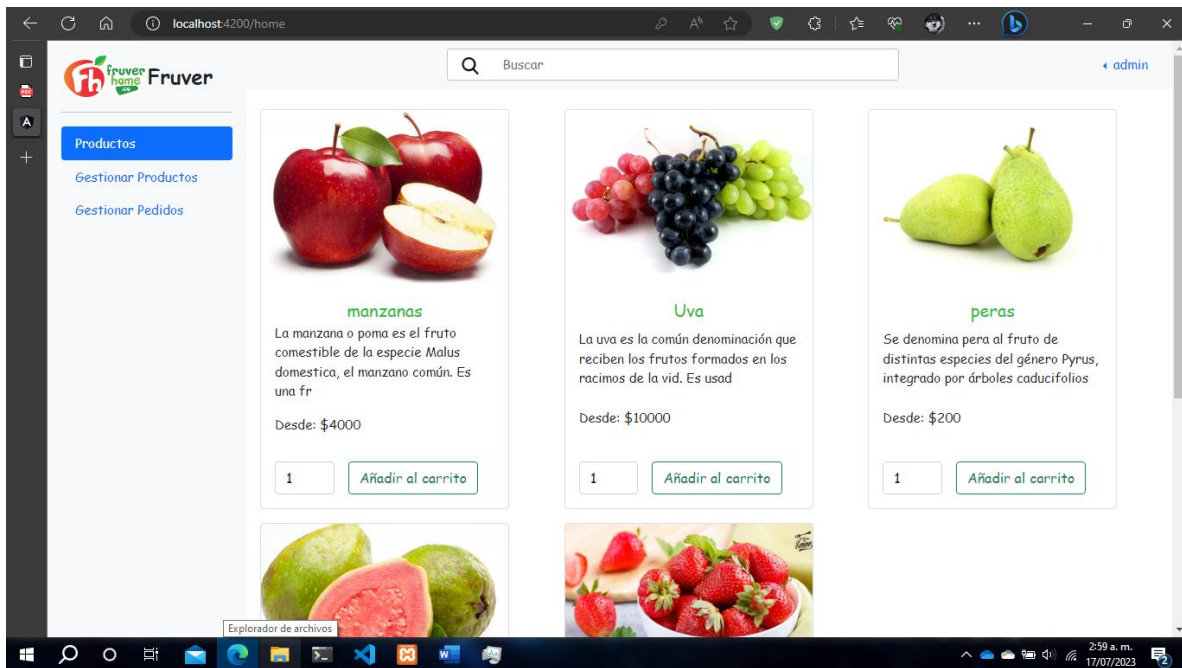
Edison Camilo

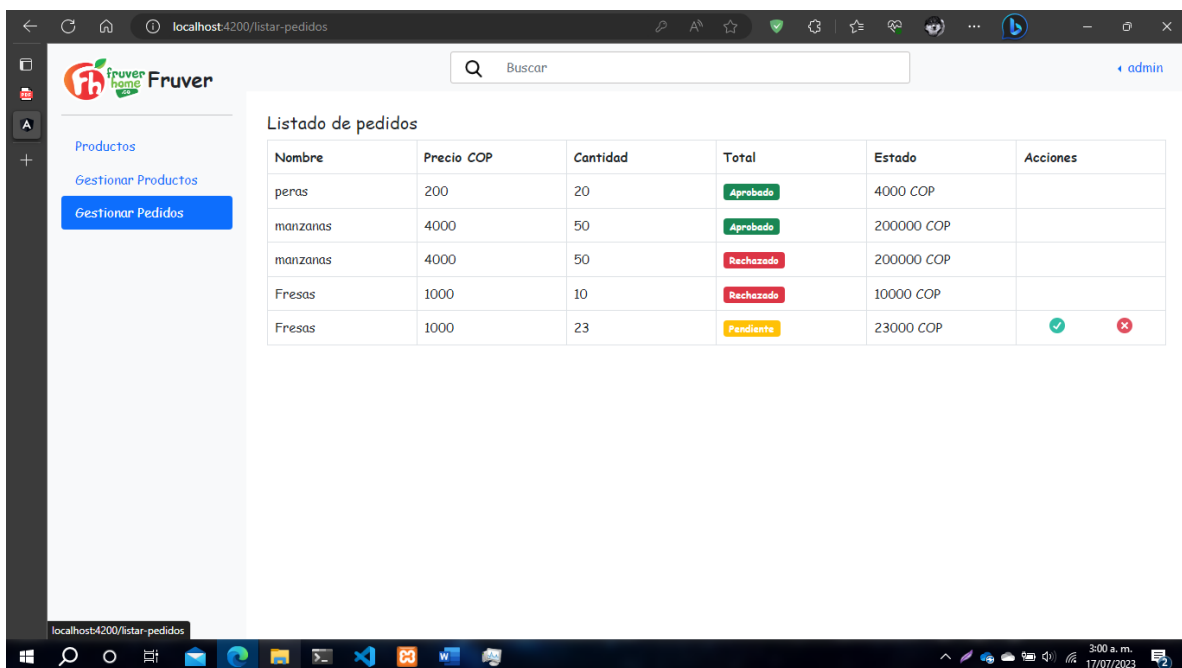
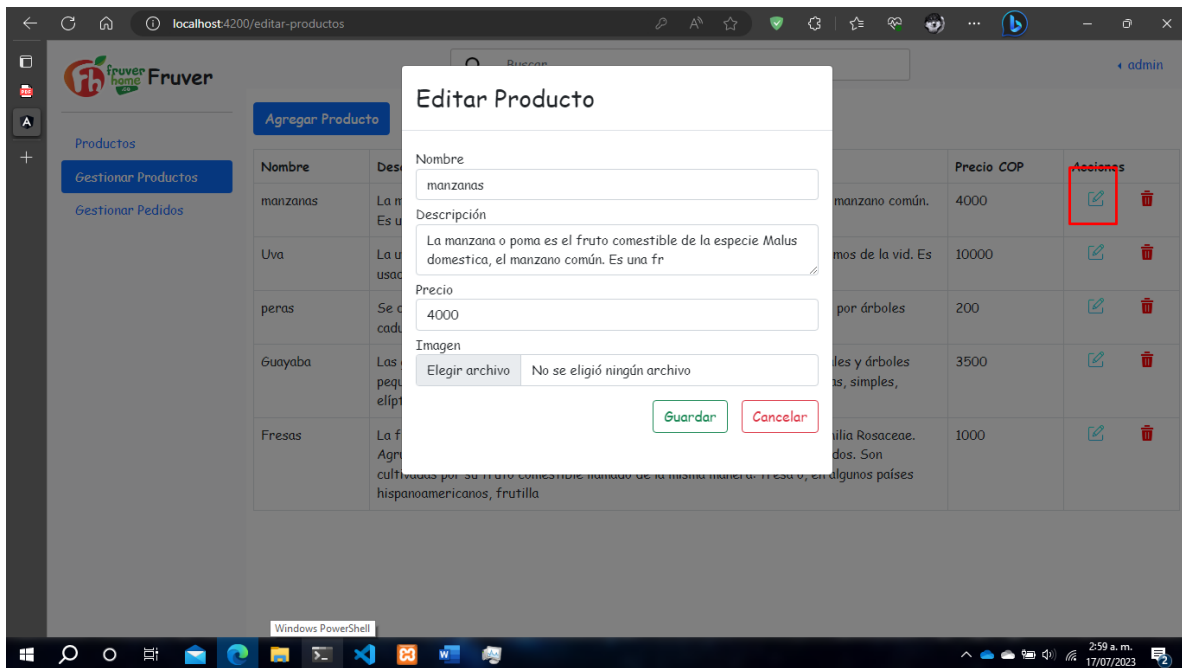
Listado de pedidos

Nombre	Precio COP	Cantidad	Total	Estado	Acciones
manzanas	4000	50	Aprobado	200000 COP	
manzanas	4000	50	Rechazado	200000 COP	
Fresas	1000	10	Rechazado	10000 COP	
Fresas	1000	7	Pendiente	7000 COP	 

2:57 a. m. 17/07/2023







Enlace git: <https://github.com/UnTalCamilo/frontendFruver.git>