

# UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS



## PROGRAMACIÓN WEB 2

### INFORME FINAL DEL PROYECTO

Sistema Web para Gestión de Restaurantes y Pedidos en Línea

#### INTEGRANTES:

Quispe Mamani Jose Gabriel  
Riveros Vilca Alberth Edwar  
Estefanero Palma Rodrigo  
Auccacusi Conde Brayan Carlos

#### FECHA DE ENTREGA:

July 28, 2025

#### CICLO ACADÉMICO:

2025-I

#### PROFESOR:

Corrales Delgado Carlo Jose Luis

**AREQUIPA – PERÚ**

2025

## Índice

<b>1 ProyectoFinal-PW2</b> . . . . .	<b>4</b>
1.1 Introducción . . . . .	4
1.2 Objetivos . . . . .	4
1.3 Descripción general . . . . .	4
1.4 Requerimientos funcionales . . . . .	5
1.5 Estructura Django . . . . .	5
1.6 Estructura Angular . . . . .	5
<b>2 Django</b> . . . . .	<b>6</b>
2.1 Dependencias . . . . .	6
2.2 Configuraciones principales . . . . .	6
Importaciones y rutas . . . . .	6
Rutas de archivos estáticos y media . . . . .	7
Seguridad . . . . .	7
Configuración de JWT (JSON Web Token) . . . . .	7
Aplicaciones instaladas . . . . .	7
Modelo de usuario personalizado . . . . .	8
Middleware . . . . .	8
CORS . . . . .	8
Base de datos . . . . .	8
Internacionalización . . . . .	9
Configuración de DRF y Djoser . . . . .	9
Otros . . . . .	9
2.3 Modelos y apps . . . . .	9
Apps creadas . . . . .	9
Modelo de Usuario Personalizado . . . . .	10
Modelos de la app restaurants . . . . .	10
2.4 Serializers . . . . .	11
Serializers de usuarios . . . . .	11
Serializers de restaurantes y pedidos . . . . .	11
2.5 Views . . . . .	12
2.6 URLs y Admin . . . . .	13
URLs principales . . . . .	13
Admin . . . . .	14
<b>3 Angular</b> . . . . .	<b>15</b>
3.1 Dependencias . . . . .	15
3.2 Registro de usuario . . . . .	15
3.3 Login . . . . .	16
3.4 Menú del Día . . . . .	16
3.5 Pedidos del Día . . . . .	17
3.6 Historial de Pedidos . . . . .	17

3.7	Explorador de Restaurantes . . . . .	18
3.8	Menú del Restaurante . . . . .	18
3.9	Confirmación de Pedido . . . . .	18
3.10	Mis Pedidos . . . . .	19
<b>4</b>	<b>Resultados . . . . .</b>	<b>19</b>
	Página de registro de usuario . . . . .	20
	Vista del menú del día (restaurante) . . . . .	20
	Pedidos del día (restaurante) . . . . .	21
	Explorador de restaurantes (cliente) . . . . .	21
	Confirmación de pedido (cliente) . . . . .	22
4.1	Conclusiones . . . . .	22
4.2	Evaluación del Trabajo en Equipo . . . . .	22
4.3	Anexos . . . . .	23

## 1 ProyectoFinal-PW2

### 1.1 Introducción

En los últimos años, la digitalización de los sistemas de pedidos de comida ha crecido de manera exponencial, impulsada por la necesidad de ofrecer un servicio más eficiente, mejorar la experiencia del usuario y facilitar operaciones remotas. Los métodos tradicionales de gestión de pedidos en restaurantes suelen presentar limitaciones, especialmente en horas de alta demanda, dificultando la organización interna y afectando la satisfacción del cliente.

En este contexto, se desarrolló un sistema web completo de gestión de pedidos para restaurantes, utilizando Angular para el frontend y Django REST Framework para el backend. Este sistema permite a los clientes buscar restaurantes disponibles, visualizar sus menús del día, realizar pedidos y dar seguimiento al estado de los mismos en tiempo real. Por otro lado, los administradores de los restaurantes pueden gestionar sus menús, aceptar o rechazar pedidos, y consultar historiales, todo desde una interfaz personalizada según su rol.

El sistema implementa un mecanismo de autenticación mediante JSON Web Tokens (JWT), lo cual garantiza un acceso seguro y controlado por tipo de usuario. La arquitectura modular y escalable facilita futuras ampliaciones como pasarelas de pago o sistemas de notificaciones.

El objetivo principal del proyecto fue brindar una solución eficiente y moderna para optimizar la interacción entre clientes y restaurantes. Este informe describe la arquitectura del sistema, las funcionalidades desarrolladas y el proceso completo de implementación. Además, se presentan los resultados alcanzados, incluyendo pruebas funcionales exitosas, validación por usuarios y el cumplimiento de hitos técnicos establecidos.

### 1.2 Objetivos

- Integrar los conocimientos adquiridos en el curso de Programación Web 2.
- Fomentar el trabajo en equipo y la colaboración en el desarrollo de software.
- Desarrollar una aplicación web real que resuelva una necesidad concreta de una empresa, aplicando buenas prácticas de desarrollo backend y frontend.

### 1.3 Descripción general

El proyecto se desarrolló para una empresa ficticia del rubro gastronómico que necesitaba digitalizar la gestión de sus restaurantes y pedidos en línea. El problema principal era la falta de un sistema centralizado para que los clientes pudieran explorar restaurantes, ver menús diarios y realizar pedidos, y para que los restaurantes pudieran gestionar sus menús y pedidos de manera eficiente.

La solución propuesta fue una plataforma web compuesta por un backend en Django y un frontend en

Angular, que permite la interacción fluida y segura entre clientes y restaurantes, automatizando los procesos de registro, autenticación, gestión de menús, pedidos y seguimiento.

#### **1.4 Requerimientos funcionales**

- Registro y autenticación de usuarios (clientes y restaurantes).
- Gestión de perfiles de usuario y restaurante.
- Creación y edición de menús diarios por parte de los restaurantes.
- Visualización de menús y exploración de restaurantes por parte de los clientes.
- Realización de pedidos por los clientes y gestión de pedidos por los restaurantes (aceptar/rechazar).
- Historial de pedidos tanto para clientes como para restaurantes.
- Generación e impresión de recibos en PDF.
- Interfaz responsive y amigable, protegida mediante autenticación JWT.

#### **1.5 Estructura Django**

```
backend/  
    settings.py  
    urls.py  
  
users/  
    models.py  
    serializers.py  
    admin.py  
  
restaurants/  
    models.py  
    serializers.py  
    views.py  
    urls.py  
    admin.py
```

#### **1.6 Estructura Angular**

```
src/  
  app/  
  auth/  
  restaurants/  
  menu-dia/
```

pedidos-dia/  
historial/  
clientes/  
explorador-restaurantes/  
menu-restaurante/  
confirmacion-pedido/  
mis-pedidos/  
services/

## 2 Django

### 2.1 Dependencias

Para el correcto funcionamiento del backend, se instalaron las siguientes dependencias principales:

```
1 pip install Pillow
2 pip install django-cors-headers
3 pip install djangorestframework-simplejwt
4 pip install djoser
```

- **Pillow:** Permite el manejo de imágenes en los modelos (por ejemplo, para fotos de platos o restaurantes).
- **django-cors-headers:** Habilita CORS para permitir peticiones desde el frontend Angular.
- **djangorestframework-simplejwt:** Proporciona autenticación basada en JWT para la API.
- **djoser:** Provee endpoints listos para registro, login y gestión de usuarios.

### 2.2 Configuraciones principales

A continuación se resumen las configuraciones clave del archivo `settings.py`:

#### Importaciones y rutas

```
1 from datetime import timedelta
2 from pathlib import Path
3 import os
```

- Se importan módulos para manejar rutas de archivos y definir tiempos de expiración (por ejemplo, para JWT).

## Rutas de archivos estáticos y media

```
1 BASE_DIR = Path(__file__).resolve().parent.parent
2 MEDIA_URL = '/media/'
3 MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
4 STATIC_URL = 'static/'
```

- `BASE_DIR` define la ruta base del proyecto.
- `MEDIA_URL` y `MEDIA_ROOT` configuran dónde se almacenan y cómo se acceden los archivos subidos por los usuarios.
- `STATIC_URL` define la ruta para los archivos estáticos (CSS, JS, imágenes).

## Seguridad

```
1 SECRET_KEY = '...'
2 DEBUG = True
3 ALLOWED_HOSTS = []
```

- `SECRET_KEY` es la clave secreta de Django, usada para seguridad interna.
- `DEBUG` activa el modo de desarrollo (debe ser `False` en producción).
- `ALLOWED_HOSTS` define los dominios permitidos para acceder a la app.

## Configuración de JWT (JSON Web Token)

```
1 SIMPLE_JWT = {
2     'ACCESS_TOKEN_LIFETIME': timedelta(hours=1),
3     'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
4 }
```

- Se usa la librería Simple JWT para autenticación basada en tokens.
- Define que el token de acceso dura 1 hora y el de refresco 1 día.

## Aplicaciones instaladas

```
1 INSTALLED_APPS = [
2     'django.contrib.admin',
3     'django.contrib.auth',
4     'rest_framework',
5     'rest_framework.authtoken',
6     'djoser',
7     'corsheaders',
8     'users',
9     'restaurants',
10 ]
```

- Incluye apps de Django (admin, auth, etc.).
- `rest_framework`: Framework para crear APIs REST.
- `rest_framework.authtoken`: Autenticación por token.
- `djoser`: Endpoints listos para autenticación y gestión de usuarios.
- `corsheaders`: Permite peticiones desde otros dominios (CORS).
- `users` y `restaurants`: Apps propias del proyecto.

## Modelo de usuario personalizado

```
1 AUTH_USER_MODEL = 'users.User'
```

- Se usa un modelo de usuario propio (`User` en la app `users`) para poder agregar campos personalizados (como tipo de usuario).

## Middleware

```
1 MIDDLEWARE = [
2     'corsheaders.middleware.CorsMiddleware',
3     # ...
4 ]
```

- Lista de middlewares que procesan cada petición/respuesta.
- `CorsMiddleware` permite el uso de CORS para el frontend Angular.

## CORS

```
1 CORS_ALLOWED_ORIGINS = [
2     "http://localhost:4200",
3 ]
```

- Permite que el frontend Angular (en el puerto 4200) haga peticiones a la API Django.

## Base de datos

```
1 DATABASES = {
2     'default': {
3         'ENGINE': 'django.db.backends.sqlite3',
4         'NAME': BASE_DIR / 'db.sqlite3',
5     }
6 }
```

- Usa SQLite como base de datos por defecto (ideal para desarrollo).

## Internacionalización

```
1 LANGUAGE_CODE = 'es-pe'  
2 TIME_ZONE = 'America/Lima'  
3 USE_I18N = True  
4 USE_TZ = True
```

- Configura el idioma y la zona horaria para Perú.

## Configuración de DRF y Djoser

```
1 REST_FRAMEWORK = {  
2     'DEFAULT_AUTHENTICATION_CLASSES': (  
3         'rest_framework_simplejwt.authentication.JWTAuthentication',  
4     ),  
5 }  
6 DJOSER = {  
7     'SERIALIZERS': {  
8         'user_create': 'users.serializers.UserCreateSerializer',  
9         'user': 'users.serializers.UserSerializer',  
10        'current_user': 'users.serializers.UserSerializer',  
11    }  
12 }
```

- REST\_FRAMEWORK: Usa JWT como método de autenticación por defecto.
- DJOSER: Define los serializadores personalizados para la gestión de usuarios.

## Otros

```
1 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

- Define el tipo de campo auto-incremental por defecto para los modelos.

## 2.3 Modelos y apps

### Apps creadas

- **users:**

Gestiona los usuarios del sistema, permitiendo distinguir entre clientes y restaurantes mediante un campo personalizado.

- **restaurants:**

Gestiona toda la lógica relacionada a restaurantes, menús, platos y pedidos.

## Modelo de Usuario Personalizado

```
1 class User(AbstractUser):
2     TIPO_USUARIO = (
3         ('cliente', 'Cliente'),
4         ('restaurante', 'Restaurante'),
5     )
6     tipo = models.CharField(max_length=15, choices=TIPO_USUARIO, default='cliente')
```

- Hereda de `AbstractUser` para aprovechar la autenticación de Django.
- Agrega el campo `tipo` para diferenciar entre usuarios cliente y restaurante.

## Modelos de la app restaurants

### Restaurant

- Código:

```
1 class Restaurant(models.Model):
2     user = models.OneToOneField(User, on_delete=models.CASCADE,
3         limit_choices_to={'tipo': 'restaurante'})
4     nombre = models.CharField(max_length=100)
5     imagen = models.ImageField(upload_to='restaurantes/', blank=True, null=True)
```

- Relaciona un usuario tipo restaurante con su información específica (nombre, imagen).

### Menu

- Código:

```
1 class Menu(models.Model):
2     restaurante = models.ForeignKey(Restaurant, on_delete=models.CASCADE)
3     fecha = models.DateField()
```

- Representa el menú del día de un restaurante.

## Entrada y Segundo

- Código:

```
1 class Entrada(models.Model):
2     nombre = models.CharField(max_length=100)
3     cantidad = models.PositiveIntegerField(default=0)
4     precio = models.DecimalField(max_digits=6, decimal_places=2)
5     imagen = models.ImageField(upload_to='entradas/', blank=True, null=True)
6     menu = models.ForeignKey('Menu', related_name='entradas', on_delete=models.CASCADE)
```

- **Entrada** y **Segundo** representan los platos disponibles en el menú del día, cada uno con nombre, cantidad, precio, imagen y relación al menú.

## Pedido

- Código:

```
1 class Pedido(models.Model):  
2     cliente = models.ForeignKey(User, on_delete=models.CASCADE,  
3         limit_choices_to={'tipo': 'cliente'})  
4     menu = models.ForeignKey(Menu, on_delete=models.CASCADE)  
5     entrada = models.ForeignKey(Entrada, on_delete=models.CASCADE)  
6     segundo = models.ForeignKey(Segundo, on_delete=models.CASCADE)  
7     estado = models.CharField(max_length=20, choices=ESTADOS, default='pendiente')  
8     fecha = models.DateTimeField(auto_now_add=True)
```

- Representa un pedido realizado por un cliente, con referencias al menú, entrada y segundo elegidos, estado del pedido y fecha.

## 2.4 Serializers

Los serializers permiten convertir los modelos de la base de datos a formatos como JSON para enviar datos al frontend, y también validar y transformar datos recibidos desde el frontend antes de guardarlos en la base de datos.

### Serializers de usuarios

- **UserCreateSerializer**

Extiende el serializer de Djoser para permitir el registro de usuarios tipo restaurante con campos extra (`nombre_restaurante`, `imagen_restaurante`).

Si el usuario es restaurante, crea automáticamente un objeto `Restaurant` relacionado.

- **UserSerializer**

Serializa los datos básicos del usuario (`id`, `username`, `tipo`).

### Serializers de restaurantes y pedidos

- **RestaurantListSerializer / RestaurantDetailSerializer**

Permiten listar y mostrar detalles de restaurantes, incluyendo la imagen.

- **MenuCreateSerializer, EntradaCreateSerializer, SegundoCreateSerializer**

Permiten crear menús, entradas y segundos (platos) con sus campos principales.

- **MenuReadSerializer**

Permite mostrar un menú junto con sus entradas y segundos relacionados.

- **PedidoCreateSerializer**

Permite crear pedidos, validando que los platos seleccionados pertenezcan al menú elegido.

- **PedidoReadSerializer**

Permite mostrar los pedidos con detalles de los platos (entrada y segundo).

- **PedidoEstadoUpdateSerializer**

Permite actualizar solo el estado de un pedido (aceptado, rechazado, etc.).

- **RestaurantUpdateSerializer**

Permite editar los datos de un restaurante.

- **PedidoSerializer**

Serializador general para mostrar todos los campos de un pedido.

**Resumen:**

Los serializers permiten controlar cómo se envían y reciben los datos entre el frontend y el backend, asegurando que la información sea válida y esté bien estructurada para cada operación del sistema (registro, menús, pedidos, etc.).

## 2.5 Views

En la app `users` no se implementan vistas personalizadas, ya que la autenticación y gestión de usuarios se maneja con Djoser y Django REST Framework, que proveen endpoints listos para registro, login, recuperación de contraseña y obtención del usuario actual.

Los serializers personalizados permiten adaptar el registro para usuarios tipo restaurante sin necesidad de crear views propias.

En la app `restaurants` sí se crean vistas personalizadas porque la lógica de negocio es más específica. Se utilizan principalmente **ViewSets** y **Generic Views** para:

- CRUD de restaurantes, menús, entradas y segundos (platos)
- Obtener menús por restaurante y por fecha
- Crear y listar pedidos, actualizar su estado
- Listar historial de pedidos y pedidos pendientes
- Actualizar perfil de restaurante

**Ejemplo de ViewSet para restaurantes:**

```
1 class RestaurantViewSet(viewsets.ModelViewSet):  
2     queryset = Restaurant.objects.all()  
3     serializer_class = RestaurantListSerializer  
4     permission_classes = [permissions.IsAuthenticatedOrReadOnly]
```

```

5     filter_backends = [filters.SearchFilter]
6     search_fields = ['nombre']

```

### Ejemplo de vista personalizada para obtener el menú del día:

```

1 class MenuHoyByRestaurantView(generics.RetrieveAPIView):
2     serializer_class = MenuReadSerializer
3     permission_classes = [permissions.AllowAny]
4
5     def get_object(self):
6         restaurante_id = self.kwargs['restaurante_id']
7         hoy = date.today()
8         try:
9             return Menu.objects.get(restaurante__id=restaurante_id, fecha=hoy)
10        except Menu.DoesNotExist:
11            restaurante = Restaurant.objects.get(id=restaurante_id)
12            menu = Menu.objects.create(restaurante=restaurante, fecha=hoy)
13            return menu

```

### Resumen:

- `users`: No tiene views propias porque Djoser y DRF ya proveen todos los endpoints necesarios para usuarios. - `restaurants`: Tiene views personalizadas para toda la lógica de restaurantes, menús, platos y pedidos, usando ViewSets y Generic Views para facilitar el desarrollo y mantener el código organizado.

## 2.6 URLs y Admin

### URLs principales

Archivo principal de URLs (`backend/urls.py`):

```

1 from django.conf import settings
2 from django.conf.urls.static import static
3 from django.contrib import admin
4 from django.urls import path, include
5
6 urlpatterns = [
7     path('admin/', admin.site.urls),
8     path('api/restaurants/', include('restaurants.urls')),
9     path('auth/', include('djoser.urls')),
10    path('auth/', include('djoser.urls.jwt')),
11 ]
12
13 if settings.DEBUG:
14     urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

- `admin/`: Acceso al panel de administración de Django.
- `api/restaurants/`: Todas las rutas de la app `restaurants` (menús, platos, pedidos, etc.).
- `auth/`: Endpoints de autenticación y gestión de usuarios proporcionados por Djoser.

URLs de la app `restaurants` (`restaurants/urls.py`):

```

1  from django.urls import path, include
2  from rest_framework.routers import DefaultRouter
3  from .views import (
4      RestaurantViewSet, MenuViewSet, EntradaViewSet, SegundoViewSet,
5      MenusByRestaurantView, PedidoCreateView, PedidosRecibidosView,
6      PedidoEstadoUpdateView, RestaurantUpdateView, MisPedidosView,
7      RestaurantDetailView, RestaurantByUserView, MenuHoyByRestaurantView,
8      PedidoDetailView, PedidosPendientesRestauranteView,
9      PedidosHistorialRestauranteView
10 )
11 router = DefaultRouter()
12 router.register(r'restaurantes', RestaurantViewSet)
13 router.register(r'menus', MenuViewSet)
14 router.register(r'entradas', EntradaViewSet)
15 router.register(r'segundos', SegundoViewSet)
16
17 urlpatterns = [
18     path('', include(router.urls)),
19     path('restaurantes/id/<int:user_id>', RestaurantByUserView.as_view(), name='restaurant-by-user'),
20     path('restaurantes/<int:pk>', RestaurantDetailView.as_view(), name='restaurant-detail'),
21     path('restaurante/<int:restaurante_id>/menus/', MenusByRestaurantView.
22         as_view(), name='menus-by-restaurant'),
23     path('restaurante/<int:restaurante_id>/menu-hoy/', MenuHoyByRestaurantView.
24         as_view(), name='menu-hoy-by-restaurant'),
25     path('pedidos/', PedidoCreateView.as_view(), name='pedido-create'),
26     path('pedidos-recibidos/', PedidosRecibidosView.as_view(), name='pedidos-
27         recibidos'),
28     path('pedidos/<int:pk>/estado/', PedidoEstadoUpdateView.as_view(), name='
29         pedido-estado-update'),
30     path('mi-restaurante/editar/', RestaurantUpdateView.as_view(), name='
31         restaurant-update'),
32     path('mis-pedidos/', MisPedidosView.as_view(), name='mis-pedidos'),
33     path('pedidos/<int:pk>', PedidoDetailView.as_view(), name='pedido-detail')
34     ,
35     path('restaurante/<int:restaurante_id>/pedidos-pendientes/',
36         PedidosPendientesRestauranteView.as_view(), name='pedidos-pendientes-
37             restaurante'),
38     path('restaurante/<int:restaurante_id>/historial-pedidos/',
39         PedidosHistorialRestauranteView.as_view(), name='historial-pedidos-
40             restaurante'),
41 ]

```

- Usa un router para CRUD automático de restaurantes, menús, entradas y segundos.
- Define rutas personalizadas para obtener menús por restaurante, menú del día, crear y listar pedidos, actualizar estado, historial, etc.

## Admin

Admin de la app `restaurants` (`restaurants/admin.py`):

```
1 from django.contrib import admin
2 from .models import Restaurant, Menu, Entrada, Segundo, Pedido
3
4 admin.site.register(Restaurant)
5 admin.site.register(Menu)
6 admin.site.register(Entrada)
7 admin.site.register(Segundo)
8 admin.site.register(Pedido)
```

- Permite administrar desde el panel de Django todos los modelos relacionados a restaurantes, menús, platos y pedidos.

Admin de la app `users` (`users/admin.py`):

```
1 from django.contrib import admin
2 from .models import User
3
4 admin.site.register(User)
```

- Permite administrar los usuarios personalizados desde el panel de Django.

#### **Resumen:**

Las URLs principales conectan el admin, la API de restaurantes y la autenticación de usuarios. El admin de Django permite gestionar todos los modelos del sistema de forma visual y sencilla.

## **3 Angular**

### **3.1 Dependencias**

Para el frontend se instalaron dependencias como:

```
1 npm install jspdf jspdf-invoice-template
```

Estas librerías permiten generar e imprimir recibos en PDF desde la aplicación.

### **3.2 Registro de usuario**

El formulario permite registrar usuarios como cliente o restaurante. Si es restaurante, se solicitan datos adicionales como nombre e imagen. Los datos se envían al backend mediante un servicio Angular que comunica con el endpoint `/auth/users/` de Django.

#### **Ejemplo de envío de datos desde el componente:**

```
1 onSubmit() {
2   const formData = new FormData();
3   formData.append('username', this.username);
```

```

4  formData.append('password', this.password);
5  formData.append('tipo', this.tipo);
6  if (this.tipo === 'restaurante') {
7    formData.append('nombre_restaurante', this.nombreRestaurante);
8    if (this.imagenRestaurante) {
9      formData.append('imagen_restaurante', this.imagenRestaurante);
10   }
11 }
12 this.auth.register(formData).subscribe({
13   next: () => { this.success = 'Registro exitoso.'; },
14   error: () => { this.error = 'Error en el registro.'; }
15 });
16 }

```

El backend valida y crea el usuario, diferenciando entre cliente y restaurante.

### 3.3 Login

El usuario ingresa su nombre y contraseña. Si las credenciales son correctas, el backend responde con un token JWT que se guarda en el navegador. El token permite acceder a rutas protegidas y obtener los datos del usuario. Según el tipo de usuario, se redirige a la vista correspondiente.

#### Ejemplo de login y redirección:

```

1  onSubmit() {
2    this.auth.login({ username: this.username, password: this.password }).subscribe({
3      next: (res) => {
4        this.auth.saveToken(res.access);
5        this.auth.getCurrentUser().subscribe({
6          next: (user) => {
7            if (user.tipo === 'restaurante') {
8              this.router.navigate(['/restaurants/menu-dia']);
9            } else {
10              this.router.navigate(['/clientes/explorador-restaurantes']);
11            }
12          }
13        });
14      },
15      error: () => {
16        this.error = 'Usuario o contraseña incorrectos';
17      }
18    });
19  }

```

### 3.4 Menú del Día

Permite al restaurante gestionar el menú diario: agregar, editar o eliminar entradas y segundos. Los cambios se envían al backend y se reflejan en la base de datos. Solo usuarios tipo restaurante pueden acceder a esta sección.

**Ejemplo de agregar una entrada:**

```
1 agregarEntrada() {  
2   const formData = new FormData();  
3   formData.append('nombre', this.nuevaEntrada.nombre);  
4   formData.append('precio', this.nuevaEntrada.precio);  
5   formData.append('cantidad', this.nuevaEntrada.cantidad);  
6   formData.append('menu', this.menuId);  
7   if (this.nuevaEntrada.imagen) {  
8     formData.append('imagen', this.nuevaEntrada.imagen);  
9   }  
10  this.restaurantService.addEntrada(formData).subscribe(() => this.  
11    cargarEntradas());  
12 }
```

**3.5 Pedidos del Día**

El restaurante visualiza los pedidos pendientes del día, con información del cliente y los platos solicitados. Puede aceptar o rechazar pedidos, y el estado se actualiza tanto en el backend como en la interfaz.

**Ejemplo de aceptar/rechazar pedido:**

```
1 aceptarPedido(pedidoId: number) {  
2   this.restaurantService.updateEstadoPedido(pedidoId, 'aceptado').subscribe()  
3     => {  
4       this_pedidos = this_pedidos.filter(p => p.id !== pedidoId);  
5     };  
6   rechazarPedido(pedidoId: number) {  
7     this.restaurantService.updateEstadoPedido(pedidoId, 'rechazado').subscribe()  
8       => {  
9         this_pedidos = this_pedidos.filter(p => p.id !== pedidoId);  
10      };  
11  }
```

**3.6 Historial de Pedidos**

Muestra al restaurante el historial de todos los pedidos (aceptados, rechazados, entregados, etc.). Permite filtrar por fecha o estado y generar recibos en PDF para cada pedido.

**Ejemplo de impresión de recibo:**

```
1 imprimirRecibo(pedido: any) {  
2   // Prepara los datos y llama a jsPDFInvoiceTemplate(props)  
3 }
```

### 3.7 Explorador de Restaurantes

Permite a los clientes buscar y explorar restaurantes disponibles. Se puede filtrar por nombre y seleccionar un restaurante para ver su menú del día.

#### Ejemplo de filtrado:

```
1 filtrarRestaurantes() {
2   const termino = this.busqueda.trim().toLowerCase();
3   this.restaurantesFiltrados = this.restaurantes.filter(r =>
4     r.nombre.toLowerCase().includes(termino)
5   );
6 }
```

### 3.8 Menú del Restaurante

El cliente visualiza el menú del día de un restaurante seleccionado, elige una entrada y un segundo, y realiza el pedido. El total se calcula automáticamente y el pedido se envía al backend.

#### Ejemplo de realizar pedido:

```
1 realizarPedido() {
2   if (!this.entradaSeleccionadaId || !this.segundoSeleccionadoId) {
3     alert('Debes seleccionar una entrada y un segundo.');
4     return;
5   }
6   const data = {
7     menu: this.menuDelDia.id,
8     entrada: this.entradaSeleccionadaId,
9     segundo: this.segundoSeleccionadoId
10  };
11  this.clienteService.crearPedido(data).subscribe({
12    next: (pedido) => {
13      this.router.navigate(['/clientes/confirmacion-pedido', pedido.id]);
14    },
15    error: () => {
16      alert('Error al realizar el pedido.');
17    }
18  });
19 }
```

### 3.9 Confirmación de Pedido

Después de realizar un pedido, el cliente ve el estado del mismo (pendiente, aceptado o rechazado). La aplicación consulta periódicamente el backend hasta que el pedido cambia de estado y luego redirige automáticamente.

#### Ejemplo de polling para estado del pedido:

```
1  ngOnInit() {
2      this.pedidoId = Number(this.route.snapshot.paramMap.get('id'));
3      this.verificarEstadoPedido();
4      this.pollingInterval = setInterval(() => this.verificarEstadoPedido(), 2000);
5  }
6  verificarEstadoPedido() {
7      this.clienteService.getPedidoDetalle(this.pedidoId).subscribe({
8          next: pedido => {
9              if (pedido.estado === 'aceptado' || pedido.estado === 'rechazado') {
10                  clearInterval(this.pollingInterval);
11                  setTimeout(() => this.router.navigate(['/clientes/mis-pedidos']), 5000)
12                      ;
13              }
14          });
15 }
```

### 3.10 Mis Pedidos

El cliente puede ver el historial y estado de todos sus pedidos realizados, con detalles como fecha, restaurante, platos y total. Permite hacer seguimiento y navegar fácilmente dentro de la app.

#### Ejemplo de carga de pedidos:

```
1  cargarMisPedidos(userId: number) {
2      this.clienteService.getPedidosPorCliente(userId).subscribe({
3          next: pedidos => {
4              this_pedidos = pedidos;
5          },
6          error: () => {
7              this_pedidos = [];
8          }
9      });
10 }
```

## 4 Resultados

A continuación se muestran algunos resultados visuales del funcionamiento de la plataforma:

## Página de registro de usuario

Formulario para registrar clientes

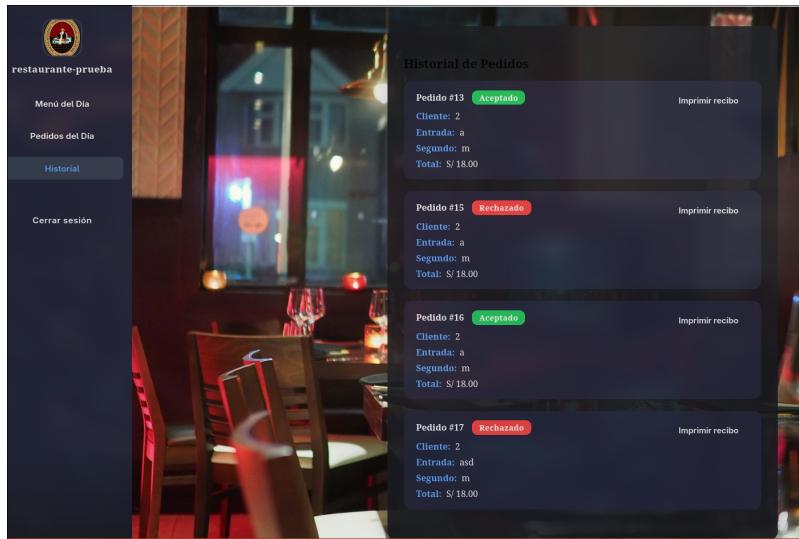
o restaurantes, incluyendo campos adicionales para restaurantes.

## Vista del menú del día (restaurante)

Gestión de entradas y segundos por

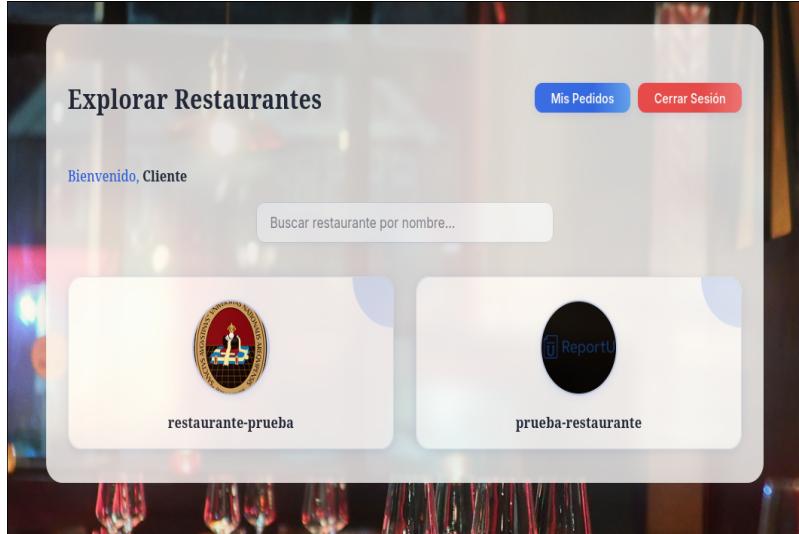
parte del restaurante.

### Pedidos del día (restaurante)



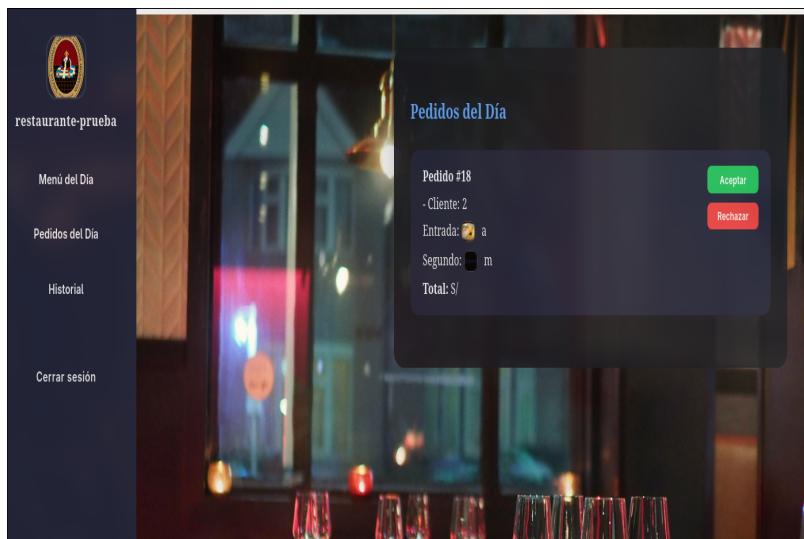
Listado de pedidos pendientes, con opciones para aceptar o rechazar.

### Explorador de restaurantes (cliente)



Panel donde el cliente puede buscar y seleccionar restaurantes.

### Confirmación de pedido (cliente)



*Estado del pedido tras ser realizado, con actualización automática.*

#### 4.1 Conclusiones

El desarrollo de este proyecto permitió aplicar de manera práctica los conocimientos adquiridos en el curso, integrando tecnologías modernas para construir una solución web robusta y escalable. El trabajo en equipo y la colaboración fueron fundamentales para lograr los objetivos propuestos.

#### 4.2 Evaluación del Trabajo en Equipo

Integrante	Responsabilidad (/5)	Proactividad (/5)	Aporte al grupo (/5)	Calificó a sus compañeros	Nota Individual (40%)
Quispe Mamani Jose	3	3	2	Sí	13
Gabriel Riveros Vilca	3	3	3	Sí	14
Alberth Edwar Estefanero Palma	5	5	5	Sí	20
Rodrigo					

Integrante	Responsabilidad (/5)	Proactividad (/5)	Aporte al grupo (/5)	Calificó a sus compañeros	Nota Individual (40%)
Auccacusi	5	5	5	Sí	20
Conde Brayan					
Carlos					

#### 4.3 Anexos

- Repositorio Backend y Frontend (Django y Angular)
- Demo en la nube
- Videos de funcionamiento