

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN
FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS



Programación web 2

Trabajo final

Integrantes:

Inca Moncca, Jherson David

Quispe Quispe, Fiorela Clariza (grupo solo de laboratorio)

Arequipa-Perú

2024

Descripción del Proyecto

El objetivo de este proyecto es desarrollar una aplicación web completa para el restaurante "Sabor Express". La aplicación permite gestionar el menú de productos, realizar pedidos, gestionar el carrito de compras, registrar usuarios, iniciar sesión y realizar el pago de órdenes. Además, cuenta con un dashboard administrativo para la gestión de productos, categorías y órdenes. El backend se desarrolla con Django y el frontend con Angular.

Requisitos Previos

- Python 3.x
- Node.js y npm
- Angular CLI
- Django
- Git
- AWS CLI (para despliegue)

Repositorios:

- Primer repositorio avance:

<https://github.com/FiorelaClarz/PROYECTO-FINAL-PWEB2>

- Repositorio final (oficial):

<https://github.com/jherson2024/restaurante>

Arquitectura del Sistema

La arquitectura del sistema sigue un modelo MVC (Model-View-Controller) en el backend y una arquitectura de componentes en el frontend. La comunicación entre el frontend y el backend se realiza mediante APIs RESTful.

Para proporcionar una API RESTful, se ha utilizado “django-cors-headers” junto con Django REST Framework, que permite manejar peticiones y respuestas en formato JSON de una manera eficiente y estructurada.

INSTALLED_APPS = [... 'rest_framework', 'corsheaders',]

Modelos de Datos

Modelo de Productos

```
class Producto(models.Model):  
    nombre = models.CharField(max_length=100)  
    descripcion = models.TextField(blank=True, null=True)  
    precio = models.DecimalField(max_digits=10, decimal_places=2)  
    categoria = models.ForeignKey(Categoría, on_delete=models.SET_NULL,  
null=True)
```

```

    imagen = models.ImageField(upload_to='static/images/', blank=True,
null=True)

    def __str__(self):
        return self.nombre

```

Modelo de Categorías

```

class Categoria(models.Model):
    nombre = models.CharField(max_length=100)
    descripcion = models.TextField(blank=True, null=True)

    def __str__(self):
        return self.nombre

```

Modelo de Clientes

```

class Cliente(models.Model):
    nombre = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    telefono = models.CharField(max_length=15, blank=True, null=True)
    direccion = models.TextField(blank=True, null=True)
    password = models.CharField(max_length=128,
default='defaultpassword')

    def __str__(self):
        return self.nombre

```

Modelo de Órdenes

```

class Orden(models.Model):
    cliente = models.ForeignKey(Cliente, on_delete=models.CASCADE)
    fecha = models.DateField()
    total = models.DecimalField(max_digits=10, decimal_places=2)
    estado = models.CharField(max_length=50, blank=True, null=True)

    def __str__(self):
        return f"Orden {self.id} de {self.cliente.nombre}"

```

Modelo de Detalles de Órdenes

```

class DetalleOrden(models.Model):
    orden = models.ForeignKey(Orden, on_delete=models.CASCADE)
    producto = models.ForeignKey(Producto, on_delete=models.CASCADE)
    cantidad = models.IntegerField()
    precio = models.DecimalField(max_digits=10, decimal_places=2) #dfs

```

```
def __str__(self):
    return f"{self.cantidad} x {self.producto.nombre}"
```

Modelo de Pagos

```
class Pago(models.Model):
    orden = models.ForeignKey(Orden, on_delete=models.CASCADE)
    fecha = models.DateField()
    metodo = models.CharField(max_length=50)
    monto = models.DecimalField(max_digits=10, decimal_places=2)
    estado=models.BooleanField(default=False)

    def __str__(self):
        return f"Pago de {self.monto} para la orden {self.orden.id}"
```

Modelo de Carritos

```
class Carrito(models.Model):
    cliente = models.OneToOneField(Cliente, on_delete=models.CASCADE)
    fecha_creacion = models.DateTimeField(auto_now_add=True)
    activo = models.BooleanField(default=True)

    def __str__(self):
        return f"Carrito de {self.cliente.nombre} - Activo: {self.activo}"
```

Modelo de Detalles de Carritos

```
class DetalleCarrito(models.Model):
    carrito = models.ForeignKey(Carrito, on_delete=models.CASCADE)
    producto = models.ForeignKey(Producto, on_delete=models.CASCADE)
```

Modelo de Usuarios

```
class User(models.Model):
    nombre = models.CharField(max_length=100, unique=True)
    password = models.CharField(max_length=128)

    def __str__(self):
        return self.nombre
```

Modelos con Clave Externa

En la aplicación de restaurante, se utilizan modelos con claves externas para definir relaciones entre diferentes entidades. Estas relaciones permiten organizar y gestionar los datos de manera más eficiente y mantener la integridad referencial en la base de datos. A continuación se presentan los modelos con claves externas y una descripción de cada uno:

Producto

Clave externa: categoría

Descripción: El modelo Producto incluye una clave externa a Categoría, indicando que cada producto pertenece a una categoría específica. Esto permite categorizar los productos y facilitar su organización y búsqueda.

Orden

Clave externa: cliente

Descripción: El modelo Orden está asociado a un Cliente, indicando que cada orden es realizada por un cliente específico. Esto permite rastrear las órdenes de cada cliente y gestionar su historial de compras.

DetalleOrden

Claves externas: orden, producto

Descripción: El modelo DetalleOrden incluye claves externas a Orden y Producto, indicando que cada detalle de orden se refiere a una orden específica y a un producto específico dentro de esa orden. Esto permite desglosar los productos incluidos en cada orden.

Pago

Clave externa: orden

Descripción: El modelo Pago está asociado a una Orden, indicando que cada pago corresponde a una orden específica. Esto facilita la gestión y seguimiento de los pagos realizados para cada orden.

Carrito

Clave externa: cliente

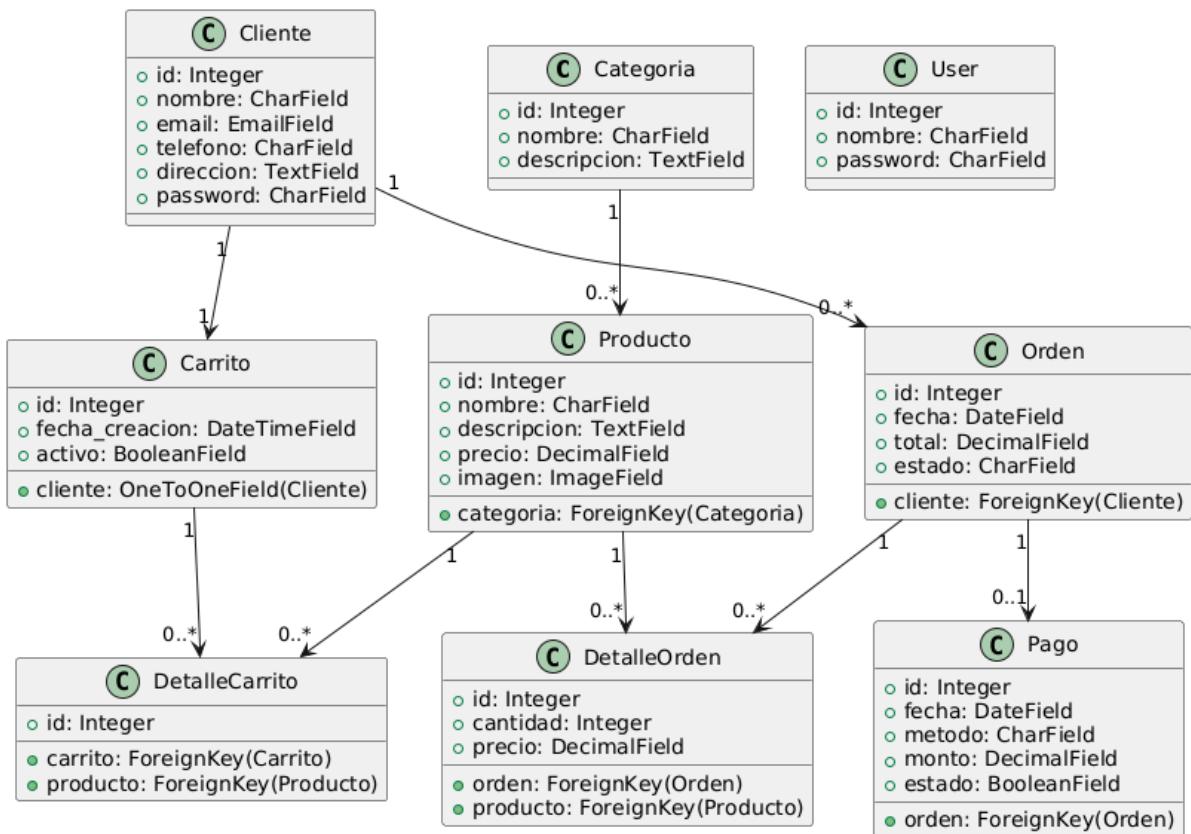
Descripción: El modelo Carrito está asociado a un Cliente, indicando que cada carrito pertenece a un cliente específico. Esto permite a los clientes agregar productos a su carrito antes de realizar una compra.

DetalleCarrito

Claves externas: carrito, producto

Descripción: El modelo DetalleCarrito incluye claves externas a Carrito y Producto, indicando que cada detalle de carrito se refiere a un producto específico dentro de un carrito específico. Esto facilita la gestión de los productos añadidos a los carritos de los clientes.

Diagrama entidad relación



Vistas de Listado (List View)

Estas vistas muestran un listado de objetos:

Listado de Productos

```
@csrf_exempt
@require_http_methods(["GET"])
def obtener_productos(request):
    productos = Producto.objects.all()
    productos_json = [
        {
            'nombre': producto.nombre,
            'descripcion': producto.descripcion,
            'precio': str(producto.precio),
            'categoria': producto.categoria.nombre if
            producto.categoria else None,
        }
    ]
    return JsonResponse(productos_json)
```

```

        'imagen_url':  

request.build_absolute_uri(f'/static/{producto.imagen_url}') if  

producto.imagen_url else None  

    }  

    for producto in productos  

]  

return JsonResponse(productos_json, safe=False)

```

Listado de Categorías

```

def obtener_categorias(request):  

    categorias = Categoria.objects.all()  

    categorias_list = [{'id': categoria.id, 'nombre': categoria.nombre}  

for categoria in categorias]  

    return JsonResponse(categorias_list, safe=False)

```

Listado de Órdenes

```

@csrf_exempt
def obtener_todas_las_ordenes(request):
    ordenes = Orden.objects.all().order_by('-fecha')
    ordenes_list = []
    for orden in ordenes:
        detalle_orden = DetalleOrden.objects.filter(orden=orden)
        orden_data = {
            'id': orden.id,
            'cliente': orden.cliente.nombre,
            'fecha': orden.fecha,
            'total': orden.total,
            'estado': orden.estado,
            'pagado': Pago.objects.filter(orden=orden,
estado=True).exists(),
            'detalles': list(detalle_orden.values('producto__nombre',
'cantidad', 'precio'))
        }
        ordenes_list.append(orden_data)
    return JsonResponse(ordenes_list, safe=False)

```

Vistas de Detalle (Detail View)

Estas vistas muestran el detalle de un objeto específico:

Detalle de Cliente

```

def get_cliente(request, cliente_id):
    cliente = get_object_or_404(Cliente, id=cliente_id)
    data = {
        'id': cliente.id,
        'nombre': cliente.nombre,
    }
    return JsonResponse(data)

```

Vistas de Crear (Create View)

Estas vistas permiten crear nuevos objetos:

Crear Producto

```

@csrf_exempt
@require_http_methods(['POST'])
def crear_producto(request):
    try:
        if request.content_type.startswith('multipart/form-data'):
            nombre = request.POST.get('nombre')
            descripcion = request.POST.get('descripcion', '')
            precio = request.POST.get('precio')
            categoria_id = request.POST.get('categoria_id')
            imagen = request.FILES.get('imagen')
            categoria = Categoria.objects.get(id=categoria_id) if categoria_id else None
            producto = Producto.objects.create(
                nombre=nombre,
                descripcion=descripcion,
                precio=precio,
                categoria=categoria,
                imagen=imagen # Guardar la imagen
            )
            return JsonResponse({'mensaje': 'Producto creado exitosamente', 'producto_id': producto.id}, status=201)
        else:
            return JsonResponse({'error': 'Tipo de contenido no soportado'}, status=400)
    except Categoria.DoesNotExist:
        return JsonResponse({'error': 'Categoría no encontrada'}, status=400)
    except Exception as e:
        return JsonResponse({'error': str(e)}, status=500)

```

Registrar Cliente

```
@csrf_exempt
def registrar_cliente(request):
    print("llego aqui")
    if request.method == 'POST':
        data = json.loads(request.body)
        nombre = data.get('nombre')
        email = data.get('email')
        telefono = data.get('telefono')
        password = data.get('password')
        if not (nombre and email and password):
            return JsonResponse({'error': 'Faltan campos obligatorios'}, status=400)
        if Cliente.objects.filter(email=email).exists():
            return JsonResponse({'error': 'El email ya está registrado'}, status=400)
        cliente = Cliente(nombre=nombre, email=email,
                           telefono=telefono, password=password)
        cliente.save()
        carrito = Carrito(cliente=cliente)
        carrito.save()
        return JsonResponse({'mensaje': 'Cliente registrado exitosamente'})
    return JsonResponse({'error': 'Método no permitido'}, status=405)
```

Crear Orden

```
class CrearOrdenView(APIView):
    def post(self, request):
        cliente_id = request.data.get('cliente_id')
        productos = request.data.get('productos')
        total = request.data.get('total')
        if not cliente_id:
            return Response({"error": "El cliente_id es requerido"}, status=status.HTTP_400_BAD_REQUEST)
        if not productos:
            return Response({"error": "Los productos son requeridos"}, status=status.HTTP_400_BAD_REQUEST)
        if not total:
            return Response({"error": "El total es requerido"}, status=status.HTTP_400_BAD_REQUEST)
        try:
            cliente = Cliente.objects.get(id=cliente_id)
```

```

        except Cliente.DoesNotExist:
            return Response({"error": "Cliente no encontrado"}, status=status.HTTP_404_NOT_FOUND)

        try:
            orden = Orden.objects.create(
                cliente=cliente,
                fecha=timezone.now().date(),
                total=total,
                estado='no pagado'
            )

            for prod in productos:
                producto = Producto.objects.get(id=prod['id'])
                DetalleOrden.objects.create(
                    orden=orden,
                    producto=producto,
                    cantidad=1,
                    precio=producto.precio
                )
        except Exception as e:
            return Response({"error": str(e)}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
    serializer = OrdenSerializer(orden)
    return Response(serializer.data, status=status.HTTP_201_CREATED)

```

Vistas de Borrar (Delete View)

Estas vistas nos permiten eliminar objetos:

Eliminar Producto

```

@csrf_exempt
@require_http_methods(['DELETE'])
def eliminar_producto(request, producto_id):
    try:
        producto = Producto.objects.get(id=producto_id)
        producto.delete()
        return JsonResponse({'mensaje': 'Producto eliminado exitosamente'}, status=200)
    except Producto.DoesNotExist:
        return JsonResponse({'error': 'Producto no encontrado'}, status=404)
    except Exception as e:
        return JsonResponse({'error': str(e)}, status=500)

```

Eliminar Categoría

```
@csrf_exempt
def eliminar_categoria(request, categoria_id):
    if request.method == 'DELETE':
        try:
            categoria = Categoria.objects.get(id=categoria_id)
            categoria.delete()
            return JsonResponse({'mensaje': 'Categoría eliminada'})
        except Categoria.DoesNotExist:
            return JsonResponse({'error': 'Categoría no encontrada'},
status=404)
```

Vistas que devuelven JSON

Utilizamos vistas que devuelven JSON para permitir la comunicación eficiente y estructurada entre el servidor y el cliente (en este caso, una aplicación web frontend). Utilizar vistas que devuelven JSON mejora la eficiencia, compatibilidad, y escalabilidad de la aplicación web, permitiendo una comunicación clara y rápida entre el frontend y el backend.

Vistas que devuelven JSON:

- productos (POST)
- obtener_precio_producto (POST)
- login_view (POST)
- obtener_productos (GET)
- carrito (POST)
- eliminar_del_carrito (DELETE)
- get_cliente (GET)
- registrar_cliente (POST)
- crear_producto (POST)
- eliminar_producto (DELETE)
- obtener_categorias (GET)
- enviar_boleta (POST)
- crear_categoria (POST)
- eliminar_categoria (DELETE)
- obtener_todas_las_ordenes (GET)
- marcar_orden_como_atendida (POST)

Programa cliente para hacer y consumir las consultas utilizando AJAX y Framework de JavaScript

En nuestra aplicación, utilizamos Angular como framework de JavaScript para el desarrollo del frontend. Angular nos permite realizar peticiones AJAX de manera eficiente y estructurada mediante su módulo HttpClient. En nuestro proyecto los servicios se encarga de interactuar con el backend Django mediante peticiones HTTP. Aquí por ejemplo en nuestro servicio “ ApiService ” se definen diversas funciones que realizan peticiones AJAX a los endpoints del backend y manejan las respuestas en formato JSON.

```

import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class ApiService {
  private baseUrl = 'http://18.226.164.178:8000/api/';
  constructor(private http: HttpClient) { }
  getProductos(): Observable<any> {
    return this.http.get(this.baseUrl + 'productos/');
  }
  getCarrito(clienteId: number): Observable<any> {
    return
  this.http.get<any>(`${this.baseUrl}carritos/obtener_carrito/`);
  }
  getProductosCarrito(clienteId: number): Observable<any> {
    return
  this.http.get<any>(`${this.baseUrl}productos_carrito/?cliente_id=${clienteId}`);
  }
  crearOrden(clienteId: number, total: number, productos: any[]): Observable<any> {
    return this.http.post<any>(`${this.baseUrl}crear-orden/`, {
      cliente_id: clienteId, total: total, productos: productos });
  }
  getClienteById(clienteId: number): Observable<any> {
    return this.http.get<any>(`${this.baseUrl}clientes/${clienteId}/`);
  }
  registrarCliente(data: { nombre: string, email: string, telefono: string, password: string }): Observable<any> {
    console.log("llego a api")
    return this.http.post<any>(`${this.baseUrl}registrar_cliente/`, data);
  }
  enviarBoleta(email: string, pdfData: string): Observable<any> {
    const url = `${this.baseUrl}enviar_boleta/`;
    const body = { email: email, pdf_data: pdfData };
    return this.http.post<any>(url, body);
  }
  agregarProductoAlCarrito(clienteId: number, productoId: number): Observable<any> {
    const data = {cliente_id: clienteId, producto_id: productoId};
  }
}

```

```

        return
    this.http.post<any>(`${this.baseUrl}carritos/agregar_producto/`, data);
}

eliminarProductoCarrito(clienteId: number, productoId: number): Observable<any> {
    const data = {cliente_id: clienteId, producto_id: productoId};
    return
    this.http.post<any>(`${this.baseUrl}carritos/eliminar_producto/`,
data);
}

eliminarCategoria(categoríaId: number): Observable<any> {
    return
    this.http.delete<any>(`${this.baseUrl}eliminar_categoria/${categoríaId}`);
}

obtenerTodasLasOrdenes(): Observable<any> {
    return
    this.http.get<any[]>(`${this.baseUrl}obtener_todas_las_ordenes/`);
}

marcarOrdenComoAtendida(ordenId: number): Observable<any> {
    return
    this.http.post<any>(`${this.baseUrl}marcar_orden_como_atendida/${ordenId}/`, {});
}

crearPago(pago: any): Observable<any> {
    const url = `${this.baseUrl}pagos/`;
    return this.http.post(url, pago);
}
}

```

Funcionalidades del sistema

1. Registro de cliente

Descripción: Permite a los nuevos usuarios registrarse en la plataforma proporcionando sus datos personales. Después de completar el registro, el usuario es redirigido automáticamente a la página de inicio de sesión del cliente.

Proceso:

1. El usuario completa el formulario de registro con su nombre, email, teléfono y contraseña.
2. Al enviar el formulario, se crea un nuevo cliente en la base de datos.

3. El usuario es redirigido a la página de inicio de sesión.

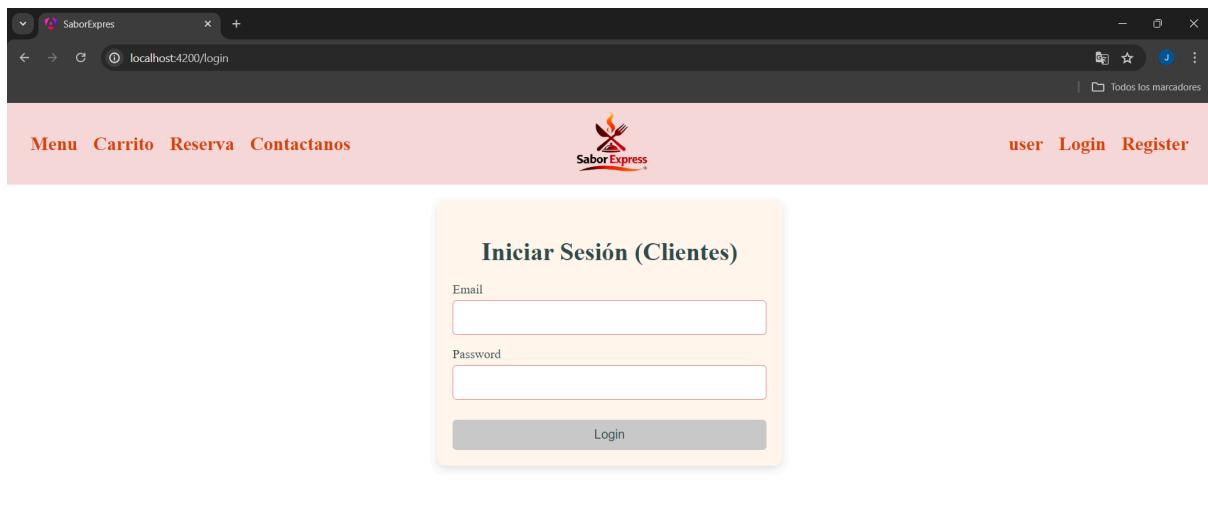
The screenshot shows a web browser window for 'SaborExpress' at 'localhost:4200/register'. The page has a light pink header with the logo 'Sabor Express' and navigation links 'Menu', 'Carrito', 'Reserva', 'Contactanos', 'user', 'Login', and 'Register'. The main content area is titled 'Registrar Cliente' (Register Client). It contains four input fields: 'Nombre' (Name), 'Email', 'Teléfono' (Phone), and 'Contraseña' (Password), each with a placeholder text inside. Below these fields is a large red 'Registrar' (Register) button.

2. Inicio de sesión de cliente

Descripción: Los clientes registrados pueden iniciar sesión en la plataforma para acceder a las funcionalidades del sistema. Después de iniciar sesión, el cliente es redirigido al menú de productos, donde su nombre aparece en la pantalla.

Proceso:

1. El cliente ingresa su email y contraseña en el formulario de inicio de sesión.
2. Si las credenciales son correctas, se autentica al cliente.
3. El cliente es redirigido al menú de productos y su nombre se muestra en la interfaz.



3. Menú de productos

Descripción: Muestra una lista de productos disponibles para la compra. Los clientes pueden seleccionar productos para añadir al carrito. El logo del carrito se actualiza para mostrar la cantidad de productos añadidos y, al hacer clic, redirige a los detalles del carrito.

Proceso:

1. El cliente visualiza la lista de productos.
2. Puede filtrar los productos por categoría.
3. Selecciona productos para añadir al carrito.

The screenshot shows the SaborExpress website menu page. At the top, there is a navigation bar with links for 'Menu', 'Carrito', 'Reserva', 'Contactanos', and a user account section ('user Logout' and 'Fiorela Clariza'). A shopping cart icon with a red '3' notification is visible. The main content area is titled 'Menú de Productos' and features three product cards:

- Ensalada César**: Described as 'Ensalada fresca con pollo, queso y aderezo César'. Price: 8.50. Add to cart button.
- Sopa de Tomate**: Described as 'Sopa cremosa de tomate con albahaca'. Price: 6.00. Add to cart button.
- Filete de Res**: Described as 'Filete de res a la parrilla con guarnición de papas'. Price: 15.00. Add to cart button.

4. Filtrado de productos por categoría

The screenshot shows the SaborExpress website menu page with a filter applied to the 'Bebidas' category. The main content area is titled 'Menú de Productos' and features two product cards:

- Coca Cola**: Described as 'Bebida gaseosa'. Price: 2.50. Add to cart button.
- Jugo de Naranja**: Described as 'Jugo de naranja natural'. Price: 3.00. Add to cart button.

5. Carrito de compra

Descripción: Muestra los productos añadidos al carrito, permitiendo a los clientes quitar productos y continuar al proceso de pago.

Proceso:

1. El cliente revisa los productos en su carrito.
2. Puede eliminar productos del carrito.

3. Continúa al proceso de pago.

The screenshot shows a web browser window for 'SaborExpress' at 'localhost:4200/cart'. The header includes navigation links 'Menu', 'Carrito', 'Reserva', 'Contactanos', a logo for 'Sabor Express', and user information 'user Logout' and 'Fiorela Clariza'. The main content is titled 'Carrito' and displays three items: 'Ensalada César' (Caesar salad), 'Sopa de Tomate' (Tomato soup), and 'Filete de Res' (Beef steak). Below each item is a small image, a name, a detailed description, and a price. At the bottom, it shows a total amount of '\$29.50'.

Item	Description	Precio
Ensalada César	Ensalada fresca con pollo, queso y aderezo César	8.50
Sopa de Tomate	Sopa cremosa de tomate con albahaca	6.00
Filete de Res	Filete de res a la parrilla con guarnición de papas	15.00

Total: \$29.50

6. Pago de Compras

Descripción: Presenta un método de pago, por ahora una simulación de pago por Yape. Después de simular el pago, el cliente es redirigido a la boleta de venta electrónica.

Proceso:

1. El cliente selecciona el método de pago.
2. Se realiza el pago.
3. El cliente es redirigido a la boleta de venta electrónica.

The screenshot shows a web browser window for 'SaborExpress' at 'localhost:4200/pago'. The header includes 'Menu', 'Carrito', 'Reserva', 'Contactanos', the restaurant logo, and user information 'user Logout Fiorela Clariza'. The main content is titled 'Pago de Compras' and 'Resumen de la Compra'. It lists three items: 'Ensalada César' (\$8.50), 'Sopa de Tomate' (\$6.00), and 'Filete de Res' (\$15.00). A total amount of '\$29.50' is displayed. Below this, there is a section titled 'Paga con Yape' with a QR code and a button 'Simular Escaneo con el QR'.

7. Boleta de venta Electrónica

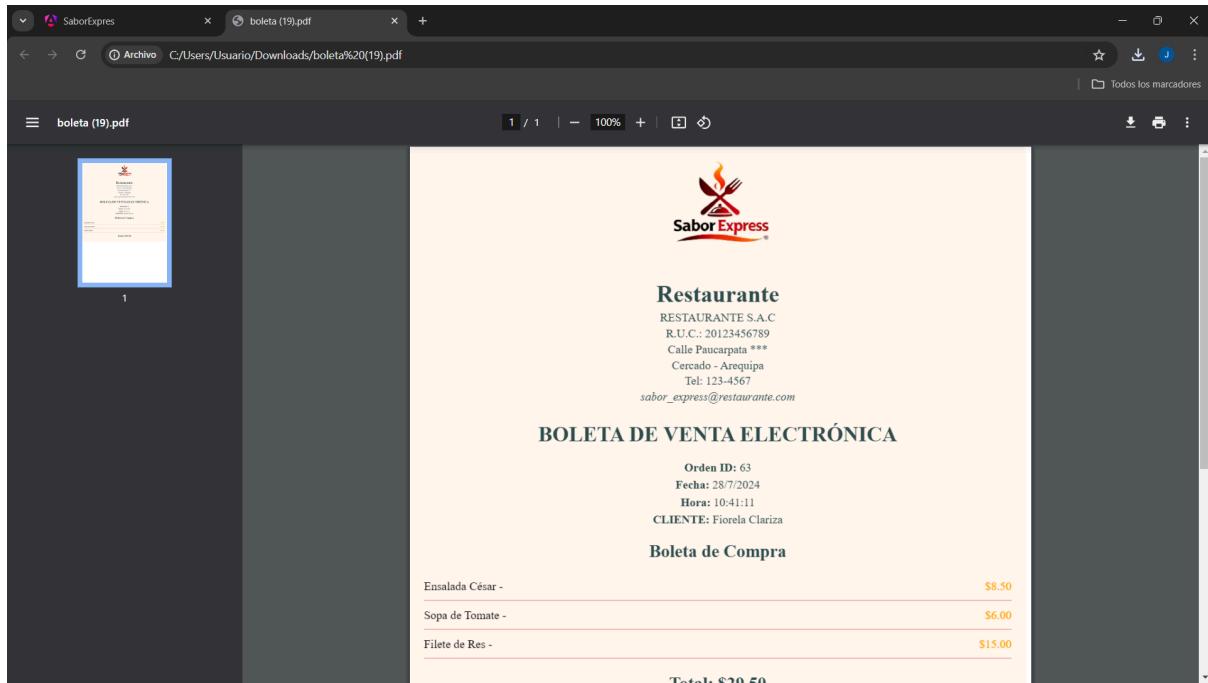
Descripción: Muestra la boleta de venta electrónica, que se puede imprimir en formato PDF o enviar por email al cliente.

Proceso:

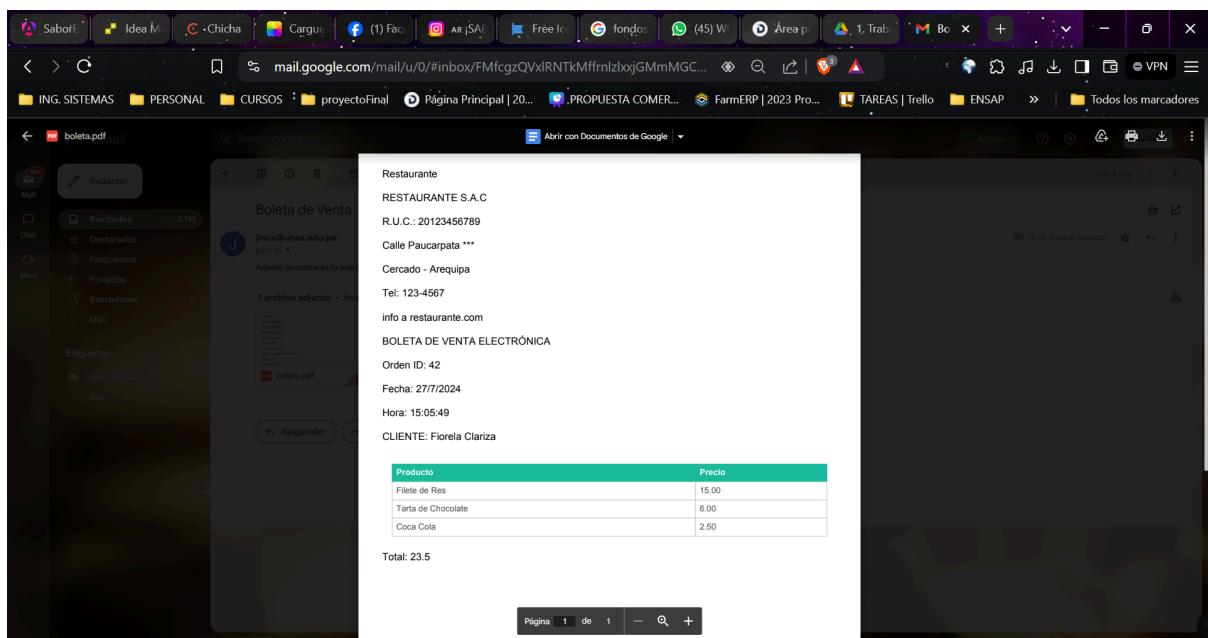
1. El cliente visualiza la boleta de venta.
2. Puede imprimir la boleta en formato PDF.
3. Puede enviar la boleta por email.

The screenshot shows a web browser window for 'SaborExpress' at 'localhost:4200/confirmacion-orden'. The header includes 'Menu', 'Carrito', 'Reserva', 'Contactanos', the restaurant logo, and user information 'user Logout Fiorela Clariza'. The main content features the restaurant logo and title 'Restaurante'. It provides contact details: RESTAURANTE S.A.C, R.U.C.: 20123456789, Calle Paucarpata ***, Cercado - Arequipa, Tel: 123-4567, and email 'sabor_express@restaurante.com'. The title 'BOLETA DE VENTA ELECTRÓNICA' is centered. Below it, the receipt summary includes 'Orden ID: 63', 'Fecha: 28/7/2024', 'Hora: 10:41:11', and 'CLIENTE: Fiorela Clariza'. The section 'Boleta de Compra' lists the purchase: 'Ensalada César - \$8.50'.

8. Impresión de boleta



9. Envío de boleta por email



10. Información para contacto

Descripción: Proporciona información de contacto para que los clientes puedan saber donde se encuentra el restaurante, comunicarse si tienen alguna pregunta o necesitan más información.

The screenshot shows a web browser window with the URL localhost:4200/contactanos. The page has a pink header bar with the SaborExpress logo and navigation links: Menu, Carrito, Reserva, Contactanos, user Logout, and a button for 'Fiorela Clariza'. Below the header is a red-bordered section titled 'Contáctanos' containing a message: 'Si tienes alguna pregunta o necesitas más información, no dudes en contactarnos.' Underneath is a box titled 'Información del Restaurante' listing details: Nombre: Restaurante S.A.C., Dirección: Calle Pucarpata ***, Ciudad: Cercado - Arequipa, Teléfono: 123-4567, Email: info@restaurante.com. To the right is a map showing the restaurant's location in Arequipa.

11. Inicio de sesión para el administrador (usuario)

Descripción: Permite a los administradores iniciar sesión en el sistema para acceder al panel de administración.

The screenshot shows a web browser window with the URL localhost:4200/user-login. The page has a pink header bar with the SaborExpress logo and navigation links: Menu, Carrito, Reserva, Contactanos, user Login, and Register. Below the header is a dark grey login form titled 'Iniciar Sesión (Admins)'. It contains two input fields labeled 'Nombre:' and 'Contraseña:', and a 'Login' button at the bottom.

12. Panel de administración

Descripción: Proporciona funcionalidades para que el administrador pueda crear y eliminar productos y categorías, así como ver todas las órdenes hechas. Las órdenes no pagadas se marcan de color naranja y las pagadas de color verde. Además, permite marcar las órdenes como atendidas.

Proceso:

1. El administrador inicia sesión y accede al panel de administración.
2. Puede crear y eliminar productos y categorías.
3. Visualiza todas las órdenes y su estado (pagadas o no pagadas).
4. Puede marcar las órdenes como atendidas.

The screenshot shows the Sabor Express dashboard interface. At the top, there's a navigation bar with links for 'Menu', 'Carrito', 'Nosotros', 'Contactanos', 'Admin' (logged in as 'Admin'), 'Iniciar Sesión', and 'Registrarse'. The main area is divided into several sections:

- Crear Producto:** A form with fields for 'Nombre', 'Descripción', 'Precio', 'Categoría', and 'Imagen' (with a file selection input). A red button labeled 'Crear Producto' is at the bottom.
- Categorías:** A list of categories with 'Eliminar' (Delete) buttons: Entradas, Platos Principales, Postres, Bebidas, Helados, and Sopas.
- Órdenes:** A section showing an order for 'Filete de Res - Cantidad: 1 - Precio: \$15.00', 'Tarta de Chocolate - Cantidad: 1 - Precio: \$6.00', and 'Coca Cola - Cantidad: 1 - Precio: \$2.50'. It includes a green button labeled 'Marcar como atendida'.
- Productos:** A list of products with 'Eliminar' buttons: Ensalada César, Sopa de Tomate, and Filete de Res.
- Crear Categoría:** A form with a 'Nombre' field and a red 'Crear Categoría' button.

Estas funcionalidades garantizan que tanto los clientes como los administradores puedan interactuar de manera eficiente con la plataforma, cubriendo todo el ciclo de compra y gestión del restaurante.

Publicación de la aplicación en la web

La aplicación se ha publicado utilizando Amazon S3 para el frontend y EC2 para el backend en la prueba gratuita. Debido a limitaciones, no pudimos adquirir un dominio personalizado y tampoco un certificado SSL por lo que la página es no segura.

Enlace:

<http://restaurantesabor.s3-website.us-east-2.amazonaws.com/>

The screenshot shows a web browser window displaying the menu of a restaurant named 'Sabor Express'. The header features a red navigation bar with links for 'Menu', 'Carrito' (Cart), 'Reserva' (Reserve), 'Contactanos' (Contact), and user authentication ('user', 'Login', 'Register'). The logo 'Sabor Express' is centered above the main content area. A shopping cart icon in the top left corner indicates 0 items. The main section is titled 'Menú de Productos' (Product Menu) and includes a search bar for filtering by category. Three menu items are displayed in cards:

- Ensalada César**: Described as 'Ensalada fresca con pollo, queso y aderezo César'. Price: 8.50. Add to cart button.
- Sopa de Tomate**: Described as 'Sopa cremosa de tomate con albahaca'. Price: 6.00. Add to cart button.
- Filete de Res**: Described as 'Filete de res a la parrilla con guarnición de papas'. Price: 15.00. Add to cart button.

CONCLUSIONES Y RECOMENDACIONES

Este proyecto ha permitido desarrollar una solución integral para la gestión de un restaurante, mejorando la experiencia del cliente y facilitando la administración de pedidos y productos. Se recomienda explorar opciones para adquirir un dominio personalizado y un certificado SSL para mejorar la seguridad y profesionalismo de la aplicación.

REFERENCIAS

- [1] Django Software Foundation. Django. [Online]. Available: <https://www.djangoproject.com/>
- [2] "Model-View-Controller (MVC) Design Pattern." [Online]. Available: <https://developer.mozilla.org/en-US/docs/Glossary/MVC>
- [3] "Angular HttpClient." [Online]. Available: <https://angular.io/guide/http>
- [4] "JavaScript Frameworks." [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- [5] "HTTP Requests." [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP>