Manual de usuario – Greenway

Elaborado por:
Sergio Gabriel Chaves Mosquera
Samuel Esteban Reyes Uribe
María José Sánchez Portillo

Presentado a
Néstor German Bolívar Pulgarin

Asignatura:
Programación Orientada a Objetos

Universiada Nacional de Colombia
2025 – 2s
Bogotá, DC
2025

Ficha Técnica

A continuación, se presentan los elementos usados y necesarios para el proyecto:

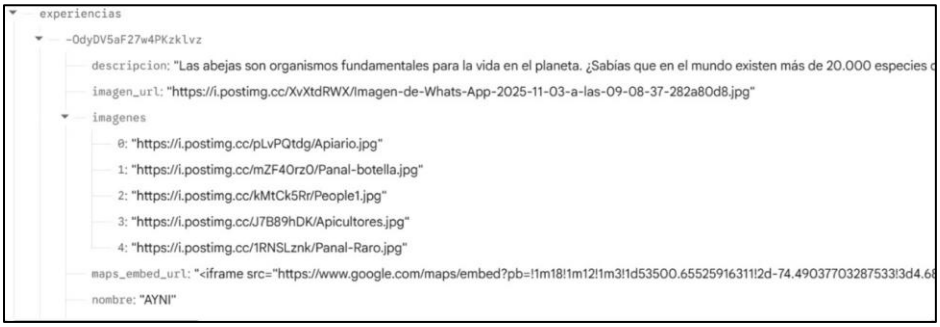
A. Backend

- Lenguaje de Programación: Python 3.13.9
- Framework Web: Flask (Microframework para el manejo de rutas y servidor).
- Servidor WSGI: Gunicorn (Para producción en Render).
- Manejo de Paquetes: pip y setuptools.

B. Base de Datos y Nube

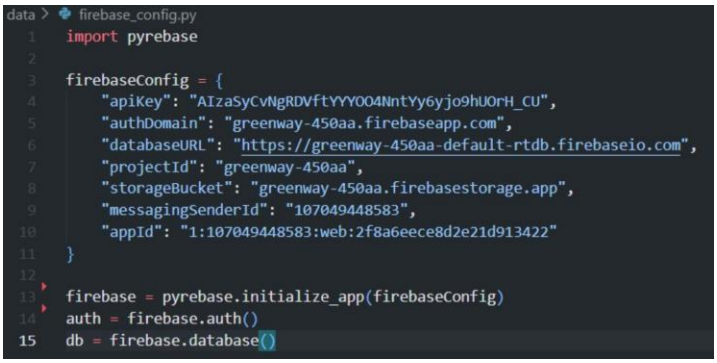
- Plataforma de Backend: Google Firebase.

- Base de Datos: Firebase Realtime Database (NoSQL, estructura de árbol JSON).

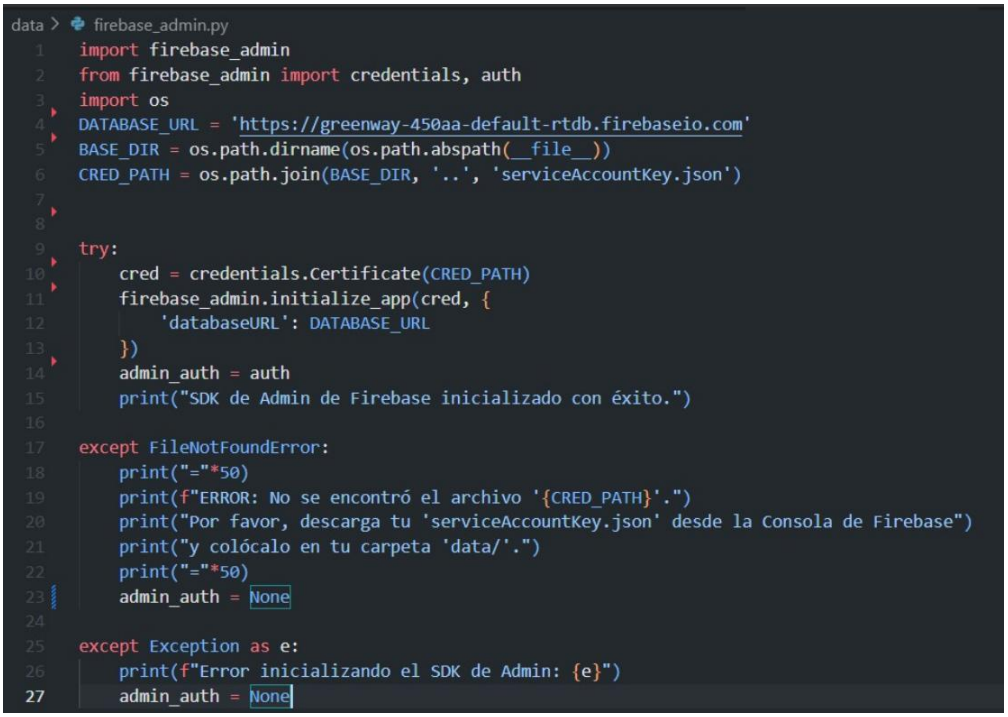


- Autenticación: Firebase Authentication (Manejo de usuarios, correos y contraseñas).

- Librería de Conexión: Pyrebase.

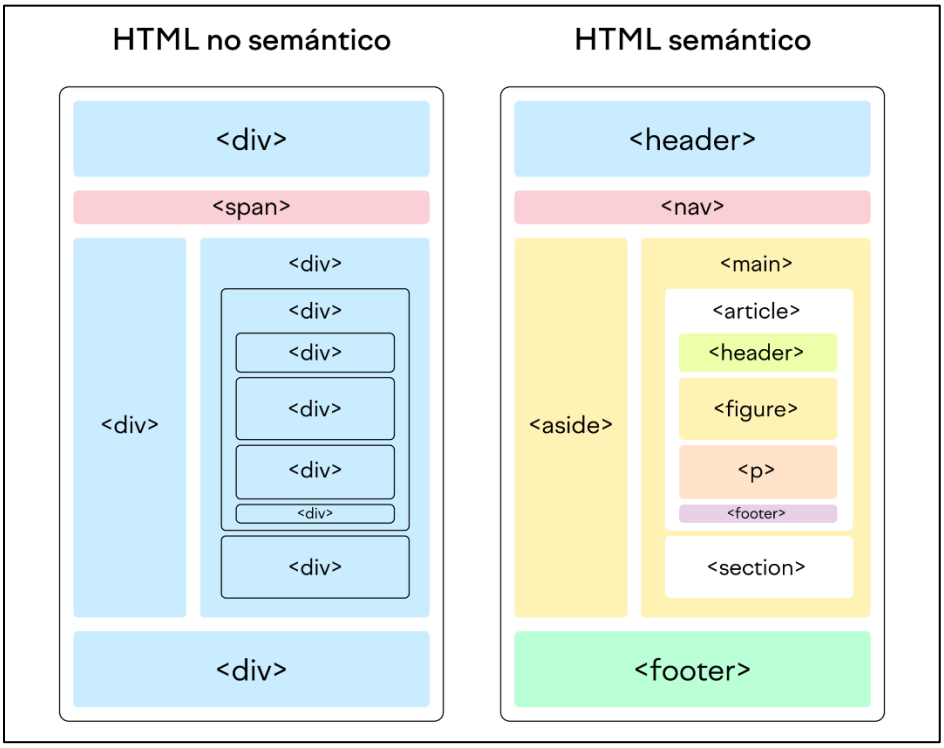


- Gestión de Admin: firebase-admin (SDK oficial para tareas como borrar usuarios o gestionar roles).



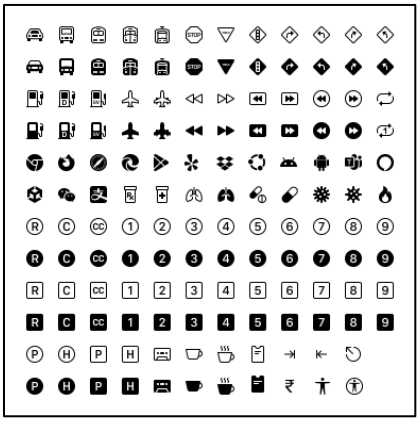
C. Frontend

- Lenguaje de Marcado: HTML5 Semántico.



- Motor de Plantillas: Jinja2

- Estilos (CSS): CSS3 Nativo, Bootstrap 5.3, Bootstrap Icons.

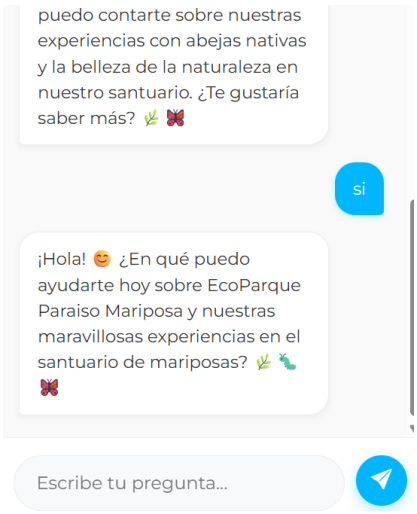


- Tipografía: Google Fonts ("Montserrat" para todo el sitio).

- Scripting: JavaScript (Vanilla JS ES6+) para la lógica del lado del cliente (chat en tiempo real, buscador, mapas).

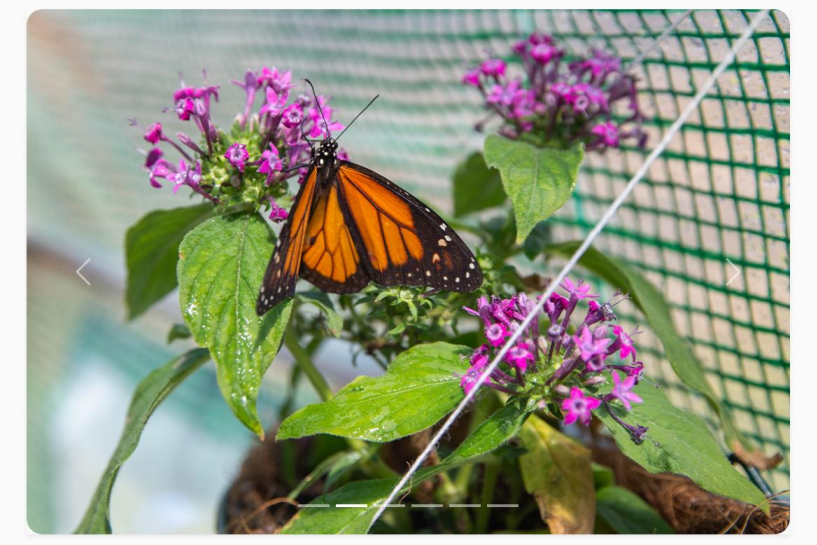
D. APIs y Servicios Externos

- Chatbot IA: OpenAI API (Modelo GPT-3.5-turbo/GPT-4) integrado mediante la librería openai.



- Mapas: Google Maps Embed API.

- Alojamiento de Imágenes: Postimages.org (Hosting externo para URLs directas).



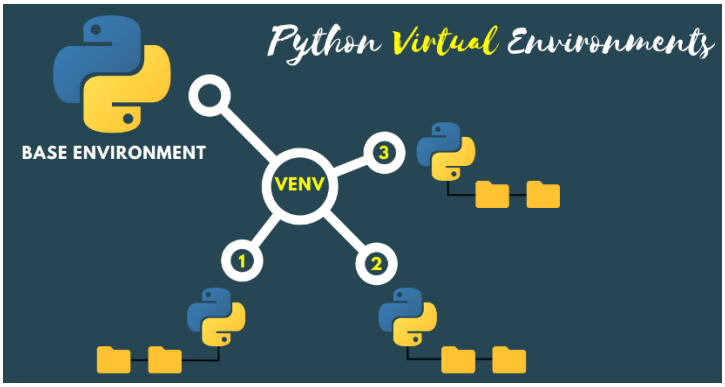
- Avatares: UI Avatars API (Generación automática de fotos de perfil con iniciales).

E. Infraestructura y Despliegue

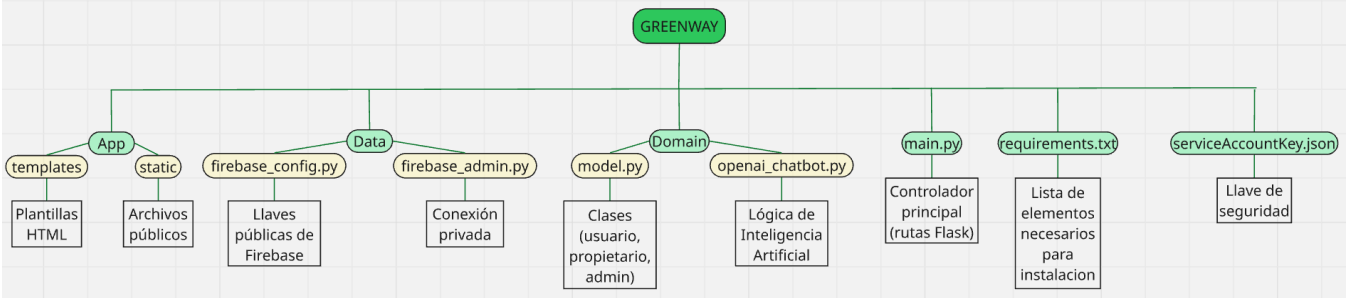
- Control de Versiones: Git y GitHub.

- Hosting (Despliegue): Render (Plataforma PaaS).

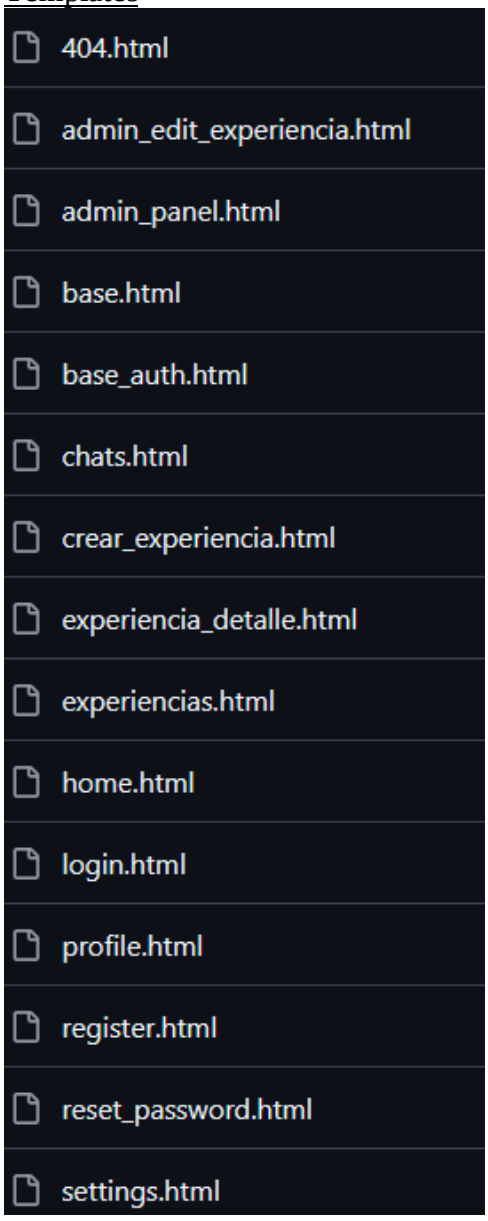
- Entorno Virtual: venv



Estructura del Proyecto



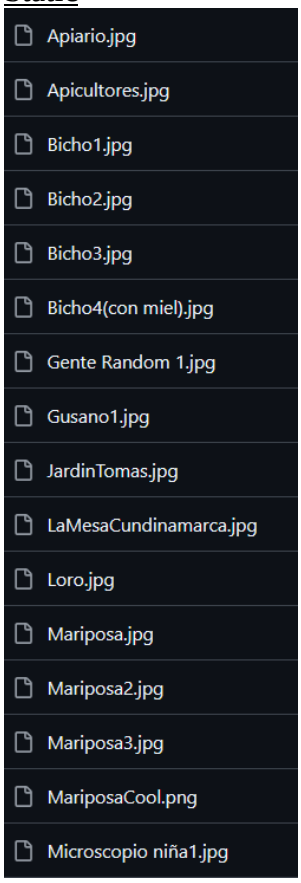
- *app:*
  - Templates




```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Bienvenido{% endblock %} - GREENWAY</title>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">
    <link href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;600;700;900&display=swap" rel="stylesheet">

    <style>
        /* --- ESTILOS COMUNES PARA TODO EL SITIO --- */

        :root {
            --greenway-primary: #A6D204; /* Verde principal */
            --greenway-accent: #00b7ff;   /* Naranja (para el chat) */
            --greenway-yellow-light: #FBEE3F; /* Amarillo alerta */
            --greenway-blue: #648BFF;       /* Azul info */
            --greenway-text: #222;
        }

        body {
            font-family: 'Montserrat', sans-serif;
            background-color: #fafafa;
            /* El padding-top se añade con JS para que sea dinámico */
        }
```

```
        /* NAVBAR (Común) */
        .navbar-greenway {
            background-color: var(--greenway-primary) !important;
            box-shadow: 0 2px 6px rgba(0,0,0,0.3);
            padding: 10px 40px;
            position: fixed;
            top: 0;
            width: 100%;
            z-index: 1030;
            transition: transform 0.3s ease-in-out;
        }
        .navbar-greenway .navbar-brand, .navbar-greenway .nav-link { color: white !important; font-weight: 600; transition: color 0.3s; }
        .navbar-greenway .nav-link:hover { color: #222 !important; }
        .navbar-greenway .btn-light { background-color: white !important; color: var(--greenway-primary) !important; font-weight: 700; border: none; transition: all 0.3s; }
        .navbar-greenway .btn-light:hover { background-color: #f8f8f8 !important; transform: scale(1.05); }
        .navbar-hidden { transform: translateY(-100%); }

        /* FOOTER (Común) */
        .footer-greenway {
            background-color: #222;
            color: #ccc;
        }
        .footer-greenway h5 { font-weight: 700; color: var(--greenway-primary); font-size: 1.5rem; }
        .footer-greenway p { font-size: 0.9rem; margin-bottom: 5px; }
        .footer-greenway strong { color: #fff; }
        .footer-greenway .copyright { background-color: rgba(0, 0, 0, 0.2); font-size: 0.8rem; }
```

- Static



- *data:*
  - firebase_config.py

```python
import pyrebase

firebaseConfig = {
    "apiKey": "AIzaSyCvNgRDVftYYYOO4NntYy6yjo9hUOrH_CU",
    "authDomain": "greenway-450aa.firebaseapp.com",
    "databaseURL": "https://greenway-450aa-default-rtdb.firebaseio.com",
    "projectId": "greenway-450aa",
    "storageBucket": "greenway-450aa.firebasestorage.app",
    "messagingSenderId": "107049448583",
    "appId": "1:107049448583:web:2f8a6eece8d2e21d913422"
}

# 1. Llamamos a la app "firebase"
firebase = pyrebase.initialize_app(firebaseConfig)

# 2. Usamos "firebase" para crear los servicios
auth = firebase.auth()
db = firebase.database()
```

- firebase_admin.py

```python
import firebase_admin
from firebase_admin import credentials, auth
import os

# --- Configuración ---

# 1. Esta es la URL de tu base de datos (de tu archivo firebase_config.py)
DATABASE_URL = 'https://greenway-450aa-default-rtdb.firebaseio.com'

# 2. Construye la ruta al archivo de clave que debe estar en esta misma carpeta ('data/')
#    os.path.abspath(__file__) -> Obtiene la ruta de este archivo (firebase_admin.py)
#    os.path.dirname(...)      -> Obtiene la carpeta que lo contiene ('data/')
#    os.path.join(...)         -> Une 'data/' + 'serviceAccountKey.json'
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
CRED_PATH = os.path.join(BASE_DIR, '..', 'serviceAccountKey.json')

# --- Fin Configuración ---

try:
    # Intenta cargar la "llave maestra" desde la ruta
    cred = credentials.Certificate(CRED_PATH)

    # Inicializa la aplicación de Admin
    firebase_admin.initialize_app(cred, {
        'databaseURL': DATABASE_URL
    })

    # Exportamos solo el módulo 'auth' del SDK de Admin
    # Lo llamamos 'admin_auth' para no confundirlo con el 'auth' de pyrebase
    admin_auth = auth
    print("SDK de Admin de Firebase inicializado con éxito.")

except FileNotFoundError:
    print("="*50)
    print(f"ERROR: No se encontró el archivo '{CRED_PATH}'.")
    print("Por favor, descarga tu 'serviceAccountKey.json' desde la Consola de Firebase")
    print("y colócalo en tu carpeta 'data/'.")
    print("="*50)
    admin_auth = None # Asigna None para que la app falle y sepas qué pasó

except Exception as e:
    print(f"Error inicializando el SDK de Admin: {e}")
    admin_auth = None
```

- *domain:*
  - Models.py

```python
class Persona:
    def __init__(self, user_id: str, nombre: str, email: str, telefono: str = ""):
        # Validación básica (El teléfono puede ser opcional al inicio, por eso el default "")
        if not all([user_id, nombre, email]):
            raise ValueError("ID, nombre y email no pueden estar vacíos")

        self.user_id = user_id
        self.nombre = nombre
        self.email = email
        self.telefono = telefono
        self.foto_url = ""  # ¡Campo nuevo para la foto de perfil!

        # Valores por defecto (se sobreescriben en las clases hijas)
        self.rol = "persona"
        self.db_node = "personas"

    def to_dict(self) -> dict:
        return {
            "nombre": self.nombre,
            "email": self.email,
            "telefono": self.telefono,
            "rol": self.rol,
            "foto_url": self.foto_url # Guardamos la foto también
        }

    def save_to_db(self, db):
        """
        Guarda la instancia en el nodo de la base de datos que le corresponde.
        """
        try:
            # print(f"Guardando en: {self.db_node}/{self.user_id}") # Debug opcional
            db.child(self.db_node).child(self.user_id).set(self.to_dict())
            return True
        except Exception as e:
```

```
            print(f"Error al guardar en la DB: {e}")
            return False # Retornamos False si falla

    @staticmethod
    def get_db_node_by_role(rol: str) -> str:
        """Método estático para saber en qué nodo buscar según el rol."""
        if rol == 'admin':
            return 'admins'
        elif rol == 'propietaria':
            return 'propietarios'
        else:
            return 'usuarios'

    @staticmethod
    def get_user_data_by_role(db, rol: str, user_id: str) -> dict:
        """
        Método estático para obtener los datos de un usuario
        buscando en el nodo correcto.
        """
        nodo = Persona.get_db_node_by_role(rol)
        try:
            return db.child(nodo).child(user_id).get().val()
        except Exception as e:
            print(f"Error al obtener datos del usuario {user_id} en {nodo}: {e}")
            return None


# --- CLASES HIJAS ---

class Usuario(Persona):
    def __init__(self, user_id: str, nombre: str, email: str, telefono: str = ""):
        super().__init__(user_id, nombre, email, telefono)
        self.rol = "usuario"
        self.db_node = "usuarios"
```

```
# --- CLASES HIJAS ---

class Usuario(Persona):
    def __init__(self, user_id: str, nombre: str, email: str, telefono: str = ""):
        super().__init__(user_id, nombre, email, telefono)
        self.rol = "usuario"
        self.db_node = "usuarios"


class Propietaria(Persona):
    def __init__(self, user_id: str, nombre: str, email: str, telefono: str = ""):
        super().__init__(user_id, nombre, email, telefono)
        self.rol = "propietaria"
        self.db_node = "propietarios"


class Admin(Persona):
    def __init__(self, user_id: str, nombre: str, email: str, telefono: str = ""):
        super().__init__(user_id, nombre, email, telefono)
        self.rol = "admin"
        self.db_node = "admins"
```

- openai_chatbot.py

```python
import os
from openai import OpenAI
from dotenv import load_dotenv

class GreenwayChatbot:
    """
    Clase orientada a objetos para manejar la lógica del chatbot
    con la API de OpenAI.
    """

    def __init__(self):
        """
        Constructor. Carga la API Key y define el "cerebro" (prompt) del bot.
        """
        # 1. Cargar la API Key desde el archivo .env que acabas de crear
        load_dotenv()
        api_key = os.getenv("OPENAI_API_KEY")

        if not api_key:
            # Si no encuentra la clave, lanza un error claro
            raise ValueError("No se encontró la API Key de OpenAI. "
                             "Asegúrate de crear un archivo .env en la raíz del proyecto "
                             "con la línea: OPENAI_API_KEY=sk-...")

        # 2. Inicializar el cliente de OpenAI
        self.client = OpenAI(api_key=api_key)

        # 3. ¡EL PROMPT CORRECTO! (El "cerebro" y "personalidad" del bot)
        # Aquí definimos las reglas de Majo y el santuario.
        self.system_prompt = """
        Eres "Greenway-bot", un asistente experto en ecoturismo y el anfitrión virtual de Greenway que es un proyecto para un santuario de mariposas llamado "EcoParque Paraiso Mariposa".

        SOBRE GREENWAY:
        - Greenway es un santuario de abejas nativas (Melipona) y un proyecto de ecoturismo (una finca) en Colombia ubicada en RNT 204618, Inspeccion La Esperanza, La Mesa, Cundinamarca.
        - El proyecto es una iniciativa familiar.
        - Tu objetivo es fomentar guiar a los clientes de la pagina, ayudarlos y convencerlos de conocer Paraiso Mariposa y el amor por la naturaleza, especialmente las abejas, tambien asi, aumentando nuestra

        TUS REGLAS:
        1. **Personalidad:** Eres amable, entusiasta, positivo y un poco "eco-consciente". Te apasiona la naturaleza y tratas de que siempre los usuarios reserven una experiencia, ayudandolos en el proceso.
        """

    def ask(self, pregunta_usuario: str) -> str:
        """
        Recibe una pregunta del usuario y devuelve la respuesta de la IA.
        """
        if not pregunta_usuario:
            return "Parece que no has escrito nada. ¡Inténtalo de nuevo!"

        try:
            # Creamos la conversación
            completion = self.client.chat.completions.create(
                model="gpt-4o-mini", # El modelo más nuevo, rápido y barato
                messages=[
                    {"role": "system", "content": self.system_prompt},
                    {"role": "user", "content": pregunta_usuario}
                ],
                max_tokens=150 # Límite para que no escriba demasiado
            )

            # Devolvemos solo el texto de la respuesta
            return completion.choices[0].message.content

        except Exception as e:
            print(f"Error en la API de OpenAI: {e}")
            # Esta es la respuesta de error que verá el usuario
            return "¡Oh, no! Parece que estoy teniendo problemas de conexión con mi cerebro de IA. Inténtalo de nuevo en un momento."
```

- *main.py:*

```python
import sys
import os
from functools import wraps
from flask import Flask, render_template, request, redirect, url_for, session, flash, Response, jsonify
from data.firebase_config import auth, db
from data.firebase_admin import admin_auth
from domain.models import Usuario, Propietaria, Admin, Persona
```

- *requirements.txt:*

```
aiohappyeyeballs==2.6.1
aiohttp==3.13.2
aiosignal==1.4.0
alpaca-trade-api==3.2.0
annotated-types==0.7.0
anyio==4.11.0
APScheduler==3.11.1
attrs==25.4.0
bcrypt==5.0.0
blinker==1.9.0
CacheControl==0.14.3
cachetools==6.2.1
certifi==2025.10.5
cffi==2.0.0
charset-normalizer==3.4.4
click==8.3.0
colorama==0.4.6
cryptography==46.0.3
deprecation==2.1.0
distlib==0.4.0
distro==1.9.0
filelock==3.20.0
firebase==4.0.1
firebase_admin==7.1.0
Flask==3.1.2
Flask-Bcrypt==1.0.1
Flask-Login==0.6.3
Flask-MySQLdb==2.0.0
Flask-WTF==1.2.2
frozenlist==1.8.0
gcloud==0.18.3
gh==0.0.4
gitdb==4.0.12
google-resumable-media==2.7.2
googleapis-common-protos==1.70.0
grpcio==1.75.1
grpcio-status==1.71.2
gunicorn==23.0.0
h11==0.16.0
h2==4.3.0
hpack==4.1.0
httpcore==1.0.9
httplib2==0.31.0
httpx==0.28.1
hyperframe==6.1.0
idna==3.11
itsdangerous==2.2.0
Jinja2==3.1.6
jiter==0.11.1
jwcrypto==1.5.6
MarkupSafe==3.0.3
msgpack==1.0.3
multidict==6.7.0
mysql-connector-python==9.4.0
mysqlclient==2.2.7
numpy==2.3.4
oauth2client==4.1.3
openai==2.6.1
packaging==25.0
pandas==2.3.3
platformdirs==4.5.0
propcache==0.4.1
proto-plus==1.26.1
protobuf==5.29.5
pyasn1==0.6.1
pyasn1_modules==0.4.2
pycparser==2.23
pycryptodome==3.23.0
pydantic==2.12.3
pydantic_core==2.41.4
PyJWT==2.10.1
PyJWT==2.10.1
pyparsing==3.2.5
Pyrebase4==4.8.0
python-dateutil==2.9.0.post0
python-dotenv==1.1.1
python-jwt==4.1.0
pytz==2025.2
PyYAML==6.0.1
requests==2.32.5
requests-toolbelt==0.10.1
rsa==4.9.1
six==1.17.0
smmap==5.0.2
sniffio==1.3.1
sseclient==0.0.27
tqdm==4.67.1
typing-inspection==0.4.2
typing_extensions==4.15.0
tzdata==2025.2
tzlocal==5.3.1
uritemplate==4.2.0
urllib3==1.26.20
virtualenv==20.35.0
websocket-client==1.9.0
websockets==10.4
Werkzeug==3.1.3
WTForms==3.2.1
yarl==1.22.0
setuptools
flask-socketio
python-socketio
python-engineio
eventlet
flask-socketio
eventlet
python-socketio
python-engineio
```

- *serviceAccountKey.json:*

```
.json
.env
serviceAccountKey.json
venv/
__pycache__/
*.pyc
.vscode/
```

Explicación para el Manual

1. Sistema de Chat en Tiempo Real

A diferencia de los sistemas tradicionales que recargan la página, Greenway utiliza una arquitectura de Escucha Activa (Listeners). El Frontend (JavaScript) se conecta directamente a los nodos de Firebase Realtime Database. Cuando un usuario envía un mensaje, este se inyecta como un objeto JSON. Automáticamente, todos los clientes conectados reciben el evento child_added, actualizando la interfaz en milisegundos sin intervención del servidor Flask.

2. Gestión de Imágenes Híbrida
Para optimizar el almacenamiento, el sistema implementa una lógica de visualización inteligente en home.html. El sistema verifica secuencialmente:

- Si existen correcciones manuales definidas por el administrador.
- Si la experiencia tiene una lista de imágenes (Formato Nuevo).
- Si la experiencia tiene una URL única (Formato Legacy).
- Si no hay datos, aplica un placeholder automático.

3. Seguridad y Roles
El sistema implementa un control de acceso basado en roles (RBAC). Mediante el decorador personalizado @role_required en Flask, se interceptan las peticiones antes de ejecutarlas, verificando si el usuario tiene el token de sesión (admin, propietaria, usuario) adecuado para ver el panel o editar recursos.