

# Proyecto Web

## WebApp con Django

Nota

Grupo B	Escuela	Asignatura
<ul style="list-style-type: none"><li>■ Ccahuana Larota, Joshep Antony</li><li>■ Christian Zapana Romero, Pedro Luis</li><li>■ Mejia Ramos, Piero Douglas</li></ul>	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 1702122

Laboratorio	Tema	Duración
09	WebApp con Django	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 25 Julio 2023	Al 05 Agosto 2023

## 1. Marco teórico

### 1.1. Django Rest Framerowk:

- Django REST framework es un conjunto de herramientas potente y flexible para crear API web.
- Algunas razones por las que podrías querer usar el marco REST:
  - La API navegable por la Web es una gran ganancia de usabilidad para sus desarrolladores.
  - Políticas de autenticación que incluyen paquetes para OAuth1 y OAuth2.
  - Serialización que admite fuentes de datos ORM y no ORM.
  - Personalizable hasta el final: solo use las vistas regulares basadas en funciones si no necesita las funciones más potentes.
  - Amplia documentación y gran apoyo de la comunidad.
  - Utilizado y confiado por empresas reconocidas internacionalmente, como Mozilla, Red Hat, Heroku y Eventbrite.

## 1.2. Angular:

Angular es una plataforma completa que permite desarrollar aplicaciones web interactivas y dinámicas utilizando TypeScript como lenguaje de programación principal. Utiliza un enfoque basado en componentes, donde cada parte de la aplicación se divide en pequeños componentes reutilizables, lo que facilita la gestión y mantenimiento del código. Angular proporciona un sistema de enrutamiento que permite crear rutas dentro de la aplicación para una experiencia de navegación fluida y permite que los usuarios interactúen con la aplicación sin necesidad de recargar la página.

Algunas razones para usar Angular:

- **Desarrollo rápido:** Angular ofrece una estructura clara y modular que facilita el desarrollo rápido de aplicaciones. Los componentes reutilizables y la arquitectura bien definida permiten a los desarrolladores escribir menos código y lograr más funcionalidades.
- **Mantenibilidad:** La estructura basada en componentes de Angular y su sistema de inyección de dependencias facilitan la separación de preocupaciones, lo que hace que el código sea más fácil de mantener y actualizar.
- **SPA (Single Page Application):** Angular permite crear aplicaciones web de una sola página, lo que mejora la experiencia del usuario al reducir los tiempos de carga y mejorar la interactividad.
- **Soporte para TypeScript:** Angular está escrito en TypeScript, lo que proporciona ventajas como la detección de errores en tiempo de compilación, una mejor organización del código y una mayor productividad para los desarrolladores.
- **Amplia comunidad y documentación:** Angular cuenta con una gran comunidad de desarrolladores y una documentación completa y actualizada, lo que facilita el aprendizaje y resolución de problemas.
- **Actualizaciones regulares:** Google y la comunidad de Angular lanzan actualizaciones periódicas con nuevas características y mejoras, lo que garantiza que la plataforma esté en constante evolución y al día con las mejores prácticas de desarrollo web.

## 2. Aplicación

### 2.1. Fundamentos de desarrollo:

En el competitivo mundo de los negocios, la eficiencia y la gestión inteligente de los recursos son fundamentales para el éxito de cualquier empresa. En el caso de una tienda de abarrotes o una bodega de tamaño mediano, donde la variedad de productos y la demanda de los clientes pueden ser significativas, contar con una aplicación web personalizada se vuelve una necesidad imperante.

La organización y administración de inventarios, el control de ventas, la gestión de proveedores y la satisfacción del cliente son aspectos críticos para el buen funcionamiento de un negocio de este tipo. Tradicionalmente, muchas tiendas de abarrotes y bodegas han dependido de procesos manuales y registros en papel para llevar a cabo estas tareas, lo que puede llevar a ineficiencias, pérdida de tiempo y falta de precisión.

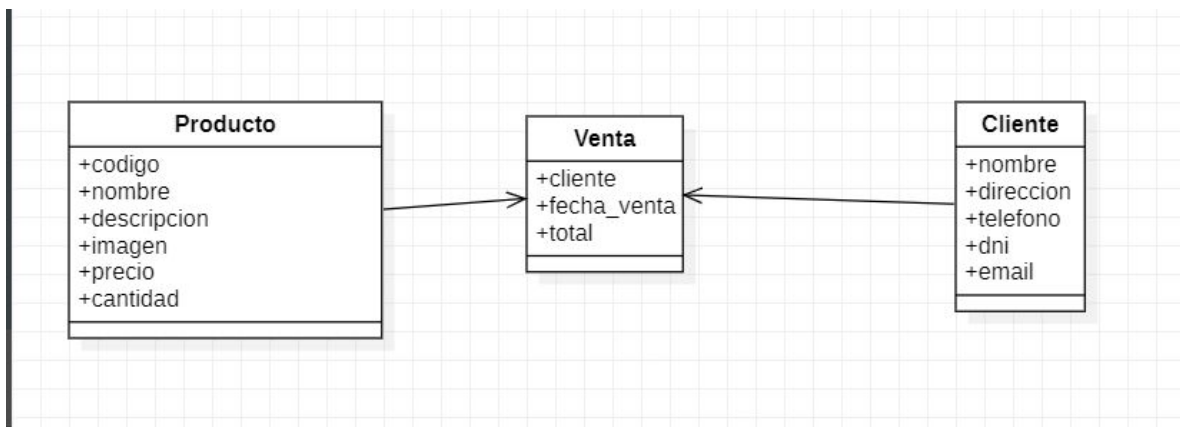
Es aquí donde una aplicación web diseñada específicamente para atender las necesidades de una tienda de abarrotes o bodega mediana se convierte en una solución inteligente y efectiva. Una aplicación web bien diseñada puede automatizar y optimizar procesos clave, brindando beneficios como:

1. Gestión de inventarios: Una aplicación web puede llevar un registro detallado de los productos en existencia, su cantidad, fecha de vencimiento y niveles de stock, facilitando la reposición de mercancía y evitando la falta de productos.
2. Control de ventas y caja: La aplicación puede llevar un registro preciso de las ventas diarias, permitiendo un seguimiento en tiempo real de los ingresos y la generación de informes detallados para análisis financiero.
3. Sistemas de punto de venta (POS): Una aplicación web puede integrar un sistema de punto de venta, agilizando las transacciones y brindando una experiencia de compra más rápida y satisfactoria para los clientes.
4. Gestión de proveedores: La aplicación puede mantener una base de datos de proveedores, facilitando la comunicación y la solicitud de mercancía de manera más eficiente.
5. Control de gastos y rentabilidad: Con la generación automática de informes financieros, se puede realizar un seguimiento de los gastos, márgenes de ganancia y rentabilidad del negocio.
6. Atención al cliente: Una aplicación web puede permitir la creación de programas de fidelización, cupones de descuento y promociones especiales para mantener satisfechos a los clientes y aumentar la lealtad hacia el negocio.

Para ello se plantea la propuesta de una aplicación para una tienda de tamaño mediano que apoye en la organización de inventario y control.

## 2.2. Organización de desarrollo:

Diagrama de entidad relación



Se presenta el diagrama de árbol de la distribución de los archivos usados dentro del proyecto "shop":

```

shop\
|-- assets
|   |-- admin
|   |-- css
|   |-- fonts
|   |-- images
|   |-- js
|
|-- crud

```

```
| |-- .angular
| |-- .vscode
| |-- node_modules
| |-- src
|     |-- app
|         |-- agregar
|         |-- eliminar
|         |-- footer
|         |-- header
|         |-- index
|         |-- list
|         |-- login
|         |-- modificar
|         |-- registro
|         |-- ...
|
|     |-- assets
|     |-- favicon.ico
|     |-- index.html
|     |-- main.ts
|     |-- styles.css
|
| |-- .editorconfig
| |-- .ignore
| |-- angular.json
| |-- package.json
| |-- package-lock.json
| |-- README.md
| |-- tsconfig.app.json
| |-- tsconfig.json
| |-- tsconfig.spec.json
|
|-- media
| |-- pics
|
|-- shop
| |-- __pycache__
| |-- __init__.py
| |-- asgi.py
| |-- settings.py
| |-- urls.py
| |-- wsgi.py
|
|-- static
| |-- css
| |-- fonts
| |-- images
| |-- js
|
|-- templates
| |-- agregar.html
| |-- base.html
| |-- elimar.html
| |-- iniciar_sesion.html
| |-- index.html
| |-- list.html
```

```
| |-- mod.html
| |-- registrar.html
|
|-- user
| |-- __pycache__
| |-- migrations
| |-- __init__.py
| |-- admin.py
| |-- apps.py
| |-- forms.py
| |-- models.py
| |-- tests.py
| |-- urls.py
| |-- views.py
|
|-- ventas
| |-- __pycache__
| |-- migrations
| |-- __init__.py
| |-- admin.py
| |-- apps.py
| |-- models.py
| |-- serializer.py
| |-- tests.py
| |-- urls.py
| |-- views.py
|
|-- db.sqlite3
|-- manage.py
```

## 2.3. Desarrollo:

### 2.3.1. Configuración del proyecto en Django:

#### ■ shop/settings.py

Agregación de las aplicaciones para usar: ventas, user, rest\_framework y corsheaders.

Listing 1: settings.py

```
1 INSTALLED_APPS = [
2     'django.contrib.admin',
3     'django.contrib.auth',
4     'django.contrib.contenttypes',
5     'django.contrib.sessions',
6     'django.contrib.messages',
7     'django.contrib.staticfiles',
8     'ventas',
9     'user',
10    'rest_framework',
11    'corsheaders',
12 ]
```

Agregación de la dirección de la carpeta a la cual consultar los templates.

Listing 2: settings.py

```
1 TEMPLATES = [  
2     {  
3         'BACKEND': 'django.template.backends.django.DjangoTemplates',  
4         'DIRS': [os.path.join(BASE_DIR, 'templates')],  
5         'APP_DIRS': True,  
6         'OPTIONS': {  
7             'context_processors': [  
8                 'django.template.context_processors.debug',  
9                 'django.template.context_processors.request',  
10                'django.contrib.auth.context_processors.auth',  
11                'django.contrib.messages.context_processors.messages',  
12            ],  
13        },  
14    },  
15 ]
```

Incorporación de la aplicación angular.

Listing 3: settings.py

```
1 CORS_ALLOW_ALL_ORIGINS = False  
2 CORS_ALLOWED_ORIGINS = [  
3     "http://localhost:4200", # Agrega aquí la URL de tu aplicación Angular  
4 ]
```

#### ■ shop/urls.py

Este código configura las rutas de URL para las aplicaciones "ventas" y "user", junto con la ruta para acceder al panel de administración. Además, asegura que los archivos de medios se sirvan correctamente durante el desarrollo del proyecto.

Listing 4: urls.py

```
1 urlpatterns = [  
2     path('', include('ventas.urls')),  
3     path('', include('user.urls')),  
4     path('admin/', admin.site.urls),  
5 ]  
6  
7 urlpatterns = urlpatterns + static(settings.MEDIA_URL,  
    document_root=settings.MEDIA_ROOT)
```

### 2.3.2. Aplicación ventas

#### ■ admin.py

El módulo admin.py, habilita la administración de los modelos "Producto" y "Venta" en la interfaz de administración de la aplicación Django, lo que facilita la gestión de los datos de estos modelos.

Listing 5: admin.py

```
1 from django.contrib import admin  
2 from .models import Producto, Venta  
3 # Register your models here.  
4  
5 admin.site.register(Producto)
```

```
6 admin.site.register(Venta)
```

#### ■ models.py

Importamos el modelo "Cliente" desde otra app para poder utilizarlo en la definición de modelos de la app actual. Esto permite establecer relaciones entre los modelos de ambas apps y utilizar la funcionalidad del modelo "Cliente" en la app actual.

Listing 6: models.py

```
1 from django.db import models
2 from user.models import Cliente
```

El código define el modelo "Producto" en Django con varios campos, incluyendo código, nombre, descripción, imagen, precio, cantidad y oferta. También define una función para mostrar el nombre del producto al imprimirlo.

Listing 7: models.py

```
1 class Producto(models.Model):
2     codigo = models.IntegerField()
3     nombre = models.CharField(max_length=100)
4     descripcion = models.TextField()
5     img = models.ImageField(upload_to='pics', default='')
6     precio = models.DecimalField(max_digits=10, decimal_places=2)
7     cantidad = models.PositiveIntegerField(default=0)
8     oferta = models.BooleanField(default=False)
9
10     def __str__(self):
11         return self.nombre
```

Se define el modelo "Venta" en Django con tres campos: cliente (una clave externa que relaciona la venta con un cliente), fecha\_venta (que guarda la fecha y hora actual de la venta), y total (que almacena el total de la venta en formato decimal). También define una función para mostrar información sobre la venta al imprimirlo, incluyendo el nombre del cliente y la fecha de la venta.

Listing 8: models.py

```
1 class Venta(models.Model):
2     cliente = models.ForeignKey(Cliente, on_delete=models.CASCADE)
3     fecha_venta = models.DateTimeField(auto_now_add=True)
4     total = models.DecimalField(max_digits=10, decimal_places=2, default=0)
5
6     def __str__(self):
7         return f"Venta del cliente: {self.cliente.nombre} - Fecha: {self.fecha_venta}"
```

#### ■ serializer.py

El código importa los modelos User, Group, y Producto de Django y el módulo serializers de Django REST Framework. Estos modelos pueden utilizarse para administrar usuarios, grupos y productos en la base de datos y se podrían crear serializadores para convertirlos en formato JSON y viceversa en una API.

Listing 9: serializer.py

```
1 from django.contrib.auth.models import User, Group
```

```
2 from rest_framework import serializers
3 from .models import Producto
```

A continuación el código define un serializador llamado `ProductoSerializer` utilizando la clase `ModelSerializer` de Django REST Framework. El serializador está configurado para trabajar con el modelo `Producto` y especifica los campos que serán serializados en formato JSON: código, nombre, descripción, img, precio y cantidad. Estos campos serán utilizados para representar y validar los datos del modelo `Producto` al interactuar con la API.

Listing 10: `serializer.py`

```
1 class ProductoSerializer(serializers.ModelSerializer):
2     class Meta:
3         model = Producto
4         fields = ('codigo', 'nombre', 'descripcion', 'img', "precio", "cantidad")
```

#### ■ `urls.py`

Configuración de las rutas URL para acceder a las vistas de la API utilizando Django REST Framework y las incluye en las URL globales del proyecto.

Listing 11: `urls.py`

```
1 from . import views
2 from django.urls import include, path
3 from django.contrib import admin
4 from rest_framework import routers
```

Configuración de una ruta URL para acceder a la vista `ProductoViewSet` en la API utilizando el enrutador `DefaultRouter`. La URL resultante será `"/producto/z` permitirá acceder a las operaciones CRUD relacionadas con el modelo `Producto` a través de la API RESTful.

Listing 12: `urls.py`

```
1 router = routers.DefaultRouter()
2 router.register(r'producto', views.ProductoViewSet)
```

Definición de diferentes rutas URL para las vistas de la aplicación.

Listing 13: `urls.py`

```
1 urlpatterns = [
2     path('agregar', views.agregar, name="agregar"),
3     path('modificar/<int:pro_id>', views.modificar, name="modificar"),
4     path('listar', views.lista, name="lista"),
5     path('eliminar/<int:pro_id>', views.eliminar, name="eliminar"),
6     path('admin/', admin.site.urls),
7     path('', include(router.urls)),
8     path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
9 ]
```

#### ■ `views.py`

1. `render`: Para renderizar plantillas HTML.
2. `redirect`: Para redireccionar a otras vistas.



3. `get_object_or_404`: Para obtener un objeto del modelo `Producto` o mostrar una página 404 si no se encuentra.
4. `Producto`: El modelo `Producto` definido en el archivo `models.py`.
5. `User` y `Group`: Modelos incorporados de Django para manejar usuarios y grupos.
6. `viewsets`: Proporciona una abstracción para trabajar con vistas basadas en conjuntos (`viewsets`) en Django REST Framework.
7. `ProductoSerializer`: El serializador definido para el modelo `Producto` en el archivo `serializer.py`.

Listing 14: `views.py`

```
1 from django.shortcuts import render, redirect, get_object_or_404
2 from .models import Producto
3 from django.shortcuts import render
4 from django.contrib.auth.models import User, Group
5 from rest_framework import viewsets
6 from .serializer import ProductoSerializer
```

La función `index` recupera todos los objetos del modelo `Producto` y los pasa al template `index.html` para ser mostrados en la vista.

Listing 15: `views.py`

```
1 def index(request):
2     products=Producto.objects.all()
3     return render(request, 'index.html', {'products':products})
```

La función `agregar` maneja la lógica para agregar un nuevo objeto `Producto` a la base de datos. Si la solicitud es de tipo `POST`, crea un nuevo objeto `Producto` con los datos recibidos del formulario y lo guarda en la base de datos. Luego, redirige al usuario a la página principal. Si la solicitud es de tipo `GET`, simplemente muestra el formulario para agregar un nuevo producto.

Listing 16: `views.py`

```
1 def agregar(request):
2     if request.method == 'POST':
3         producto = Producto()
4         producto.codigo=request.POST['codigo']
5         producto.nombre=request.POST['nombre']
6         producto.img=request.FILES.get('img')
7         producto.descripcion=request.POST['desc']
8         producto.precio=request.POST['precio']
9         producto.cantidad=request.POST['cantidad']
10        if 'oferta' in request.POST:
11            if request.POST['oferta'] == 'on':
12                producto.oferta = True
13        else:
14            producto.oferta = False
15        producto.save()
16        return redirect('/')
17    else:
18        return render(request, 'agregar.html')
```

La función modificar maneja la lógica para modificar un objeto Producto existente en la base de datos. Primero, obtiene el objeto Producto con el identificador (pro\_id) especificado en la URL utilizando la función `get_object_or_404`, que devuelve el objeto si existe o muestra una página de error 404 si no se encuentra.

Si la solicitud es de tipo POST, actualiza los campos del objeto Producto con los datos recibidos del formulario y lo guarda en la base de datos. Luego, redirige al usuario a la página principal. Si la solicitud es de tipo GET, muestra el formulario para modificar el producto, con los campos prellenados con los valores actuales del producto.

Listing 17: views.py

```
1 def modificar(request, pro_id):
2     prod = get_object_or_404(Producto, pk=pro_id)
3     if request.method == 'POST':
4         prod.codigo=request.POST['codigo']
5         prod.nombre=request.POST['nombre']
6         print(prod.img)
7         if prod.img != '':
8             prod.img=request.FILES.get('img')
9         prod.descripcion=request.POST['desc']
10        prod.precio=request.POST['precio']
11        prod.cantidad=request.POST['cantidad']
12        if 'oferta' in request.POST:
13            if request.POST['oferta'] == 'on':
14                prod.oferta = True
15        else:
16            prod.oferta = False
17        prod.save()
18        return redirect('/')
19    else:
20        return render(request, 'mod.html', {'prod': prod})
```

La función lista obtiene todos los objetos Producto existentes en la base de datos y los pasa al template 'list.html' para ser mostrados en una lista. Luego, devuelve el resultado renderizado al usuario.

Listing 18: views.py

```
1 def lista(request):
2     products=Producto.objects.all()
3     return render(request, 'list.html', {'products':products} )
```

La función eliminar busca un objeto Producto en la base de datos utilizando el identificador pro\_id. Si el método de la solicitud es POST, significa que se ha enviado una confirmación para eliminar el producto, por lo que se procede a eliminar el objeto Producto de la base de datos y luego redirige al usuario a la página principal (/).

Si el método de la solicitud no es POST, significa que el usuario aún no ha confirmado la eliminación y se muestra la plantilla `eliminar.html` con la información del producto para que el usuario pueda confirmar la eliminación.

Listing 19: views.py

```
1 def eliminar(request, pro_id):
2     prod = get_object_or_404(Producto, pk=pro_id)
3     if request.method == 'POST':
```

```
4     prod.delete()
5     return redirect('/')
6 else:
7     return render(request, 'eliminar.html', {'prod': prod})
```

La clase `ProductoViewSet` define una vista basada en `ModelViewSet` que utiliza el modelo `Producto` y el serializador `ProductoSerializer`. Proporciona funcionalidades CRUD para interactuar con los objetos `Producto` a través de la API RESTful.

Listing 20: views.py

```
1 class ProductoViewSet(viewsets.ModelViewSet):
2     queryset = Producto.objects.all()
3     serializer_class = ProductoSerializer
```

### 2.3.3. Aplicación user

#### ■ admin.py

El código registra el modelo `Cliente` en la interfaz de administración de Django para que pueda ser administrado y gestionado a través del panel de administración del sitio web. Esto permite a los usuarios autorizados ver, agregar, modificar y eliminar objetos `Cliente` desde el área de administración del sitio.

Listing 21: admin.py

```
1 from django.contrib import admin
2 from .models import Cliente
3 # Register your models here.
4 admin.site.register(Cliente)
```

#### ■ forms.py

El código importa las clases necesarias para trabajar con formularios de Django, incluyendo el formulario de creación de usuarios y el modelo de usuario.

Listing 22: forms.py

```
1 from django import forms
2 from django.contrib.auth.forms import UserCreationForm
3 from django.contrib.auth.models import User
```

El código define una clase `CustomUserCreationForm` que hereda del formulario de creación de usuarios predeterminado `UserCreationForm` en Django. Además de los campos de usuario y contraseña, este formulario personalizado agrega cuatro campos adicionales: email, numero, dni, y direccion. Estos campos permiten ingresar información adicional al crear un nuevo usuario en la aplicación. La clase `Meta` se utiliza para definir el modelo `User` al que se asocia el formulario y los campos que deben mostrarse en el formulario.

Listing 23: forms.py

```
1 class CustomUserCreationForm(UserCreationForm):
2     email = forms.EmailField()
3     numero = forms.CharField(max_length=15)
4     dni = forms.CharField(max_length=15)
5     direccion = forms.CharField(max_length=100)
```

```
6
7 class Meta:
8     model = User
9     fields = ['username', 'email', 'numero', 'dni', 'direccion', 'password1',
               'password2']
```

#### ■ models.py

El código define el modelo Cliente en Django con cuatro campos: nombre, direccion, telefono, y dni. Además, se ha definido el método `__str__` para que al imprimir un objeto Cliente, se muestre su nombre.

Listing 24: models.py

```
1 class Cliente(models.Model):
2     nombre = models.CharField(max_length=100)
3     direccion = models.CharField(max_length=200)
4     telefono = models.CharField(max_length=20)
5     dni = models.IntegerField()
6
7     def __str__(self):
8         return self.nombre
```

#### ■ urls.py

Definición de diferentes rutas URL para las vistas de la aplicación.

Listing 25: urls.py

```
1 admin.site.register(Cliente)
```

#### ■ views.py

El código importa módulos y funciones necesarias para el manejo de usuarios en Django, incluyendo formularios personalizados para crear y autenticar usuarios, el modelo de usuario predefinido, y funciones para realizar el inicio y cierre de sesión de usuarios.

Listing 26: views.py

```
1 from django.shortcuts import render, redirect
2
3 # Formulario para crear usuario
4 from .forms import CustomUserCreationForm
5 from django.contrib.auth.forms import UserCreationForm
6
7 # Formulario para comprobar si un usuario existe, inciar sesion
8 from django.contrib.auth.forms import AuthenticationForm
9
10 from django.contrib.auth.models import User
11 from django.db import IntegrityError
12
13 # Crear usuario
14 from django.contrib.auth import login
15
16 # Salir de sesion
17 from django.contrib.auth import logout
18
```

```
19 # Autenticar usuario
20 from django.contrib.auth import authenticate
```

Esta función maneja el registro de nuevos usuarios en el sistema. Si la solicitud es de tipo GET, se muestra el formulario de registro (CustomUserCreationForm) en la página 'registrar.html'. Si la solicitud es de tipo POST, se verifica que las contraseñas ingresadas coincidan y luego intenta crear un nuevo superusuario (administrador) en la base de datos utilizando los datos proporcionados en el formulario. Si el usuario ya existe, se muestra un mensaje de error en el formulario. Si las contraseñas no coinciden, también se muestra un mensaje de error. Luego, se inicia sesión con el nuevo usuario creado y se redirige a la página 'lista'.

Listing 27: views.py

```
1 def registro(request):
2     if request.method == 'GET':
3         return render(request, 'registrar.html', {"form": CustomUserCreationForm})
4
5     else:
6         if request.POST["password1"] == request.POST["password2"]:
7             try:
8                 user = User.objects.create_superuser(
9                     request.POST["username"], password=request.POST["password1"])
10                user.save()
11                # Cokiee del usuario que incia con la creacion de cuenta
12                login(request, user)
13                return redirect('lista')
14                # El error de IntegrityError => error especifico cuando se quiere crear un
15                # superuser que ya existe
16            except IntegrityError:
17                return render(request, 'registrar.html', {"form": CustomUserCreationForm,
18                                                            "error": "El usuario ya existe"})
19
20        return render(request, 'registrar.html', {"form": CustomUserCreationForm,
21                                                    "error": "Las contraseas no coinciden"})
```

La función cerrar\_sesion se encarga de cerrar la sesión del usuario actual y redireccionarlo a la página de inicio ('index').

Listing 28: views.py

```
1 def cerrar_sesion(request):
2     logout(request)
3     return redirect('index')
```

La función inciar\_sesion maneja el proceso de inicio de sesión de un usuario. Si el método de la solicitud es GET, se muestra el formulario de inicio de sesión (inciar\_sesion.html) con el formulario AuthenticationForm. Si el método de la solicitud es POST, se autentica al usuario utilizando las credenciales proporcionadas en el formulario de inicio de sesión. Si las credenciales son válidas, el usuario se autentica y se redirige a la página "lista". Si las credenciales no son válidas, se muestra un mensaje de error en la página de inicio de sesión.

Listing 29: views.py

```
1 def inciar_sesion(request):
2     if request.method == 'GET':
3         return render(request, 'inciar_sesion.html', {"form": AuthenticationForm})
```

```
4
5     else:
6         user = authenticate(
7             request, username=request.POST['username'],
8             password=request.POST['password'])
9
10        if user is None:
11            return render(request, 'inciar_sesion.html', {
12                "form": AuthenticationForm,
13                'error': 'Nombre de usuario o contrasea no valida'
14            })
15        else:
16            login(request, user)
17            return redirect('lista')
```

#### 2.3.4. CRUD en Angular:

##### ■ agregar

El código define el componente `.AgregarComponent`.<sup>en</sup> Angular, que permite agregar un nuevo producto mediante el envío de un formulario. Se utiliza el servicio `"HttpClient"` para realizar una solicitud HTTP POST al servidor con los datos del producto, incluyendo una imagen si se selecciona. La respuesta del servidor se muestra en la consola del navegador, indicando si el producto fue agregado con éxito o si ocurrió algún error.

Listing 30: agregar.component.ts

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-agregar',
  templateUrl: './agregar.component.html',
  styleUrls: ['./agregar.component.css']
})
export class AgregarComponent {
  nuevoProducto: any = {
    codigo: 0,
    nombre: '',
    descripcion: '',
    img: null,
    precio: 0,
    cantidad: 0,
    oferta: false
  };
  selectedFile: File | null = null;

  constructor(private http: HttpClient) { }

  ngOnInit(): void {
  }

  agregarProducto(): void {
    const formData = new FormData();
    formData.append('codigo', this.nuevoProducto.codigo.toString());
    formData.append('nombre', this.nuevoProducto.nombre);
```

```

    formData.append('descripcion', this.nuevoProducto.descripcion);
    if (this.selectedFile) {
        formData.append('img', this.selectedFile, this.selectedFile.name);
    }
    formData.append('precio', this.nuevoProducto.precio.toString());
    formData.append('cantidad', this.nuevoProducto.cantidad.toString());
    formData.append('oferta', this.nuevoProducto.oferta.toString());

    this.http.post('http://localhost:8000/producto/', formData)
        .subscribe(
            (response) => {
                console.log('Producto agregado con éxito:', response);
                // Puedes redirigir al usuario a la lista de productos o realizar otra acción
            },
            (error) => {
                console.error('Error al agregar el producto:', error);
            }
        );
}
onFileSelected(event: any): void {
    this.selectedFile = event.target.files[0] as File;
}
}

```

#### ■ eliminar

Este código corresponde a un componente de Angular llamado `EliminarComponent`. La función principal del componente es obtener los detalles de un producto a partir de su ID, mostrarlos en la página y permitir eliminar el producto mediante una solicitud HTTP DELETE al servidor.

Funciones principales:

1. `ngOnInit()`: Se ejecuta cuando el componente se inicializa. Obtiene el ID del producto desde la URL y realiza una solicitud HTTP GET al servidor para obtener los detalles del producto correspondiente.
2. `eliminarProducto()`: Se ejecuta cuando el usuario confirma la eliminación del producto. Realiza una solicitud HTTP DELETE al servidor para eliminar el producto correspondiente.

Listing 31: `eliminar.component.ts`

```

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { HttpClient } from '@angular/common/http';

@Component({
    selector: 'app-eliminar',
    templateUrl: './eliminar.component.html',
    styleUrls: ['./eliminar.component.css']
})
export class EliminarComponent implements OnInit {
    productoId: number = 0;
    prod: any = {};
    constructor(private route: ActivatedRoute, private http: HttpClient, private router: Router) {}
    ngOnInit(): void {
        const idParam = this.route.snapshot.paramMap.get('id');
    }
}

```

```
if (idParam !== null) {
  this.productoId = +idParam;
} else {
  console.log("No puede ser null")
}
this.http.get<any>('http://localhost:8000/producto/${this.productoId}/').subscribe(
  (data) => {
    this.prod = data;
    console.log(this.prod.img)
  },
  (error) => {
    console.error('Error al obtener los detalles del producto:', error);
  }
);
}

eliminarProducto(): void {
  this.http.delete('http://localhost:8000/producto/${this.productoId}/')
    .subscribe(
      () => {
        console.log('Producto eliminado con éxito');
        this.router.navigate(['']);
      },
      (error) => {
        console.error('Error al eliminar el producto:', error);
      }
    );
}
}
```

#### ■ footer

El código define el componente "FooterComponent".<sup>en</sup> Angular, que se utiliza para mostrar el pie de página de la aplicación. No contiene funciones adicionales.

Listing 32: footer.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-footer',
  templateUrl: './footer.component.html',
  styleUrls: ['./footer.component.css']
})
export class FooterComponent {
}
```

#### ■ header

El código define el componente "HeaderComponent".<sup>en</sup> Angular, que se utiliza para mostrar el encabezado de la aplicación. No contiene funciones adicionales.

Listing 33: header.component.ts

```
import { Component } from '@angular/core';
```



```
@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent {
}
```

## ■ index

El código define un componente de Angular llamado "IndexComponent" que muestra una lista de productos obtenidos a través del servicio `^piService`. Al inicializarse el componente, se llama a la función `getProductos()` para obtener los datos y asignarlos a la variable `productos`, que luego se muestra en la plantilla HTML asociada.

Listing 34: index.component.ts

```
import { Component } from '@angular/core';
import { ApiService } from '../api.service';

@Component({
  selector: 'app-index',
  templateUrl: './index.component.html',
  styleUrls: ['./index.component.css']
})
export class IndexComponent {
  productos: any[] = [];

  constructor(private api: ApiService) {
    this.getProductos();
  }

  getProductos = () => {
    this.api.getAllProductos().subscribe (
      data => {
        console.log(data);
        this.productos = data;
      },
      error => {
        console.log(error);
      }
    )
  }
}
```

## ■ list

Para esta sección muestra una lista de productos obtenidos a través del servicio `^piService` que permite navegar hacia las páginas de modificación o eliminación de un producto específico.

1. `@Component(...)`: Es el decorador que define el componente. Aquí se especifica el selector (`'app-list'`), la plantilla HTML (`templateUrl`), y la hoja de estilos (`styleUrls`) para el componente. También se declara el proveedor del servicio `^piService` para que esté disponible en este componente.

2. productos: any[] = []; Es una variable que almacenará la lista de productos obtenida desde el servicio.
3. productoId: number = 0; Es una variable que almacena el ID del producto seleccionado.
4. prod: any = {}; Es una variable que almacenará la información detallada de un producto específico.
5. constructor(...): Es el constructor del componente. Aquí se inyectan los servicios .^piService", .^activatedRoute", "HttpClient", Router". También se llama a la función "getProductos()" para obtener la lista de productos al inicializar el componente.
6. getProductos(): Es una función que utiliza el servicio .^piService" para obtener todos los productos y los asigna a la variable "productos".
7. irAModificar(\_url: string): Es una función que extrae el ID del producto de la URL y navega hacia la página de modificación del producto utilizando el servicio Router".
8. irAEliminar(\_url: string): Es una función similar a irAModificar()" pero navega hacia la página de eliminación del producto.
9. obtenerUltimoNumeroDeURL(url: string): number — null: Es una función auxiliar que extrae el último número de una URL. Se utiliza para obtener el ID del producto de la URL.

Listing 35: list.component.ts

```
import { Component } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { HttpClient } from '@angular/common/http';
import { ApiService } from '../api.service'; // Asegrate de importar tu servicio
    ApiService

@Component({
  selector: 'app-list',
  templateUrl: './list.component.html',
  styleUrls: ['./list.component.css'],
  providers: [ApiService]
})
export class ListComponent {
  productos: any[] = [];
  productoId: number = 0;
  prod: any = {};

  constructor(private api: ApiService, private route: ActivatedRoute, private http:
    HttpClient, private router: Router) {
    this.getProductos();
  }

  getProductos = () => {
    this.api.getAllProductos().subscribe (
      data => {
        console.log(data);
        this.productos = data;
      },
      error => {
        console.log(error);
      }
    );
  }
}
```

```
irAModificar(_url: string): void {
  const idProducto = this.obtenerUltimoNumeroDeURL(_url)
  this.router.navigate(['/modificar', idProducto]);
}
irAEliminar(_url: string): void {
  const idProducto = this.obtenerUltimoNumeroDeURL(_url)
  this.router.navigate(['/eliminar', idProducto]);
}

obtenerUltimoNumeroDeURL(url: string): number | null {
  const matchResult = url.match(/(\d+)\$/);
  if (matchResult) {
    return +matchResult[1];
  }
  return null;
}
}
```

#### ■ login

El código define el componente "LoginComponent.<sup>en</sup> Angular, que se utiliza para manejar el inicio de sesión en la aplicación. El componente contiene dos variables, `username` y `password`, que se utilizan para almacenar las credenciales de inicio de sesión ingresadas por el usuario.

El componente tiene un constructor que inyecta los servicios `AuthService` y `Router` para manejar la autenticación y la navegación en la aplicación, respectivamente.

Además, el componente contiene una función `iniciarSesion()` que se llama cuando el usuario intenta iniciar sesión. Dentro de esta función, se crea un objeto `credenciales` que contiene el nombre de usuario y la contraseña ingresados por el usuario. Sin embargo, en el código proporcionado, la función no realiza ninguna acción adicional y debe completarse para enviar las credenciales al servidor y procesar la respuesta de autenticación.

Listing 36: login.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../auth.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  username: string = '';
  password: string = '';

  constructor(private authService: AuthService, private router: Router) {}

  iniciarSesion() {
    const credenciales = {
      username: this.username,
      password: this.password
    };
  }
}
```

```
}  
}
```

#### ■ modificar

El código define el componente "ModificarComponent".<sup>en</sup> Angular para manejar la modificación de un producto. Utiliza servicios como "ActivatedRoute" "HttpClient" para obtener detalles del producto y realizar solicitudes HTTP al servidor. La función "modificarProducto()" actualiza el producto con los datos modificados mediante una solicitud HTTP de tipo PUT. La función "onFileSelected()" se llama cuando el usuario selecciona un archivo de imagen. La función "regresarInicio()" redirige al usuario a la página principal después de la modificación del producto.

Listing 37: modificar.component.ts

```
import { Component, OnInit } from '@angular/core';  
import { ActivatedRoute, Router } from '@angular/router';  
import { HttpClient } from '@angular/common/http';  
  
@Component({  
  selector: 'app-modificar',  
  templateUrl: './modificar.component.html',  
  styleUrls: ['./modificar.component.css']  
})  
export class ModificarComponent implements OnInit {  
  productoId: number = 0;  
  prod: any = {};  
  modificacionExitosa: boolean = false;  
  selectedFile: File | null = null;  
  
  constructor(private route: ActivatedRoute, private http: HttpClient, private router:  
    Router) { }  
  
  ngOnInit(): void {  
    const idParam = this.route.snapshot.paramMap.get('id');  
    if (idParam !== null) {  
      this.productoId = +idParam;  
    } else {  
      console.log("No puede ser null")  
    }  
    // Realizar solicitud HTTP para obtener los detalles del producto  
    this.http.get<any>('http://localhost:8000/producto/${this.productoId}/').subscribe(  
      (data) => {  
        this.prod = data;  
      },  
      (error) => {  
        console.error('Error al obtener los detalles del producto:', error);  
      }  
    );  
  }  
  
  modificarProducto(): void {  
    const formData = new FormData();  
    formData.append('codigo', this.prod.codigo);  
    formData.append('nombre', this.prod.nombre);  
    formData.append('descripcion', this.prod.descripcion);  
    if (this.selectedFile) {
```

```
        formData.append('img', this.selectedFile, this.selectedFile.name);
    }
    formData.append('precio', this.prod.precio);
    formData.append('cantidad', this.prod.cantidad);
    formData.append('oferta', this.prod.oferta);

    this.http.put('http://localhost:8000/producto/${this.productoId}/', formData)
        .subscribe(
            () => {
                console.log('Producto modificado con éxito');
                this.modificacionExitosa = true;
            },
            (error) => {
                console.error('Error al modificar el producto:', error);
            }
        );
}

onFileSelected(event: any): void {
    this.selectedFile = event.target.files[0] as File;
}

regresarInicio(): void {
    this.router.navigate(['/']); // Cambia la ruta segun tu configuracin
}
}
```

#### ■ registro

El código define el componente RegistroComponent.<sup>en</sup> Angular para manejar el registro de nuevos usuarios. Los datos del usuario se capturan mediante formularios y luego se verifica que las contraseñas coincidan. Luego, se crea un objeto usuario con los datos proporcionados y se llama al servicio `.authService.registrarUsuario()` para realizar una solicitud HTTP de registro. Si el registro es exitoso, el usuario es redirigido a la página principal; de lo contrario, se muestra un mensaje de error en la consola.

Listing 38: registro.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../auth.service';

@Component({
    selector: 'app-registro',
    templateUrl: './registro.component.html',
    styleUrls: ['./registro.component.css']
})
export class RegistroComponent implements OnInit {
    nombre: string = '';
    direccion: string = '';
    telefono: string = '';
    dni: number | null = null;
    password1: string = '';
    password2: string = '';
```

```
constructor(private authService: AuthService, private router: Router) { }

ngOnInit(): void {
}

registrarUsuario(): void {
  if (this.password1 !== this.password2) {
    console.error('Las contraseñas no coinciden');
    return;
  }

  const usuario = {
    username: this.nombre,
    direccion: this.direccion,
    telefono: this.telefono,
    dni: this.dni,
    password: this.password1
  };
  console.error(usuario);
  this.authService.registrarUsuario(usuario)
    .subscribe(
      () => {
        console.log('Usuario registrado con éxito');
        this.router.navigate(['/']);
      },
      error => {
        console.error('Error al registrar usuario:', error);
      }
    );
}
}
```

## 2.4. Capturas de ejecución

Iniciando el proyecto de Django:

```
Windows PowerShell
PS C:\Users\piro\OneDrive\Documentos\Universidad\pweb2\Lab_Grupal\trabFinal\shop> python .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified some issues:

WARNINGS:
?: (urls.N005) URL namespace 'admin' isn't unique. You may not be able to reverse all URLs in this namespace

System check identified 1 issue (0 silenced).
August 05, 2023 - 23:12:55
Django version 4.2.3, using settings 'shop.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[05/Aug/2023 23:12:58] "GET /producto/ HTTP/1.1" 200 1085
[05/Aug/2023 23:12:58] "GET /producto/ HTTP/1.1" 200 1085
[05/Aug/2023 23:12:58] "GET /media/pics/sal.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:12:58] "GET /media/pics/cerealChoco.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:12:58] "GET /media/pics/milo.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:12:58] "GET /media/pics/gelatina.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:12:58] "GET /media/pics/Aceite_LFQxE1w.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:13:10] "GET / HTTP/1.1" 200 5447
[05/Aug/2023 23:14:01] "GET /producto/ HTTP/1.1" 200 1085
[05/Aug/2023 23:16:14] "POST /producto/ HTTP/1.1" 201 196
[05/Aug/2023 23:16:22] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:22] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:22] "GET /media/pics/7756305001015.jpg HTTP/1.1" 200 46671
[05/Aug/2023 23:16:31] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:34] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:34] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:34] "GET /media/pics/7756305001015.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:17:25] "GET /producto/9/ HTTP/1.1" 200 196
[05/Aug/2023 23:17:25] "GET /media/pics/Aceite_LFQxE1w.jpg HTTP/1.1" 200 53138
[05/Aug/2023 23:18:46] "OPTIONS /producto/9/ HTTP/1.1" 200 0
[05/Aug/2023 23:18:46] "PUT /producto/9/ HTTP/1.1" 200 197
[05/Aug/2023 23:19:01] "GET /producto/ HTTP/1.1" 200 1283
[05/Aug/2023 23:19:01] "GET /media/pics/7756305001015.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:19:33] "GET /producto/9/ HTTP/1.1" 200 197
[05/Aug/2023 23:19:47] "DELETE /producto/9/ HTTP/1.1" 204 0
```

Iniciando Angular:

```
Windows PowerShell
PS C:\Users\piro\OneDrive\Documentos\Universidad\pweb2\Lab_Grupal\trabFinal\shop> python .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified some issues:






WARNINGS:
?: (urls.N005) URL namespace 'admin' isn't unique. You may not be able to reverse all URLs in this namespace

System check identified 1 issue (0 silenced).
August 05, 2023 - 23:12:55
Django version 4.2.3, using settings 'shop.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

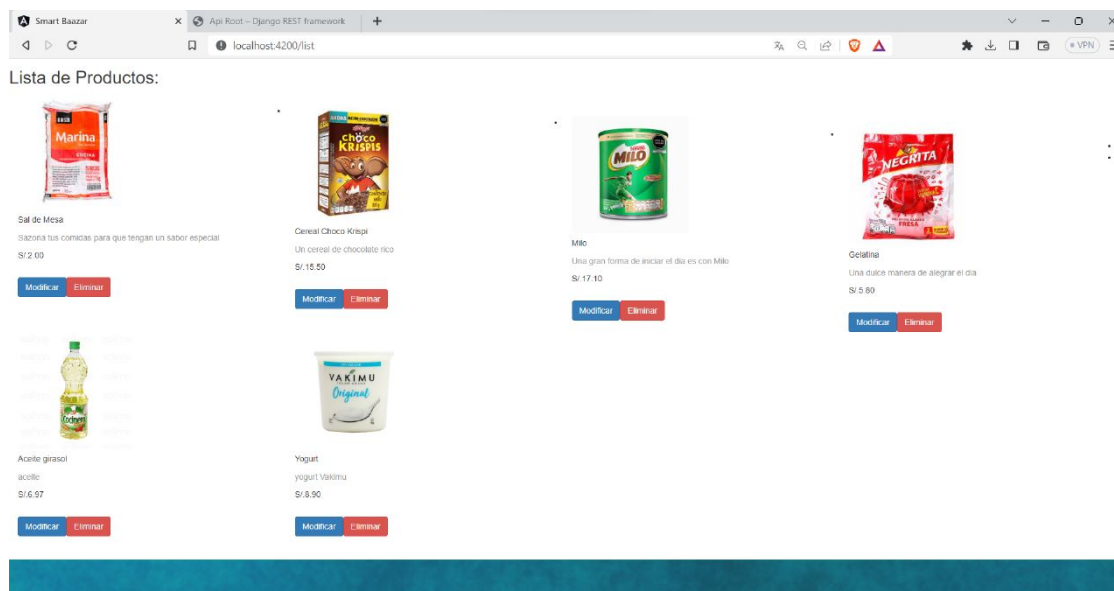
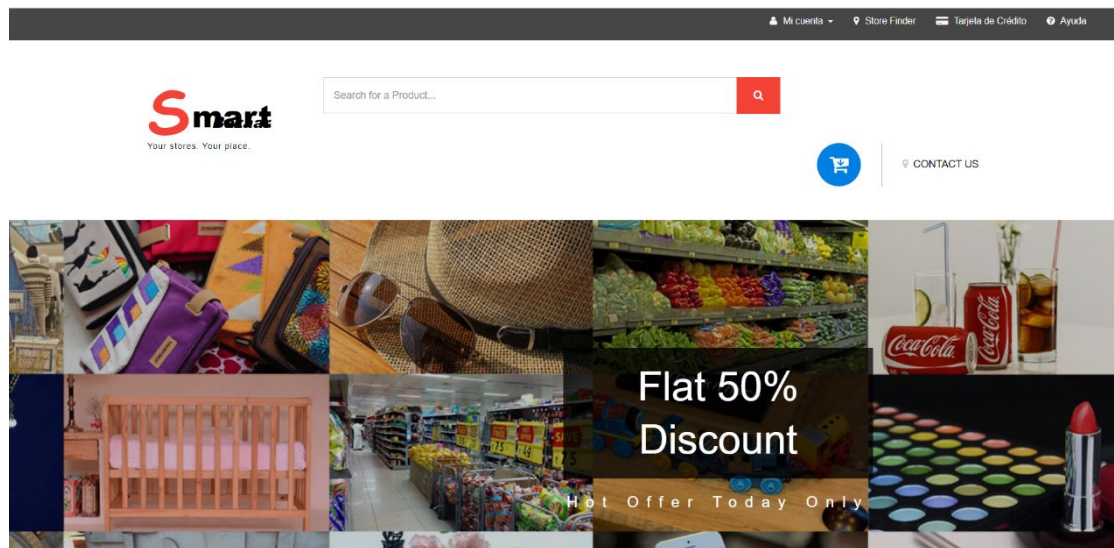
[05/Aug/2023 23:12:58] "GET /producto/ HTTP/1.1" 200 1085
[05/Aug/2023 23:12:58] "GET /producto/ HTTP/1.1" 200 1085
[05/Aug/2023 23:12:58] "GET /media/pics/sal.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:12:58] "GET /media/pics/cerealChoco.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:12:58] "GET /media/pics/milo.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:12:58] "GET /media/pics/gelatina.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:12:58] "GET /media/pics/Aceite_LFQxE1w.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:13:10] "GET / HTTP/1.1" 200 5447
[05/Aug/2023 23:14:01] "GET /producto/ HTTP/1.1" 200 1085
[05/Aug/2023 23:16:14] "POST /producto/ HTTP/1.1" 201 196
[05/Aug/2023 23:16:22] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:22] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:22] "GET /media/pics/7756305001015.jpg HTTP/1.1" 200 46671
[05/Aug/2023 23:16:31] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:34] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:34] "GET /producto/ HTTP/1.1" 200 1282
[05/Aug/2023 23:16:34] "GET /media/pics/7756305001015.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:17:25] "GET /producto/9/ HTTP/1.1" 200 196
[05/Aug/2023 23:18:46] "OPTIONS /producto/9/ HTTP/1.1" 200 0
[05/Aug/2023 23:18:46] "PUT /producto/9/ HTTP/1.1" 200 197
[05/Aug/2023 23:19:01] "GET /producto/ HTTP/1.1" 200 1283
[05/Aug/2023 23:19:01] "GET /media/pics/7756305001015.jpg HTTP/1.1" 304 0
[05/Aug/2023 23:19:33] "GET /producto/9/ HTTP/1.1" 200 197
[05/Aug/2023 23:19:47] "DELETE /producto/9/ HTTP/1.1" 204 0
```

## 2.5. Capturas de ejecución de la aplicación en la web

### Nuestros Productos

 <b>Sal de Mesa</b> Sazona tus comidas para que tengan un sabor especial S/.2.00 <a href="#">Add to cart</a>	 <b>Cereal Choco Krispi</b> Un cereal de chocolate rico S/.15.50 <a href="#">Add to cart</a>	 <b>Milo</b> Una gran forma de iniciar el día es con Milo S/.17.10 <a href="#">Add to cart</a>	 <b>Gelatina</b> Una dulce manera de alegrar el día S/.5.80 <a href="#">Add to cart</a>
 <b>Acete girasol</b> aceite <a href="#">Add to cart</a>			





Smart Bazaar x Api Root - Django REST framework +

localhost:4200/modificar/9

### Aceite girasol

aceite  
El código del producto es 43123  
Cuesta S/ 10  
Cuentas con 6 unidades  
No está en oferta

Código:  
43123

Nombre:  
Aceite girasol

Imagen:  
Seleccionar archivo Sin archivos seleccionados

Descripción:  
aceite

Precio:  
10.00

Cantidad:  
6

Oferta: ☐

**Modificar**


El producto ha sido modificado exitosamente

**Regresar al inicio**

Smart Bazaar x Api Root - Django REST framework +

localhost:4200/modificar/9

### Modificar el artículo



### Aceite girasol

aceite  
El código del producto es 43123  
Cuesta S/ 6.97  
Cuentas con 6 unidades  
No está en oferta

Código:  
43123

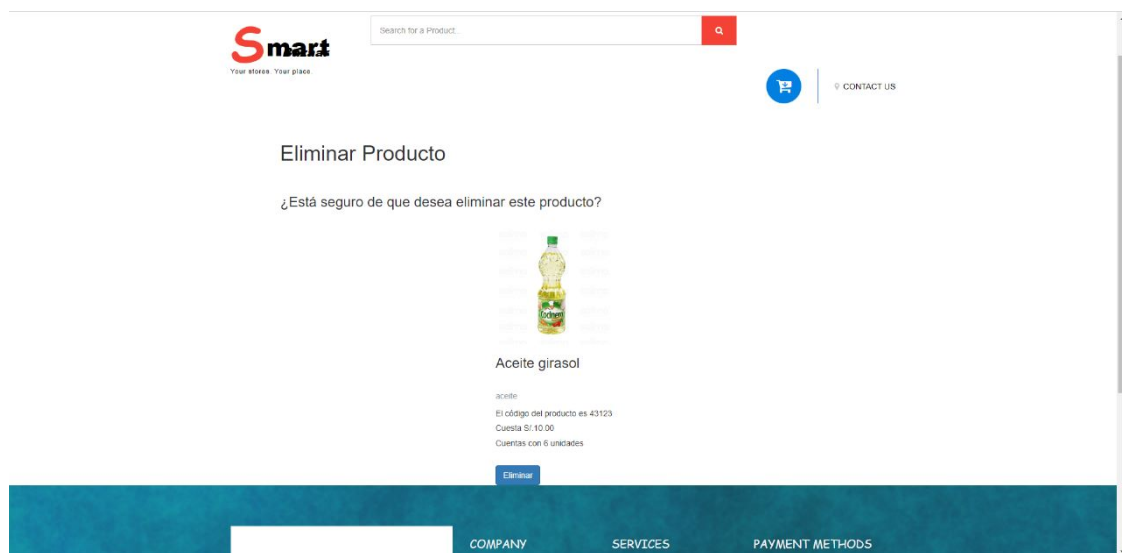
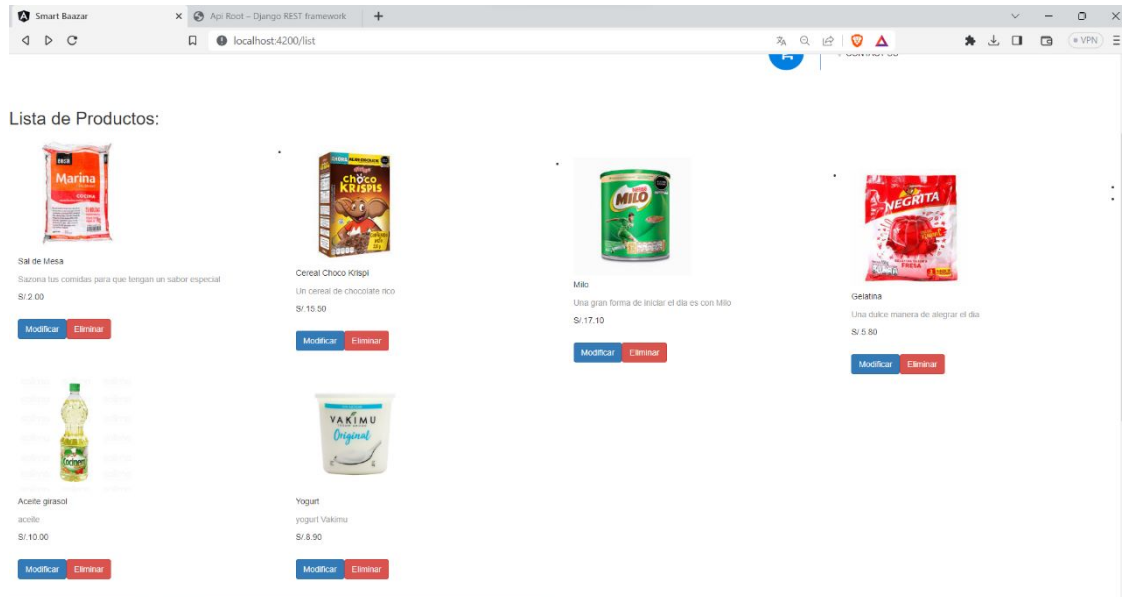
Nombre:  
Aceite girasol

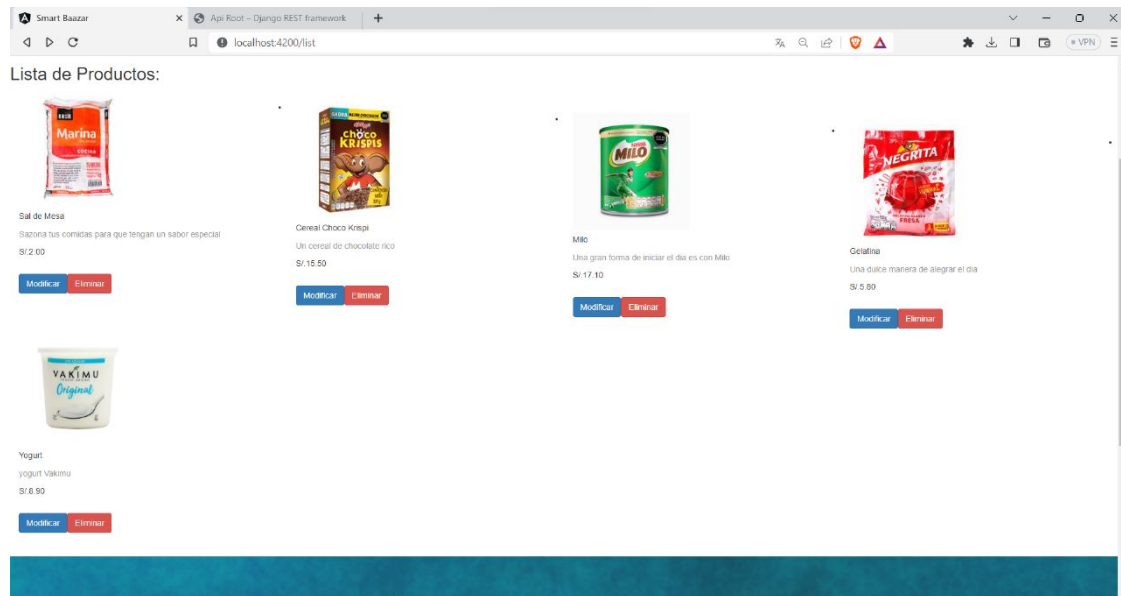
Imagen:  
Seleccionar archivo Sin archivos seleccionados

Descripción:  
aceite

Precio:  
6.97

Cantidad:  
6





### 3. URL de Referencias

Repositorio github de la aplicación

- [https://github.com/christian460/Lab\\_Grupal.git](https://github.com/christian460/Lab_Grupal.git)
- [https://github.com/christian460/Lab\\_Grupal/tree/0f044c54f26876b61072090568ea725e19df117e/trabFinal/shop](https://github.com/christian460/Lab_Grupal/tree/0f044c54f26876b61072090568ea725e19df117e/trabFinal/shop)