

**Carrera:**

Ingeniería en sistemas de la información cuarto semestre

**Materia:**

Diseño de sistemas

**Docente:**

Verónica Pailiacho

**Estudiante:**

Jeremy Jácome

**Tema:**

Diagrama de casos de uso

**Fecha:**

30 de junio de 2025

**🧾 INFORME TÉCNICO - SISTEMA DE RUTAS MEDIANTE GRAFOS**

**1. Introducción**

Este proyecto consiste en un sistema web desarrollado con Flask que implementa un **algoritmo de grafos** para encontrar rutas óptimas entre ciudades de Ecuador. El sistema está diseñado para ser utilizado tanto por usuarios regulares como por administradores, permitiendo búsquedas de rutas eficientes y gestión completa de las entidades involucradas.

**2. Objetivos**

* Implementar una red de rutas mediante grafos usando NetworkX.
* Aplicar el algoritmo de **Dijkstra** para encontrar la ruta más económica entre ciudades.
* Visualizar el grafo resultante con matplotlib.
* Brindar autenticación segura y gestión administrativa completa del sistema.

**3. Arquitectura del Sistema**

El proyecto sigue una arquitectura **MVC** (Modelo-Vista-Controlador) modular con separación clara de responsabilidades:

**🧩 Componentes Principales:**

* **Modelos (models/)**: Representan la base de datos (usuarios, provincias, ciudades, rutas).
* **Vistas (templates/)**: Interfaz de usuario HTML con AdminLTE.
* **Controladores (routes/)**: Manejan la lógica HTTP.
* **Servicios (services/)**: Lógica de negocio.
* **Controladores de grafos (controllers/)**: Implementan los algoritmos de grafos.
* **Repositorio (repositories/)**: Acceso a datos desde SQLAlchemy.
* **Configuración (.env, config.py)**: Variables del entorno y configuración de Flask.
* **Entorno Virtual (.venv/)**: Contiene dependencias aisladas para ejecución del sistema.

**4. Descripción del Funcionamiento del Grafo**

**🧠 Lógica del Grafo**

* **Nodos**: Cada ciudad registrada en la base de datos es un nodo.
* **Aristas**: Las rutas entre ciudades son aristas con un peso que representa el costo/distancia.
* **Construcción**: construir\_grafo() transforma los datos relacionales en un grafo de NetworkX.
* **Visualización**: grafo\_a\_imagen() genera una imagen del grafo usando matplotlib con un layout de tipo spring.

[[CAPTURA: Grafo generado]]

**🚀 Algoritmo de Ruta Óptima**

* Se utiliza una **modificación del algoritmo de Dijkstra** implementada en camino\_optimo\_con\_costera().
* Calcula el camino más económico entre dos ciudades.
* También identifica si en la ruta aparecen ciudades costeras (es\_costera=True).

[[CAPTURA: Resultado de ruta óptima con ciudades costeras]]

**5. Flujo del Sistema**

1. **Inicio de Sesión**: El usuario se autentica mediante Flask-Login.
2. **Dashboard**: Accede a /home o al panel de administración si es admin.
3. **Búsqueda de Ruta**:
   * Selecciona ciudad de origen y destino.
   * El sistema construye el grafo.
   * Se ejecuta Dijkstra y se visualiza la ruta.

[[CAPTURA: Formulario de búsqueda de rutas]]

1. **Visualización**: Se muestra la ruta y el costo, y el grafo generado.

**6. Gestión Administrativa**

El administrador accede a un panel completo para realizar operaciones CRUD sobre:

* Usuarios
* Provincias
* Ciudades (incluye atributo costero)
* Rutas (define peso entre ciudades)

[[CAPTURA: Panel de administración]]

**7. Base de Datos**

**Entidades:**

* **usuarios**: Control de acceso y roles.
* **provincias**: División territorial del Ecuador.
* **ciudades**: Nodo del grafo, con provincia y atributo es\_costera.
* **rutas**: Aristas del grafo con campo peso (km).

[[CAPTURA: Diagrama Entidad-Relación]]

**Relaciones:**

* Una provincia tiene muchas ciudades.
* Una ruta conecta dos ciudades.

**8. Tecnologías Utilizadas**

* **Backend**: Flask, SQLAlchemy, Flask-Login, bcrypt
* **Algoritmos**: NetworkX (grafos), matplotlib (visualización)
* **Frontend**: AdminLTE, Bootstrap, jQuery
* **Base de Datos**: MySQL (puerto 3307, charset UTF-8)
* **Entorno**: .venv/ con requirements.txt para instalación reproducible

**9. Seguridad**

* **Autenticación**: Controlada por Flask-Login.
* **Autorización**: Decoradores para rutas de administrador.
* **Encriptación**: Contraseñas protegidas con bcrypt.

**10. Ejecución del Proyecto**

**Requisitos:**

bash

CopyEdit

# Activar entorno virtual

.venv\Scripts\activate

# Instalar dependencias

pip install -r requirements.txt

# Inicializar base de datos

python init\_db.py

# Ejecutar sistema

python app.py

* Acceso: http://localhost:4000
* Admin: usuario: admin, contraseña: admin

**11. Conclusión**

Este proyecto implementa de forma efectiva un sistema de rutas inteligente usando estructuras de grafos. Se logra una separación limpia de capas, seguridad robusta y una experiencia de usuario intuitiva. El uso de algoritmos avanzados como Dijkstra y la generación visual del grafo ofrecen un valor académico y técnico significativo.

**PEQUEÑAS MEJORAS (opcionales):**

**En la sección 4 "Funcionamiento del Grafo":**

Agregar una línea sobre complejidad:

"El algoritmo de Dijkstra tiene una complejidad temporal de O(V²) donde V es el número de vértices (ciudades)."

**En la sección 8 "Tecnologías":**

Especificar por qué elegiste cada tecnología:

"NetworkX por su optimización en algoritmos de grafos, AdminLTE por su interfaz profesional preconfigurada."

**En la sección 10 "Ejecución":**

Mencionar requisitos previos:

"Requisitos previos: Python 3.8+, MySQL/Laragon ejecutándose en puerto 3307."

**📸 CAPTURAS RECOMENDADAS:**

1. **Grafo generado** con las 6 ciudades conectadas
2. **Resultado de búsqueda** mostrando ruta Quito→Loja con costo
3. **Panel de administración** con las tablas CRUD
4. **Formulario de búsqueda** con dropdowns de ciudades
5. **Dashboard principal** después del login
6. **Diagrama ER** de la base de datos (opcional pero impresionante)

**Sistema de Rutas mediante grafos**

Voy a limpiar todos los archivos innecesarios y luego analizar cada archivo del proyecto para tu documentación técnica. Ahora voy a analizar cada archivo para tu documentación técnica:

## 📋 \*\*ANÁLISIS COMPLETO DE ARCHIVOS PARA DOCUMENTACIÓN TÉCNICA\*\*

### \*\*🎯 ARCHIVOS PRINCIPALES (Raíz del Proyecto)\*\*

#### \*\*1. app.py - ARCHIVO PRINCIPAL DE LA APLICACIÓN\*\*

- \*\*Qué hace\*\*: Punto de entrada principal del sistema Flask

- \*\*Para qué\*\*: Configura y ejecuta toda la aplicación web

- \*\*Por qué\*\*: Centraliza la configuración de Flask, base de datos, autenticación y rutas

- \*\*Funciones clave\*\*:

- `create\_app()`: Factory pattern para crear la aplicación

- Configuración de Flask-Login para autenticación

- Inicialización de SQLAlchemy y bcrypt

- Registro de blueprints (rutas modulares)

- Manejo de errores 404/500

- Auto-creación de tablas de base de datos

- Verificación del estado del sistema al inicio

#### \*\*2. requirements.txt - DEPENDENCIAS DEL PROYECTO\*\*

- \*\*Qué hace\*\*: Lista todas las librerías Python necesarias

- \*\*Para qué\*\*: Gestión de dependencias del proyecto

- \*\*Por qué\*\*: Permite instalación automática con `pip install -r requirements.txt`

- \*\*Dependencias clave\*\*: Flask, SQLAlchemy, NetworkX, matplotlib, bcrypt

#### \*\*3. .env - CONFIGURACIÓN DEL ENTORNO\*\*

- \*\*Qué hace\*\*: Variables de configuración sensibles

- \*\*Para qué\*\*: Configuración de base de datos y secretos

- \*\*Por qué\*\*: Separar configuración del código por seguridad

- \*\*Contiene\*\*: Host DB, puerto 3307, credenciales, secret key

#### \*\*4. README.md - DOCUMENTACIÓN BÁSICA\*\*

- \*\*Qué hace\*\*: Documentación básica del proyecto

- \*\*Para qué\*\*: Instrucciones de instalación y uso

- \*\*Por qué\*\*: Primera información que ve cualquier desarrollador

#### \*\*5. DOCUMENTACION\_COMPLETA.md - DOCUMENTACIÓN TÉCNICA\*\*

- \*\*Qué hace\*\*: Documentación técnica detallada

- \*\*Para qué\*\*: Referencia completa de la arquitectura

- \*\*Por qué\*\*: Facilita mantenimiento y comprensión del sistema

---

### \*\*📂 CARPETA models - CAPA DE DATOS\*\*

#### \*\*\_\_init\_\_.py\*\*

- \*\*Qué hace\*\*: Inicializa el paquete models

- \*\*Para qué\*\*: Centraliza importaciones de modelos

- \*\*Por qué\*\*: Facilita imports y organización modular

#### \*\*config.py\*\*

- \*\*Qué hace\*\*: Configuración de la aplicación Flask

- \*\*Para qué\*\*: Centralizar configuración de base de datos

- \*\*Por qué\*\*: Separar configuración de lógica de negocio

- \*\*Configuraciones\*\*: URI MySQL, charset UTF-8, secret key

#### \*\*database.py\*\*

- \*\*Qué hace\*\*: Instancias globales de SQLAlchemy y bcrypt

- \*\*Para qué\*\*: Evitar importaciones circulares

- \*\*Por qué\*\*: Patrón singleton para recursos compartidos

#### \*\*models.py\*\*

- \*\*Qué hace\*\*: Importaciones centralizadas de todos los modelos

- \*\*Para qué\*\*: Facilitar acceso a modelos desde app.py

- \*\*Por qué\*\*: Simplificar importaciones en el archivo principal

### \*\*📊 SUBCARPETA entities - MODELOS DE BASE DE DATOS\*\*

#### \*\*user.py\*\*

- \*\*Qué hace\*\*: Modelo de usuario con autenticación

- \*\*Para qué\*\*: Gestión de usuarios y roles del sistema

- \*\*Por qué\*\*: Autenticación y autorización seguras

- \*\*Funcionalidades\*\*:

- Encriptación de contraseñas con bcrypt

- Integración con Flask-Login

- Campo `es\_admin` para roles

- Métodos de verificación

#### \*\*`entities/provincia.py`\*\*

- \*\*Qué hace\*\*: Modelo de provincias de Ecuador

- \*\*Para qué\*\*: Organización territorial del país

- \*\*Por qué\*\*: Estructura jerárquica provincia → ciudad

- \*\*Relación\*\*: Una provincia tiene muchas ciudades

#### \*\*`entities/ciudad.py`\*\*

- \*\*Qué hace\*\*: Modelo de ciudades con atributos especiales

- \*\*Para qué\*\*: Nodos del grafo de rutas

- \*\*Por qué\*\*: Cada ciudad es un vértice en el algoritmo de grafos

- \*\*Atributos especiales\*\*:

- `es\_costera`: Marcador para ciudades costeras

- Relación con provincia

- Base para origen/destino de rutas

#### \*\*`entities/ruta.py`\*\*

- \*\*Qué hace\*\*: Modelo de conexiones entre ciudades

- \*\*Para qué\*\*: Aristas del grafo con pesos (costos)

- \*\*Por qué\*\*: Implementa las conexiones para algoritmo de Dijkstra

- \*\*Campos clave\*\*:

- `ciudad\_origen\_id`: Ciudad de salida

- `ciudad\_destino\_id`: Ciudad de llegada

- `peso`: Costo/distancia de la ruta

### \*\*🗄️ SUBCARPETA repositories - ACCESO A DATOS\*\*

#### \*\*`repositories/base\_repository.py`\*\*

- \*\*Qué hace\*\*: Clase base con operaciones CRUD genéricas

- \*\*Para qué\*\*: Evitar repetición de código en repositorios

- \*\*Por qué\*\*: Principio DRY (Don't Repeat Yourself)

- \*\*Métodos\*\*: create, get\_by\_id, get\_all, update, delete

#### \*\*`repositories/user\_repository.py`\*\*

- \*\*Qué hace\*\*: Operaciones específicas de usuarios

- \*\*Para qué\*\*: Consultas complejas de autenticación

- \*\*Por qué\*\*: Encapsular lógica de acceso a datos de usuarios

#### \*\*`repositories/ciudad\_repository.py`\*\*

- \*\*Qué hace\*\*: Operaciones específicas de ciudades

- \*\*Para qué\*\*: Consultas para construcción del grafo

- \*\*Por qué\*\*: Optimizar consultas de ciudades para algoritmos

#### \*\*`repositories/provincia\_repository.py`\*\*

- \*\*Qué hace\*\*: Operaciones específicas de provincias

- \*\*Para qué\*\*: Gestión administrativa de provincias

- \*\*Por qué\*\*: Separar responsabilidades por entidad

#### \*\*`repositories/ruta\_repository.py`\*\*

- \*\*Qué hace\*\*: Operaciones específicas de rutas

- \*\*Para qué\*\*: Consultas para construcción del grafo

- \*\*Por qué\*\*: Base fundamental para algoritmos de rutas

### \*\*🔧 SUBCARPETA services - LÓGICA DE NEGOCIO\*\*

#### \*\*`services/user\_service.py`\*\*

- \*\*Qué hace\*\*: Lógica de negocio para usuarios

- \*\*Para qué\*\*: Procesos complejos de autenticación

- \*\*Por qué\*\*: Separar lógica de negocio del acceso a datos

#### \*\*`services/grafo\_service.py`\*\*

- \*\*Qué hace\*\*: Lógica de negocio para grafos y algoritmos

- \*\*Para qué\*\*: Construcción de grafos y cálculos de rutas

- \*\*Por qué\*\*: Encapsular algoritmos complejos

---

### \*\*🌐 CARPETA routes - CONTROLADORES WEB\*\*

#### \*\*\_\_init\_\_.py\*\*

- \*\*Qué hace\*\*: Registra todos los blueprints

- \*\*Para qué\*\*: Modularización de rutas

- \*\*Por qué\*\*: Organización limpia del código

#### \*\*auth\_routes.py\*\*

- \*\*Qué hace\*\*: Maneja autenticación (login/register/logout)

- \*\*Para qué\*\*: Control de acceso al sistema

- \*\*Por qué\*\*: Seguridad y gestión de sesiones

- \*\*Rutas\*\*: `/login`, `/register`, `/logout`

#### \*\*home\_routes.py\*\*

- \*\*Qué hace\*\*: Página principal después del login

- \*\*Para qué\*\*: Dashboard del usuario

- \*\*Por qué\*\*: Punto central de navegación

#### \*\*admin\_routes.py\*\*

- \*\*Qué hace\*\*: Panel de administración completo

- \*\*Para qué\*\*: Gestión CRUD de todas las entidades

- \*\*Por qué\*\*: Interfaz administrativa para el sistema

- \*\*Funcionalidades\*\*: Gestión de usuarios, provincias, ciudades, rutas

#### \*\*rutaeconomica\_routes.py\*\*

- \*\*Qué hace\*\*: Búsqueda de rutas óptimas entre ciudades

- \*\*Para qué\*\*: Funcionalidad principal del sistema

- \*\*Por qué\*\*: Implementa el algoritmo de Dijkstra

- \*\*Características\*\*:

- Algoritmo de camino más corto

- Visualización de grafos

- Resultados con costos

#### \*\*ruta\_fija\_routes.py\*\*

- \*\*Qué hace\*\*: Ruta predefinida específica Ibarra→Loja

- \*\*Para qué\*\*: Demostración de ruta fija del sistema

- \*\*Por qué\*\*: Ejemplo concreto de funcionalidad

---

### \*\*🎮 CARPETA controllers - LÓGICA DE GRAFOS\*\*

#### \*\*grafo\_utils.py - NÚCLEO DEL SISTEMA\*\*

- \*\*Qué hace\*\*: Implementa toda la lógica de grafos y algoritmos

- \*\*Para qué\*\*: Core del sistema de rutas

- \*\*Por qué\*\*: Corazón matemático de la aplicación

- \*\*Funciones principales\*\*:

##### \*\*`construir\_grafo()`\*\*

- \*\*Qué hace\*\*: Construye grafo de NetworkX desde base de datos

- \*\*Para qué\*\*: Convertir datos relacionales en estructura de grafo

- \*\*Por qué\*\*: Base para todos los algoritmos

##### \*\*`grafo\_a\_imagen()`\*\*

- \*\*Qué hace\*\*: Genera visualización PNG del grafo

- \*\*Para qué\*\*: Representación visual de la red de rutas

- \*\*Por qué\*\*: Facilita comprensión del usuario

- \*\*Tecnología\*\*: matplotlib + NetworkX con layout spring

##### \*\*`camino\_optimo\_con\_costera()`\*\*

- \*\*Qué hace\*\*: Implementa algoritmo de Dijkstra modificado

- \*\*Para qué\*\*: Encontrar ruta más económica entre ciudades

- \*\*Por qué\*\*: Algoritmo óptimo para caminos más cortos

- \*\*Características\*\*:

- Algoritmo de Dijkstra clásico

- Detección de ciudades costeras en la ruta

- Retorna camino completo con costo total

##### \*\*`obtener\_ciudades()`\*\*

- \*\*Qué hace\*\*: Lista todas las ciudades disponibles

- \*\*Para qué\*\*: Poblar formularios de selección

- \*\*Por qué\*\*: Interfaz usuario-friendly

##### \*\*`estadisticas\_grafo()`\*\*

- \*\*Qué hace\*\*: Calcula estadísticas del grafo

- \*\*Para qué\*\*: Información analítica del sistema

- \*\*Por qué\*\*: Insights sobre la red de rutas

---

### \*\*🎨 CARPETA templates - INTERFAZ DE USUARIO\*\*

#### \*\*`base.html`\*\*

- \*\*Qué hace\*\*: Plantilla base con navegación y estilos

- \*\*Para qué\*\*: Layout común para todas las páginas

- \*\*Por qué\*\*: Consistencia visual y DRY en frontend

- \*\*Tecnología\*\*: AdminLTE + Bootstrap

#### \*\*`home.html`\*\*

- \*\*Qué hace\*\*: Dashboard principal del usuario

- \*\*Para qué\*\*: Punto de entrada después del login

- \*\*Por qué\*\*: Navegación centralizada a funcionalidades

#### \*\*`formulario\_simple.html`\*\*

- \*\*Qué hace\*\*: Interfaz para búsqueda de rutas económicas

- \*\*Para qué\*\*: Funcionalidad principal del sistema

- \*\*Por qué\*\*: Interfaz amigable para algoritmo complejo

- \*\*Características\*\*:

- Selección de origen y destino

- Visualización de resultados

- Mostrar grafo generado

#### \*\*`ruta\_fija.html`\*\*

- \*\*Qué hace\*\*: Muestra ruta predefinida Ibarra→Loja

- \*\*Para qué\*\*: Demostración específica

- \*\*Por qué\*\*: Ejemplo concreto de uso

#### \*\*`error.html`\*\*

- \*\*Qué hace\*\*: Páginas de error personalizadas

- \*\*Para qué\*\*: Manejo elegante de errores

- \*\*Por qué\*\*: Experiencia de usuario profesional

### \*\*📁 SUBCARPETA auth\*\*

#### \*\*`login.html` / `register.html`\*\*

- \*\*Qué hace\*\*: Formularios de autenticación

- \*\*Para qué\*\*: Control de acceso al sistema

- \*\*Por qué\*\*: Seguridad y gestión de usuarios

### \*\*📁 SUBCARPETA admin\*\*

#### \*\*`dashboard.html`\*\*

- \*\*Qué hace\*\*: Panel de control administrativo

- \*\*Para qué\*\*: Vista general del sistema para admin

- \*\*Por qué\*\*: Gestión centralizada

#### \*\*`ciudades.html`, `provincias.html`, `rutas.html`\*\*

- \*\*Qué hace\*\*: Interfaces CRUD para cada entidad

- \*\*Para qué\*\*: Gestión completa de datos

- \*\*Por qué\*\*: Administración sin necesidad de base de datos directa

---

### \*\*🎨 CARPETA static - RECURSOS ESTÁTICOS\*\*

#### \*\*img\*\*

- \*\*Qué hace\*\*: Imágenes del sistema (logos, iconos)

- \*\*Para qué\*\*: Identidad visual de la aplicación

- \*\*Por qué\*\*: Branding y experiencia visual

#### \*\*plugins\*\*

- \*\*Qué hace\*\*: Librerías frontend (Bootstrap, AdminLTE, jQuery)

- \*\*Para qué\*\*: Funcionalidad y estilos avanzados

- \*\*Por qué\*\*: Interfaz profesional sin desarrollo desde cero

---

### \*\*📄 CARPETA codigos\_sql\*\*

#### \*\*laragon\_database.sql\*\*

- \*\*Qué hace\*\*: Script SQL para inicialización de base de datos

- \*\*Para qué\*\*: Crear estructura inicial y datos de prueba

- \*\*Por qué\*\*: Deployment automático de la base de datos

---

## 🎯 \*\*ARQUITECTURA DEL SISTEMA DE GRAFOS\*\*

### \*\*📊 Cómo Funciona el Grafo:\*\*

1. \*\*Nodos (Vértices)\*\*: Cada ciudad es un nodo

2. \*\*Aristas (Edges)\*\*: Cada ruta es una arista con peso (costo)

3. \*\*Algoritmo\*\*: Dijkstra para camino más corto

4. \*\*Visualización\*\*: NetworkX + matplotlib con layout spring

### \*\*🔄 Lógica de las Rutas:\*\*

1. \*\*Construcción\*\*: `construir\_grafo()` lee BD y crea grafo NetworkX

2. \*\*Cálculo\*\*: `camino\_optimo\_con\_costera()` aplica Dijkstra

3. \*\*Resultado\*\*: Retorna camino completo + costo total + ciudades costeras

4. \*\*Visualización\*\*: `grafo\_a\_imagen()` genera PNG del grafo

Ahora te explico qué hace la carpeta .venv:

## 📂 \*\*CARPETA .venv - ENTORNO VIRTUAL DE PYTHON\*\*

### \*\*🔍 ¿Qué hace la carpeta .venv?\*\*

La carpeta .venv es un \*\*entorno virtual de Python\*\* (virtual environment). Es una carpeta especial que contiene una instalación aislada de Python con sus propias librerías.

### \*\*🎯 ¿Para qué sirve?\*\*

#### \*\*1. AISLAMIENTO DE DEPENDENCIAS\*\*

- \*\*Problema\*\*: Si instalas librerías globalmente en Python, pueden haber conflictos entre proyectos

- \*\*Solución\*\*: Cada proyecto tiene su propio entorno aislado

- \*\*Ejemplo\*\*:

- Proyecto A necesita Flask 2.0

- Proyecto B necesita Flask 1.1

- Sin .venv → Conflicto

- Con .venv → Cada uno en su entorno

#### \*\*2. GESTIÓN LIMPIA DE VERSIONES\*\*

- \*\*Tu proyecto\*\*: Usa versiones específicas definidas en requirements.txt

- \*\*Otros proyectos\*\*: No se afectan por las librerías de este proyecto

- \*\*Sistema\*\*: Python global queda limpio

#### \*\*3. REPRODUCIBILIDAD\*\*

- \*\*En tu máquina\*\*: Instalas librerías en .venv

- \*\*En otra máquina\*\*: Se puede recrear exactamente el mismo entorno

- \*\*Comando\*\*: `pip install -r requirements.txt`

### \*\*🗂️ ¿Qué contiene la carpeta .venv?\*\* ### \*\*📋 Estructura de .venv:\*\*

#### \*\*`Scripts/` (Windows) o `bin/` (Linux/Mac)\*\*

- \*\*Qué contiene\*\*: Ejecutables de Python y activación del entorno

- \*\*Archivos importantes\*\*:

- `python.exe` - Intérprete de Python del entorno

- `pip.exe` - Gestor de paquetes del entorno

- `activate.bat` - Script para activar el entorno

#### \*\*`Lib/` - LIBRERÍAS INSTALADAS\*\*

- \*\*Qué contiene\*\*: Todas las librerías Python del proyecto

- \*\*Subcarpetas\*\*:

- `site-packages/` - Donde se instalan las librerías con pip

- Aquí están Flask, SQLAlchemy, NetworkX, etc.

#### \*\*`Include/`\*\*

- \*\*Qué contiene\*\*: Archivos header de Python

- \*\*Para qué\*\*: Compilación de extensiones C/C++

#### \*\*`pyvenv.cfg`\*\*

- \*\*Qué contiene\*\*: Configuración del entorno virtual

- \*\*Para qué\*\*: Define versión de Python y configuraciones

### \*\*🔧 ¿Por qué es importante para tu proyecto?\*\*

#### \*\*1. PARA DESARROLLO\*\*

```bash

# Activar entorno (Windows)

.venv\Scripts\activate

# Instalar dependencias

pip install -r requirements.txt

# Ahora todas las librerías están aisladas

python app.py

```

#### \*\*2. PARA PRESENTACIÓN\*\*

- \*\*Ventaja\*\*: Demuestras buenas prácticas de desarrollo

- \*\*Profesional\*\*: Los entornos virtuales son estándar en la industria

- \*\*Reproducible\*\*: El profesor puede recrear tu entorno exacto

#### \*\*3. PARA DEPLOYMENT\*\*

- \*\*Documentación\*\*: Explicas cómo configurar el entorno

- \*\*Replicabilidad\*\*: Funciona igual en cualquier máquina

### \*\*⚠️ ¿Debo incluir .venv en la documentación?\*\*

#### \*\*SÍ, porque demuestra:\*\*

- ✅ \*\*Buenas prácticas\*\* de desarrollo Python

- ✅ \*\*Gestión profesional\*\* de dependencias

- ✅ \*\*Conocimiento\*\* de herramientas estándar

- ✅ \*\*Código reproducible\*\* y profesional

#### \*\*En tu documentación técnica, incluye:\*\*

\*\*Para la sección de "Arquitectura del Sistema":\*\*

### \*\*🔧 Gestión de Entorno Virtual (.venv/)\*\*

- \*\*Qué hace\*\*: Aislamiento de dependencias Python específicas del proyecto

- \*\*Para qué\*\*: Evitar conflictos entre librerías de diferentes proyectos

- \*\*Por qué\*\*: Buena práctica estándar en desarrollo Python profesional

- \*\*Beneficios\*\*:

- Reproducibilidad del entorno

- Gestión limpia de versiones

- Aislamiento de dependencias

- Deployment consistente

\*\*Comandos importantes:\*\*

```bash

# Crear entorno virtual

python -m venv .venv

# Activar entorno (Windows)

.venv\Scripts\activate

# Instalar dependencias

pip install -r requirements.txt

# Desactivar entorno

deactivate

```

### \*\*📊 Relación con requirements.txt:\*\*

- .venv → Contiene las librerías instaladas

- requirements.txt → Lista las librerías necesarias

- \*\*Flujo\*\*: requirements.txt define → `pip install` → se guarda en .venv

\*\*¡Es una carpeta muy importante que demuestra profesionalismo en tu proyecto!\*\* 🎯

**# 📋 DOCUMENTACIÓN COMPLETA DEL PROYECTO**

**## Sistema de Rutas de Ecuador - Flask Web Application**

---

**## 🔍 \*\*EXPLICACIÓN DE TABLAS DE BASE DE DATOS\*\***

**### \*\*¿Por qué hay "users" y "usuarios"?\*\***

- **\*\*El sistema USA SOLO la tabla "usuarios"\*\*** (definida en `models/entities/user.py`)

- **\*\*La tabla "users" puede ser residual\*\*** de configuraciones anteriores de MySQL

- **\*\*IMPORTANTE\*\***: El modelo `User` especifica `\_\_tablename\_\_ = 'usuarios'`

- **\*\*Todos los registros se guardan en "usuarios"\*\*** ✅

**\*\*Verificación\*\***: El sistema funciona correctamente con la tabla "usuarios"

---

**## 📁 \*\*ESTRUCTURA DEL PROYECTO\*\***

**### \*\*🎯 ARCHIVOS PRINCIPALES\*\***

**#### \*\*`app.py`\*\* - ARCHIVO PRINCIPAL**

- **\*\*Función\*\***: Punto de entrada de la aplicación Flask

- **\*\*Qué hace\*\***:

  - Configura Flask, base de datos, autenticación

  - Registra blueprints (rutas)

  - Maneja errores 404/500

  - Verifica estado de la base de datos

- **\*\*Puerto\*\***: 4000

- **\*\*Para ejecutar\*\***: `python app.py`

**#### \*\*`forms.py`\*\* - FORMULARIOS WEB**

- **\*\*Función\*\***: Define formularios usando Flask-WTF

- **\*\*Qué contiene\*\***:

  - LoginForm: Formulario de login

  - RegistroForm: Formulario de registro

  - Validaciones de campos

- **\*\*Por qué\*\***: Seguridad y validación de datos

**#### \*\*`requirements.txt`\*\* - DEPENDENCIAS**

- **\*\*Función\*\***: Lista todas las librerías Python necesarias

- **\*\*Para instalar\*\***: `pip install -r requirements.txt`

- **\*\*Contiene\*\***: Flask, SQLAlchemy, NetworkX, matplotlib, etc.

**#### \*\*`init\_db.py`\*\* - INICIALIZACIÓN DE BASE DE DATOS**

- **\*\*Función\*\***: Script para crear datos iniciales

- **\*\*Qué hace\*\***:

  - Crea 6 provincias de Ecuador

  - Crea 6 ciudades principales

  - Crea 16 rutas entre ciudades

  - Crea usuario admin (admin/admin)

- **\*\*Para ejecutar\*\***: `python init\_db.py`

---

**### \*\*📂 CARPETA `models/`\*\* - MODELOS DE BASE DE DATOS**

**#### \*\*`models/\_\_init\_\_.py`\*\***

- **\*\*Función\*\***: Hace que `models` sea un paquete Python

- **\*\*Importa\*\***: Todos los modelos para uso fácil

**#### \*\*`models/config.py`\*\* - CONFIGURACIÓN**

- **\*\*Función\*\***: Configuración de la base de datos

- **\*\*Contiene\*\***:

  - URI de MySQL (puerto 3307)

  - Configuración de charset UTF-8

  - Secret key para sesiones

**#### \*\*`models/database.py`\*\* - INSTANCIAS DE BD**

- **\*\*Función\*\***: Instancias globales de SQLAlchemy y bcrypt

- **\*\*Por qué\*\***: Evita importaciones circulares

**#### \*\*`models/models.py`\*\* - IMPORTACIONES CENTRALES**

- **\*\*Función\*\***: Importa todos los modelos en un solo lugar

- **\*\*Para\*\***: Facilitar imports en otras partes

**### \*\*📊 ENTIDADES (Modelos de BD)\*\***

**#### \*\*`models/entities/user.py`\*\* - MODELO USUARIO**

- **\*\*Tabla\*\***: `usuarios` (NO "users")

- **\*\*Campos\*\***: id, username, email, password\_hash, es\_admin, created\_at

- **\*\*Métodos\*\***:

  - `set\_password()`: Encripta contraseñas

  - `check\_password()`: Verifica contraseñas

  - `is\_admin()`: Verifica si es administrador

- **\*\*Autenticación\*\***: Usa Flask-Login

**#### \*\*`models/entities/provincia.py`\*\* - MODELO PROVINCIA**

- **\*\*Tabla\*\***: `provincias`

- **\*\*Campos\*\***: id, nombre, codigo, created\_at

- **\*\*Relación\*\***: Una provincia tiene muchas ciudades

**#### \*\*`models/entities/ciudad.py`\*\* - MODELO CIUDAD**

- **\*\*Tabla\*\***: `ciudades`

- **\*\*Campos\*\***: id, nombre, provincia\_id, es\_costera, created\_at

- **\*\*Relaciones\*\***:

  - Pertenece a una provincia

  - Puede ser origen/destino de rutas

**#### \*\*`models/entities/ruta.py`\*\* - MODELO RUTA**

- **\*\*Tabla\*\***: `rutas`

- **\*\*Campos\*\***: id, ciudad\_origen\_id, ciudad\_destino\_id, peso, created\_at

- **\*\*Función\*\***: Define conexiones entre ciudades con costos

**### \*\*🗄️ REPOSITORIOS (Acceso a Datos)\*\***

**#### \*\*`repositories/base\_repository.py`\*\***

- **\*\*Función\*\***: Clase base para operaciones CRUD

- **\*\*Métodos\*\***: create, get\_by\_id, get\_all, update, delete

- **\*\*Principio\*\***: DRY (Don't Repeat Yourself)

**#### \*\*`repositories/user\_repository.py`\*\***

- **\*\*Función\*\***: Operaciones específicas de usuarios

- **\*\*Métodos\*\***: get\_by\_username, get\_by\_email

**#### \*\*`repositories/ciudad\_repository.py`\*\***

- **\*\*Función\*\***: Operaciones específicas de ciudades

- **\*\*Métodos\*\***: get\_by\_provincia, get\_costeras

**#### \*\*`repositories/provincia\_repository.py`\*\***

- **\*\*Función\*\***: Operaciones específicas de provincias

**#### \*\*`repositories/ruta\_repository.py`\*\***

- **\*\*Función\*\***: Operaciones específicas de rutas

- **\*\*Métodos\*\***: get\_by\_origen, get\_by\_destino

**### \*\*🔧 SERVICIOS (Lógica de Negocio)\*\***

**#### \*\*`services/user\_service.py`\*\***

- **\*\*Función\*\***: Lógica de negocio para usuarios

- **\*\*Métodos\*\***: register\_user, authenticate\_user

**#### \*\*`services/grafo\_service.py`\*\***

- **\*\*Función\*\***: Lógica de grafos y algoritmos

- **\*\*Métodos\*\***: construir\_grafo, calcular\_ruta\_optima

---

**### \*\*🌐 CARPETA `routes/`\*\* - RUTAS WEB**

**#### \*\*`routes/\_\_init\_\_.py`\*\***

- **\*\*Función\*\***: Registra todos los blueprints

- **\*\*Qué hace\*\***: Centraliza el registro de rutas

**#### \*\*`routes/auth\_routes.py`\*\* - AUTENTICACIÓN**

- **\*\*Rutas\*\***:

  - `/login` - Página de login

  - `/register` - Página de registro

  - `/logout` - Cerrar sesión

- **\*\*Función\*\***: Maneja autenticación de usuarios

**#### \*\*`routes/home\_routes.py`\*\* - PÁGINA PRINCIPAL**

- **\*\*Rutas\*\***:

  - `/home` - Dashboard del usuario

- **\*\*Función\*\***: Página principal después del login

**#### \*\*`routes/admin\_routes.py`\*\* - ADMINISTRACIÓN**

- **\*\*Rutas\*\***:

  - `/admin/dashboard` - Panel de control

  - `/admin/usuarios` - Gestión de usuarios

  - `/admin/provincias` - Gestión de provincias

  - `/admin/ciudades` - Gestión de ciudades

  - `/admin/rutas` - Gestión de rutas

- **\*\*Función\*\***: Panel de administración completo

- **\*\*Protección\*\***: Solo usuarios admin

**#### \*\*`routes/rutaeconomica\_routes.py`\*\* - BÚSQUEDA DE RUTAS**

- **\*\*Rutas\*\***:

  - `/ruta\_economica` - Buscar ruta óptima

  - `/grafo` - Ver grafo de rutas

- **\*\*Función\*\***: Algoritmo de Dijkstra para ruta más económica

**#### \*\*`routes/ruta\_fija\_routes.py`\*\* - RUTA FIJA**

- **\*\*Rutas\*\***:

  - `/ruta\_fija` - Ruta predefinida Ibarra→Loja

  - `/ruta\_fija/grafo` - Grafo de la ruta fija

- **\*\*Función\*\***: Muestra ruta específica con costo

---

**### \*\*🎮 CARPETA `controllers/`\*\* - CONTROLADORES**

**#### \*\*`controllers/grafo\_utils.py`\*\* - UTILIDADES DE GRAFOS**

- **\*\*Funciones principales\*\***:

  - `construir\_grafo()`: Crea grafo de NetworkX desde BD

  - `grafo\_a\_imagen()`: Genera imagen PNG del grafo

  - `camino\_optimo\_con\_costera()`: Algoritmo de Dijkstra

  - `obtener\_ciudades()`: Lista todas las ciudades

  - `estadisticas\_grafo()`: Estadísticas del grafo

- **\*\*Tecnologías\*\***: NetworkX, matplotlib

- **\*\*Algoritmo\*\***: Dijkstra para camino más corto

---

**### \*\*🎨 CARPETA `templates/`\*\* - PLANTILLAS HTML**

**#### \*\*`base.html`\*\* - PLANTILLA BASE**

- **\*\*Función\*\***: Layout base para todas las páginas

- **\*\*Contiene\*\***: Navegación, estilos, scripts comunes

- **\*\*Framework\*\***: AdminLTE (Bootstrap)

**#### \*\*`home.html`\*\* - PÁGINA PRINCIPAL**

- **\*\*Función\*\***: Dashboard del usuario logueado

- **\*\*Características\*\***: Navegación a todas las funcionalidades

**#### \*\*`formulario\_simple.html`\*\* - BÚSQUEDA DE RUTAS**

- **\*\*Función\*\***: Formulario para buscar rutas entre ciudades

- **\*\*Características\*\***:

  - Dropdown de ciudades

  - Muestra resultado con costo

  - Visualización del grafo

**#### \*\*`ruta\_fija.html`\*\* - RUTA FIJA**

- **\*\*Función\*\***: Muestra ruta predefinida Ibarra→Loja

- **\*\*Características\*\***:

  - Ruta fija calculada

  - Costo total

  - Indicador de ciudades costeras

**#### \*\*`error.html`\*\* - PÁGINA DE ERRORES**

- **\*\*Función\*\***: Maneja errores 404, 500, etc.

- **\*\*Características\*\***: Mensajes de error amigables

**### \*\*📁 CARPETA `templates/auth/`\*\* - AUTENTICACIÓN**

**#### \*\*`login.html`\*\* - PÁGINA DE LOGIN**

- **\*\*Función\*\***: Formulario de inicio de sesión

- **\*\*Campos\*\***: Username/email, contraseña

**#### \*\*`register.html`\*\* - PÁGINA DE REGISTRO**

- **\*\*Función\*\***: Formulario de registro de usuarios

- **\*\*Campos\*\***: Username, email, contraseña, confirmación

**### \*\*📁 CARPETA `templates/admin/`\*\* - ADMINISTRACIÓN**

**#### \*\*`dashboard.html`\*\* - DASHBOARD ADMIN**

- **\*\*Función\*\***: Panel de control del administrador

- **\*\*Estadísticas\*\***: Usuarios, provincias, ciudades, rutas

**#### \*\*`usuarios.html`\*\* - GESTIÓN DE USUARIOS**

- **\*\*Función\*\***: Lista y administra usuarios del sistema

**#### \*\*`provincias.html`\*\* - GESTIÓN DE PROVINCIAS**

- **\*\*Función\*\***: CRUD completo de provincias

**#### \*\*`ciudades.html`\*\* - GESTIÓN DE CIUDADES**

- **\*\*Función\*\***: CRUD completo de ciudades

- **\*\*Características\*\***: Asignación a provincias, marca costera

**#### \*\*`rutas.html`\*\* - GESTIÓN DE RUTAS**

- **\*\*Función\*\***: CRUD completo de rutas entre ciudades

- **\*\*Características\*\***: Origen, destino, peso/costo

---

**### \*\*🎨 CARPETA `static/`\*\* - ARCHIVOS ESTÁTICOS**

**#### \*\*`static/img/`\*\* - IMÁGENES**

- **\*\*`Logo\_J.png`\*\***: Logo de la aplicación

- **\*\*`kuchau.png`\*\***: Imagen decorativa

**#### \*\*`static/plugins/`\*\* - LIBRERÍAS FRONTEND**

- **\*\*Bootstrap\*\***: Framework CSS

- **\*\*AdminLTE\*\***: Tema de administración

- **\*\*FontAwesome\*\***: Iconos

- **\*\*jQuery\*\***: JavaScript

**#### \*\*`static/dist/`\*\* - ARCHIVOS COMPILADOS**

- **\*\*CSS\*\***: Estilos de AdminLTE

- **\*\*JS\*\***: Scripts de AdminLTE

---

**### \*\*🔧 CARPETA `.venv/`\*\* - ENTORNO VIRTUAL PYTHON**

**#### \*\*¿Qué hace?\*\***

- **\*\*Función\*\***: Entorno virtual aislado con todas las dependencias Python

- **\*\*Para qué\*\***: Evita conflictos entre librerías de diferentes proyectos

- **\*\*Por qué\*\***: Buena práctica estándar en desarrollo Python profesional

**#### \*\*Contenido:\*\***

- **\*\*`Scripts/`\*\***: Ejecutables de Python y pip del entorno

- **\*\*`Lib/site-packages/`\*\***: Todas las librerías instaladas (Flask, SQLAlchemy, NetworkX, etc.)

- **\*\*`pyvenv.cfg`\*\***: Configuración del entorno virtual

**#### \*\*Comandos importantes:\*\***

```bash

# Activar entorno (NECESARIO antes de ejecutar)

.venv\Scripts\activate

# Instalar dependencias

pip install -r requirements.txt

# Ejecutar aplicación

python app.py

# Desactivar entorno

deactivate

```

**#### \*\*⚠️ IMPORTANTE:\*\***

- **\*\*NO borrar esta carpeta\*\*** - contiene todas las librerías necesarias

- **\*\*Sin .venv la aplicación NO funciona\*\***

- **\*\*Demuestra desarrollo profesional Python\*\***

---

**## 🚀 \*\*FLUJO DE LA APLICACIÓN\*\***

**### \*\*1. Inicio\*\***

1. Usuario accede a `http://localhost:4000`

2. `app.py` detecta si está logueado

3. Si no: redirige a `/login`

4. Si sí: redirige a `/home` o `/admin/dashboard`

**### \*\*2. Autenticación\*\***

1. Usuario ingresa credenciales en `login.html`

2. `auth\_routes.py` procesa el login

3. `user\_service.py` verifica credenciales

4. Flask-Login maneja la sesión

**### \*\*3. Funcionalidades\*\***

- **\*\*Usuario normal\*\***: Puede buscar rutas económicas

- **\*\*Administrador\*\***: Puede gestionar todo el sistema

**### \*\*4. Búsqueda de Rutas\*\***

1. Usuario selecciona origen y destino

2. `rutaeconomica\_routes.py` recibe la petición

3. `grafo\_utils.py` ejecuta algoritmo de Dijkstra

4. Retorna ruta óptima con costo

---

**## 🔧 \*\*TECNOLOGÍAS UTILIZADAS\*\***

**### \*\*Backend\*\***

- **\*\*Flask\*\***: Framework web de Python

- **\*\*SQLAlchemy\*\***: ORM para base de datos

- **\*\*Flask-Login\*\***: Manejo de sesiones

- **\*\*bcrypt\*\***: Encriptación de contraseñas

- **\*\*NetworkX\*\***: Algoritmos de grafos

- **\*\*matplotlib\*\***: Generación de gráficos

**### \*\*Frontend\*\***

- **\*\*AdminLTE\*\***: Framework de administración

- **\*\*Bootstrap\*\***: Framework CSS

- **\*\*jQuery\*\***: JavaScript

- **\*\*FontAwesome\*\***: Iconos

**### \*\*Base de Datos\*\***

- **\*\*MySQL\*\***: Sistema de gestión de base de datos

- **\*\*Charset UTF-8\*\***: Soporte para acentos españoles

---

**## 📊 \*\*DATOS DEL SISTEMA\*\***

**### \*\*6 Provincias de Ecuador\*\***

1. Pichincha (PIC)

2. Guayas (GUA)

3. Azuay (AZU)

4. Manabí (MAN)

5. Imbabura (IMB)

6. Loja (LOJ)

**### \*\*6 Ciudades Principales\*\***

1. **\*\*Quito\*\*** (Pichincha) - Capital

2. **\*\*Guayaquil\*\*** (Guayas) - Costera ⚓

3. **\*\*Cuenca\*\*** (Azuay)

4. **\*\*Portoviejo\*\*** (Manabí) - Costera ⚓

5. **\*\*Ibarra\*\*** (Imbabura)

6. **\*\*Loja\*\*** (Loja)

**### \*\*16 Rutas Principales\*\***

- Conexiones bidireccionales entre ciudades

- Distancias reales en kilómetros

- Ejemplo: Quito ↔ Guayaquil (420 km)

---

**## 🎯 \*\*PUNTOS CLAVE PARA LA EXPOSICIÓN\*\***

**### \*\*1. Arquitectura MVC\*\***

- **\*\*Modelos\*\***: Entidades de base de datos

- **\*\*Vistas\*\***: Templates HTML

- **\*\*Controladores\*\***: Routes y services

**### \*\*2. Principios SOLID\*\***

- **\*\*Single Responsibility\*\***: Cada clase tiene una función específica

- **\*\*Open/Closed\*\***: Extensible sin modificar código existente

- **\*\*Dependency Injection\*\***: Repositorios inyectados en servicios

**### \*\*3. Seguridad\*\***

- **\*\*Autenticación\*\***: Flask-Login

- **\*\*Encriptación\*\***: bcrypt para contraseñas

- **\*\*Autorización\*\***: Decoradores admin\_required

**### \*\*4. Algoritmos\*\***

- **\*\*Dijkstra\*\***: Para encontrar la ruta más económica

- **\*\*NetworkX\*\***: Librería especializada en grafos

**### \*\*5. Base de Datos\*\***

- **\*\*Relaciones\*\***: Foreign keys entre tablas

- **\*\*Integridad\*\***: Cascadas para eliminación

- **\*\*UTF-8\*\***: Soporte completo para español

---

**## ⚡ \*\*COMANDOS IMPORTANTES\*\***

```bash

# Inicializar base de datos

python init\_db.py

# Ejecutar aplicación

python app.py

# Acceder al sistema

http://localhost:4000

# Credenciales admin

Usuario: admin

Contraseña: admin

```

---

**## ✅ \*\*SISTEMA COMPLETAMENTE FUNCIONAL\*\***

- ✅ Autenticación de usuarios

- ✅ Panel de administración

- ✅ Gestión completa de CRUD

- ✅ Algoritmo de rutas óptimas

- ✅ Visualización de grafos

- ✅ Interfaz responsive

- ✅ Base de datos UTF-8

- ✅ Manejo de errores

- ✅ Seguridad implementada

**\*\*El proyecto está listo para producción y exposición\*\*** 🎉

Esta estructura te permitirá crear una documentación técnica completa explicando cada componente del sistema. 🎉