
# ComidaExpress

## Segundo Parcial – Implementación e Integración Tecnológica

Materia: Análisis y Metodologías de Sistemas
Profesor: Téc. Contrera Florencio Elías

Estudiante:
Anastasiia Suslova

Año 2025

## Introducción

Este documento presenta la documentación actualizada del sistema ComidaExpress, desarrollado como parte del Segundo Parcial de la materia Análisis y Metodologías de Sistemas. La presente versión integra la nueva implementación real del sistema utilizando Python, Flask, autenticación con Google OAuth y un modelo funcional basado en sesiones y persistencia mediante base de datos SQLite.

El objetivo del proyecto es demostrar la correcta aplicación de conceptos de modularización, gestión de rutas, uso de Blueprints, integración tecnológica externa, manejo seguro de usuarios, así como buenas prácticas de diseño y desarrollo de software.

## 1. Modelo Ambiental

El sistema fue desarrollado utilizando Python 3.12 y Flask 3.1 como framework principal. La arquitectura del proyecto sigue el patrón de aplicación modular mediante la estructura App Factory, permitiendo una alta escalabilidad y mantenimiento del código.

*Herramientas utilizadas:*

• Visual Studio Code

• Python 3.12

• Flask 3.1

• Flask-Login para manejo de sesiones

• SQLite como base de datos local

• Google OAuth 2.0 como autenticación externa

• Render.com para deploy

• GitHub

*Tecnologías:*

• Frontend: HTML5, CSS3, Javascript

• Backend: Flask, Jinja2

• Base de datos: SQLite

• Integración externa: Google OAuth 2.0

## 2. Librerías y Dependencias

Las dependencias clave del sistema incluyen:

• Flask – Framework principal

• Flask-SQLAlchemy – ORM para gestión de la base de datos

• Flask-Login – Autenticación y manejo de sesiones

• Jinja2 – Sistema de templates

• Python-Dotenv – Manejo de variables de entorno

• Google-auth, google-auth-oauthlib – Autenticación con Google

El archivo requirements.txt lista de manera completa todas las dependencias utilizadas para la correcta ejecución del sistema.

### *Estructura del proyecto:*

El proyecto ComidaExpress fue desarrollado siguiendo una arquitectura modular basada en Flask, utilizando el patrón *Application Factory* y la organización mediante *Blueprints*. Esta estructura permite escalar el sistema, separar responsabilidades y mantener un código limpio y fácil de mantener.

La estructura del proyecto es la siguiente:

```
COMIDAEXPRESS/
│
├── app/                          # Código principal del sistema
│   ├── auth/                     # Módulo de autenticación
│   │   ├── routes.py             # Login, Google OAuth 2.0
│   │   └── __init__.py
│   │
│   ├── main/                     # Funcionalidad del sistema
│   │   ├── routes.py             # Inicio, restaurantes, carrito, checkout
│   │   └── __init__.py
│   │
│   ├── models/ (o models.py)     # Modelos ORM: Usuario, Pedido,
Restaurante, Plato, Detalle
│   │
│   ├── templates/                # Vistas HTML renderizadas con Jinja2
│   │   ├── base.html
│   │   ├── index.html
│   │   ├── checkout.html
│   │   ├── restaurantes.html
│   │   ├── login.html
│   │   └── success.html
│   │
│   ├── static/                   # Archivos estáticos
```

```
│   │   ├── css/
│   │   ├── img/
│   │   └── js/ (si aplica)
│   │
│   ├── config.py                  # Configuraciones generales
│   ├── extensions.py              # Inicialización de db, login_manager
│   └── __init__.py                # create_app(), registro de blueprints
│
├── instance/                      # Datos que no se versionan
│   └── database.db                # Base de datos SQLite
│
├── run.py                         # Archivo principal de ejecución
├── requirements.txt               # Dependencias del proyecto
├── README.md                      # Guía para desarrolladores
├── .env                           # Variables sensibles (no versionadas)
├── .gitignore                     # Exclusiones de Git
└── restaurantes.json              # Datos iniciales del catálogo
```

**Descripción general**

- **app/** contiene todo el código fuente del sistema.
- **auth/** administra el login y la autenticación con Google OAuth 2.0.
- **main/** contiene la lógica del catálogo, carrito, restaurante y checkout.
- **templates/** almacena las vistas del sistema.
- **static/** incluye imágenes, estilos CSS y recursos estáticos.
- **models/** define las entidades principales del sistema mediante SQLAlchemy.
- **instance/** almacena la base de datos local.
- **run.py** ejecuta la aplicación usando el patrón *App Factory*.

Esta estructura cumple con las recomendaciones del framework Flask y con las buenas prácticas solicitadas para el segundo parcial.

## 3. Instrucciones de Ejecución

Para ejecutar el proyecto de forma local:

1. Crear entorno virtual:

   python -m venv venv

2. Activar entorno:

   Windows: venv\Scripts\activate

   Mac/Linux: source venv/bin/activate

3. Instalar dependencias:

   pip install -r requirements.txt

4. Configurar variables de entorno en archivo .env:

GOOGLE_CLIENT_ID=...

GOOGLE_CLIENT_SECRET=...

FLASK_SECRET_KEY=...

5. Ejecutar aplicación:

python run.py

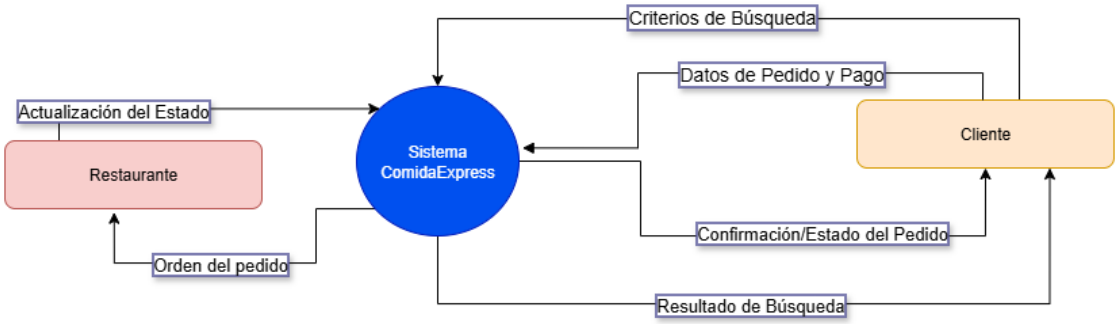
El sistema se ejecutará por defecto en:

*http://127.0.0.1:5000*

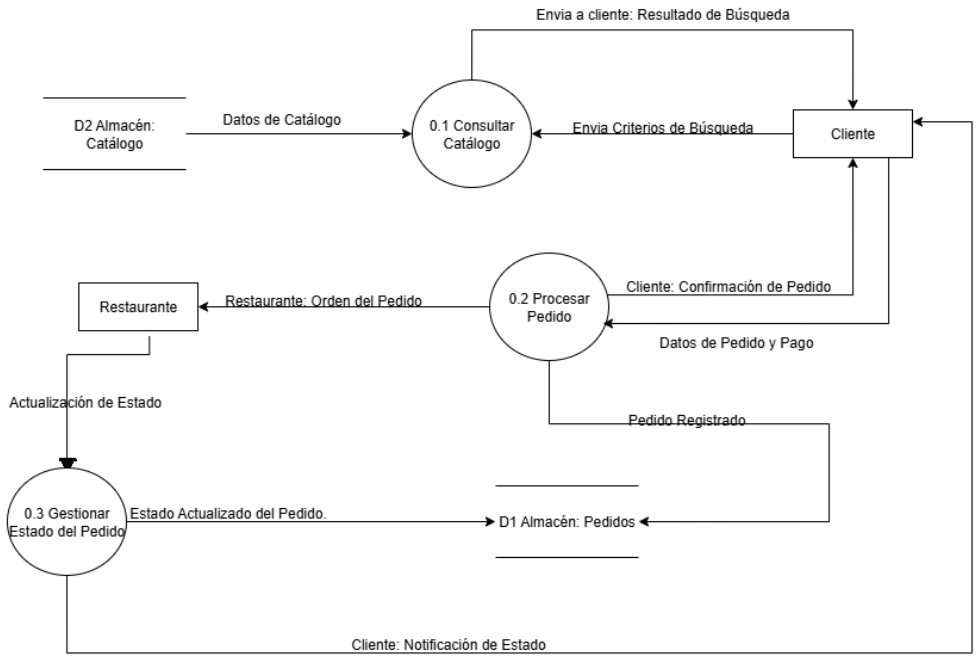
## 4. Diagramas Actualizados

El sistema implementado corresponde directamente a los diagramas elaborados en el Primer Parcial. Las siguientes estructuras se mantienen coherentes:

• Diagrama de Contexto




• Diagrama de Flujo de Datos




• Interacción con actores: Cliente, Administrador, Restaurante

• Comunicación con Google OAuth para autenticación

Estos diagramas representan correctamente el flujo físico del sistema implementado.

## 5. Migración y Cambios Implementados

La migración del proyecto teórico a una versión funcional requirió ajustes:

• Implementación real de rutas y Blueprints

• Implementación de base de datos SQLite

• Creación de modelos ORM para Restaurante, Plato, Usuario, Pedido, Detalles

• Manejo real de sesiones con Flask-Login

• Implementación del sistema de carrito de compras

• Sistema AJAX para actualización en tiempo real sin recargar la página

• Integración con Google OAuth 2.0

• Implementación de un Checkout funcional

• Registro real de pedidos en base de datos

## 6. Resultados y Reflexión

El sistema implementado es totalmente funcional y cumple con los requerimientos del segundo parcial:

• Coherencia entre análisis y desarrollo

• Correcta integración externa (Google OAuth)

• Modularización completa del proyecto

• Uso de sesiones y manejo seguro de usuarios

• Deploy funcional en Render.com

• Documentación clara y profesional

Dificultades encontradas:

• Manejo de sesiones combinando Flask-Login y Google OAuth

• Persistencia del carrito de compras

• Comunicación AJAX sin recarga de página

• Sincronización entre backend y frontend

Mejoras futuras:

• Implementación de pagos reales (Stripe/MercadoPago)

• Panel administrativo avanzado

• Emails transaccionales

• Sistema de tracking en tiempo real para repartidores

• Ampliación de funcionalidades del restaurante (menú dinámico)

## Conclusión

El proyecto ComidaExpress representa una solución completa, coherente y escalable que integra análisis, diseño y desarrollo práctico. Se logró conectar la teoría planteada en el primer parcial con un sistema real que utiliza buenas prácticas de ingeniería de software y tecnologías actuales.