ComidaExpress

Segundo Parcial – Implementación e Integración Tecnológica

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

Estudiante:
Anastasiia Suslova

Año 2025

1. Entorno de Desarrollo

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 basado en la estructura *App Factory*, lo que garantiza una alta escalabilidad y mantenibilidad del código.

Herramientas utilizadas:

- Visual Studio Code
- Python 3.12
- Flask 3.1
- Flask-Login para la gestión de sesiones
- SQLite como base de datos local
- Google OAuth 2.0 para autenticación externa
- Render.com para el despliegue
- GitHub

Tecnologías:

- Frontend: HTML5, CSS3, JavaScript
- Backend: Flask, Jinja2
- Base de datos: SQLite
- Integración externa: Google OAuth 2.0

Estructura del proyecto

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

Estructura de directorios:

```
COMIDAEXPRESS/
│
├── _pycache/
├── .venv/
│
├── app/
│   ├── api/                    # Módulos independientes expuestos como servicios JSON
│   │   ├── __init__.py
│   │   ├── platos_api.py
│   │   ├── restaurantes_api.py
│   │   └── tipos_api.py
│   │
```

```
├── auth/                # Módulo de autenticación
│   ├── routes.py        # Login  Google OAuth 2.0
│   └── __init__.py
│
├── cart/                # Módulo del carrito
│   ├── routes.py        # Agregar, eliminar, actualizar
│   └── __init__.py
│
├── main/                # Lógica principal del sistema
│   ├── routes.py        # Inicio, restaurantes, carrito, checkout
│   └── __init__.py
│
├── models/              # Modelos ORM: Usuario, Pedido, Restaurante,
│   ├── models.py          Plato, Detalles, Tipos
│   └── __init__.py
│
├── orders/              # Módulo de pedidos
│   ├── routes.py        crear pedido, agregar en DB
│   └── __init__.py
│
├── templates/           # Plantillas HTML Jinja2
│   ├── base.html
│   ├── index.html
│   ├── checkout.html
│   ├── restaurantes.html
│   ├── login.html
│   └── success.html
│
├── static/              # Archivos estáticos
│   ├── css/
│   ├── img/
│   └── js/
│
└── __init__.py          #  create_app + registro blueprints
│
├── instance/            # No se incluye en Git
│   └── database.db          # Base de datos SQLite
│
├── run.py               # Archivo principal de ejecución
├── requirements.txt         # Dependencias del proyecto
├── README.md            # Guía para desarrolladores
├── .env                 # Datos sensibles (no incluido en Git)
├── .gitignore           # Exclusiones de Git
├── restaurantes.json        # Datos iniciales del catálogo
├── tipo_comida.json         # Datos iniciales del catálogo
└── platos.json      # Datos iniciales del catálogo
```

2. Bibliotecas y Dependencias

Las dependencias principales del sistema incluyen:

- Flask — framework principal
- Flask-SQLAlchemy — ORM para interacción con la base de datos
- Flask-Login — autenticación y gestión de sesiones
- Jinja2 — motor de plantillas
- python-dotenv — gestión de variables de entorno
- google-auth, google-auth-oauthlib — autenticación mediante Google

El archivo requirements.txt contiene la lista completa de dependencias necesarias para la correcta ejecución del sistema.

3. Instrucciones de Ejecución

Para ejecutar el proyecto localmente:

1. Crear un entorno virtual:

   ```
   python -m venv venv
   ```

2. Activar el entorno:

   ```
   Windows: .venv\Scripts\activate
   ```

3. Instalar dependencias:

   ```
   pip install -r requirements.txt
   ```

4. Ejecutar la aplicación:

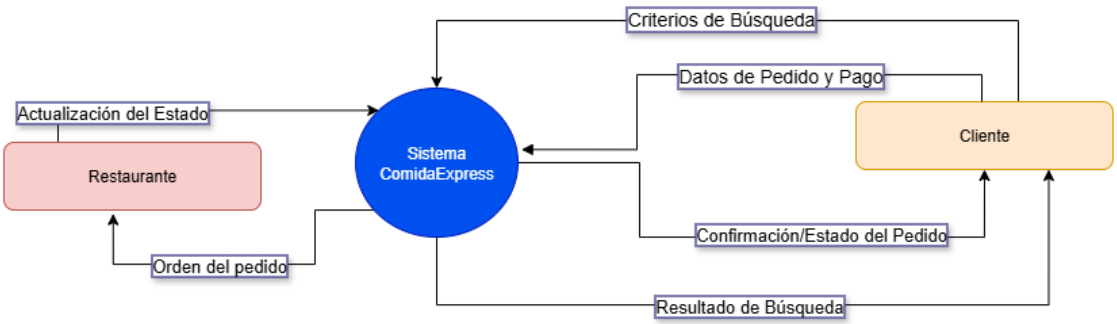
   ```
   python run.py
   ```

El sistema está disponible en:

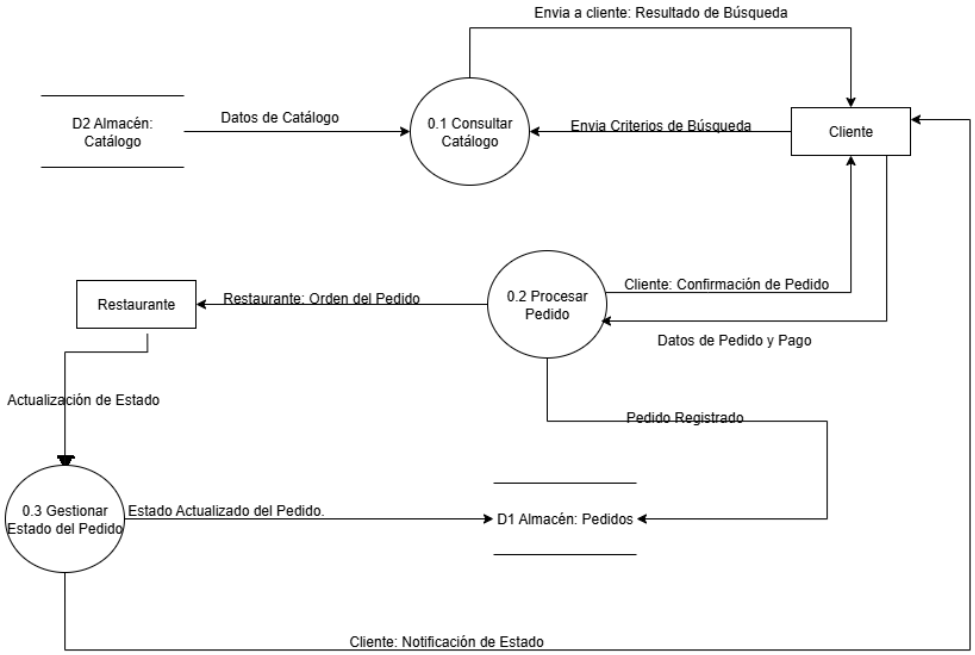
http://127.0.0.1:5000

4. Diagramas Actualizados

- Diagrama de context



- Diagrama de flujo de datos



5. Migración y Cambios Aplicados

La transición del proyecto teórico a una versión funcional requirió:

- Implementación real de rutas y Blueprints
- Integración de la base de datos SQLite
- Creación de modelos ORM: Restaurante, Plato, Usuario, Pedido, Detalles
- Implementación del carrito de compras
- Actualización dinámica mediante AJAX sin recargar la página
- Integración con Google OAuth 2.0
- Registro de pedidos en la base de datos

Dificultades:

- Persistencia del contenido del carrito
- Comunicación AJAX sin recarga
- Despliegue en Render sin datos iniciales en SQLite

Mejoras futuras:

- Integración de pagos reales (Stripe/MercadoPago)
- Desarrollo de panel administrativo
- Envío de correos transaccionales
- Seguimiento de repartidores en tiempo real
- Ampliación del módulo de restaurantes