AgroGrid – Mapeo Completo y Arquitectura

1. Propósito General

AgroGrid es una plataforma web que conecta agricultores, compradores y transportistas en Ecuador, facilitando la compra, venta y logística de productos agrícolas. Ofrece paneles personalizados, análisis de datos, gestión de pedidos, rutas inteligentes y soporte por chatbot con IA (DeepSeek).

---

2. Estructura de Carpetas y Archivos

```
Agrogrid/
|
|-- app/
|   |-- __init__.py
|   |-- analytics.py               # Análisis de ventas, compras y paneles (Numpy/Pandas)
|   |-- analytics_historial.py     # Análisis histórico de datos
|   |-- chatbot_knowledge.py       # Base de conocimientos para respuestas instantáneas del
|   |-- chatbot_prompts.py         # Prompt y configuración del chatbot (DeepSeek)
|   |-- controllers.py             # Lógica de controladores (MVC)
|   |-- decision_tree_seguro.py    # Árbol de decisión para seguros agrícolas
|   |-- grafo_transporte.py        # Lógica de grafos y rutas (NetworkX)
|   |-- models.py                  # Modelos de base de datos (SQLAlchemy)
|   |-- routes.py                  # Todas las rutas Flask (API, vistas, autenticación, cha
|   |-- taxonomia.py               # Taxonomía de productos y categorías
|   |-- token_utils.py             # Utilidades para manejo de tokens
|   |-- ubicacion.py               # Lógica de ubicación y geolocalización
|   |-- utils/
|   |   |-- __init__.py
|   |   |-- recomendador.py        # Algoritmo de recomendación de productos
|   |   |-- sorting.py             # Algoritmos de ordenamiento
|   |   |-- readme.txt
|   |-- static/
|   |   |-- css/                   # Estilos CSS (por rol/página)
|   |   |-- js/                    # Scripts JS (por rol/página)
|   |   |-- images/                # Imágenes y logos
|   |   |-- uploads/               # Archivos subidos por usuarios
|   |   |-- barplot_envios.png     # Gráficos generados
|   |-- templates/
|   |   |-- base.html              # Plantilla base
|   |   |-- index.html             # Inicio
|   |   |-- login.html, register.html, etc.
|   |   |-- agricultor/            # Vistas específicas de agricultor
|   |   |-- comprador/             # Vistas específicas de comprador
|   |   |-- transportista/         # Vistas específicas de transportista
```

```
|   |   |-- partials/              # Componentes parciales (navbar, footer)
|   |   |-- email/                 # Plantillas de correo
|
|-- geojson/                       # Archivos geográficos para rutas y análisis espacial
|-- scripts/                       # Scripts de automatización y carga de datos
|-- migrations/                    # Migraciones de base de datos (Flask-Migrate/Alembic)
|-- .env                           # Variables de entorno (API keys, configs)
|-- config.py                      # Configuración global del proyecto
|-- requirements.txt               # Dependencias del proyecto
|-- README.md                      # Documentación general y guía de uso
|-- run.py                         # Script principal para lanzar la app Flask
|-- app.db                         # Base de datos SQLite local
|-- crear_tablas_sqlalchemy.py     # Script para crear tablas
|-- poblar_coordenadas_cantones.py  # Script para poblar coordenadas
|-- poblar_grafo_ors.py            # Script para poblar grafo de rutas
|-- poblar_grafo_vecinos.py        # Script para poblar grafo de vecinos
|-- grafo_cantonal_geodesico.json  # Grafo de rutas geodésicas
|-- grafo_cantonal_vecinos.json    # Grafo de vecinos
|-- test_openrouter.py             # Test de integración de IA
|-- testingdb.py, ver_bd_estructura.py, ver_cantones_latlon.py # Utilidades de base de datos
```

---

3. Principales Módulos y Funcionalidades

Backend (Flask)

- **app/routes.py:** Todas las rutas de la API y vistas web, integración con DeepSeek para el chatbot, lógica de paneles, autenticación, compras, ventas y logística.
- **app/models.py:** Modelos de usuario, producto, orden, carrito, vehículo, viaje, testimonio, etc.
- **app/analytics.py:** Análisis y agregación de ventas/compras, paneles de usuario.
- **app/chatbot_knowledge.py:** Preguntas frecuentes y respuestas instantáneas (minimiza uso de LLM).
- **app/chatbot_prompts.py:** Prompt optimizado para DeepSeek, instrucciones y contexto para el chatbot.
- **app/grafo_transporte.py:** Algoritmos de rutas y logística con NetworkX.
- **app/utils/**: Algoritmos de recomendación y utilidades.

Frontend (Jinja2, JS, CSS)

- **app/templates/**: Plantillas HTML para cada rol, paneles, carrito, testimonios, ayuda, etc.
- **app/static/css/**: Estilos para cada panel y página.

- **app/static/js/**: Scripts para interacción, dashboards, paneles, chatbot, etc.

Otros

- **geojson/**: Mapas y datos espaciales para rutas y análisis logístico.
- **scripts/**: Automatización, carga de datos, pruebas.
- **migrations/**: Control de versiones de la base de datos.

---

4. Integraciones y Seguridad

- **DeepSeek API:** Chatbot Gridi responde usando DeepSeek (API key en `.env`).
- **Flask-Login:** Autenticación de usuarios y gestión de sesiones.
- **Flask-Mail:** Notificaciones y recuperación de contraseña.
- **NetworkX:** Cálculo de rutas óptimas para logística.
- **Numpy/Pandas:** Análisis de datos de ventas y compras.
- **SQLAlchemy:** ORM para la base de datos.
- **Privacidad:** Datos sensibles protegidos, sin exposición en chatbot ni interfaz.

---

5. Flujos de Usuario

- **Agricultor:** Publica productos, gestiona ventas, analiza panel, recibe pagos/calificaciones.
- **Comprador:** Explora catálogo, compra, visualiza panel de compras, historial, estadísticas.
- **Transportista:** Registra vehículo, acepta viajes, visualiza rutas y calificaciones.
- **Soporte:** Acceso administrativo y ayuda.

---

6. ¿Cómo iniciar el proyecto?

1. Clonar el repositorio y crear entorno virtual.
2. Instalar dependencias (`pip install -r requirements.txt`).
3. Configurar `.env` con claves y variables.
4. Ejecutar migraciones y scripts de carga si es necesario.
5. Ejecutar `python run.py` para lanzar la app.
6. Acceder vía navegador y probar paneles, chatbot, compras, ventas, rutas, etc.

---

7. Diagrama de Arquitectura Visual y Flujos

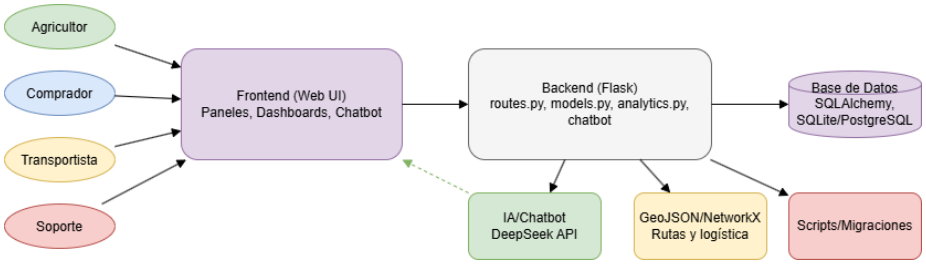

Figure 1: Diagrama de Arquitectura

- **Frontend:** Plantillas HTML (Jinja2), CSS, JS → Interfaz de usuario y paneles.
- **Backend:** Flask (routes.py, models.py, analytics.py, chatbot, lógica de negocio)
- **Base de Datos:** SQLAlchemy (SQLite, PostgreSQL, etc.)
- **IA/Chatbot:** DeepSeek API
- **GeoJSON/NetworkX:** Rutas y logística
- **Scripts y migraciones:** Automatización y mantenimiento

Flujos principales: 1. Usuario accede a la web, inicia sesión y navega su panel según rol. 2. Puede comprar, vender, transportar o solicitar soporte. 3. El chatbot responde preguntas frecuentes con la base de conocimientos y usa DeepSeek para consultas complejas. 4. Los datos de compras, ventas y rutas se analizan y muestran en paneles personalizados.

Nota: El diagrama visual debe generarse y guardarse como `AgroGrid_Arquitectura.png` en la raíz del proyecto para ser incluido en el PDF final.