CALLTECH

MANUAL TECNICO

Sistema de agenda de contactos - CallTech

FABIAN CACERES, SEBASTIAN SANCEZ,
ADOLF ACUÑA

CALLTECH

Tabla de contenido

Introducción

En la era digital actual, la gestión eficaz de contactos es esencial, pero las soluciones existentes a menudo son limitadas o complejas. CallTech emerge como una respuesta moderna y funcional, una aplicación web desarrollada con tecnologías contemporáneas para ofrecer una gestión integral de contactos. Este proyecto aborda las deficiencias de las soluciones tradicionales, como la falta de categorización inteligente, la funcionalidad de compartir moderna (incluidos códigos QR), capacidades de búsqueda y filtrado limitadas, y la ausencia de interfaces verdaderamente adaptables a diferentes dispositivos.

CallTech es una plataforma escalable y fácil de usar, diseñada con una arquitectura modular para asegurar su mantenibilidad y extensibilidad a largo plazo. Está dirigido a una variedad de usuarios, desde profesionales independientes y pequeñas empresas hasta usuarios individuales, todos buscando una alternativa sofisticada para organizar su información de contacto. La filosofía central de CallTech se basa en tres principios fundamentales: simplicidad con funcionalidad completa, seguridad y privacidad de los datos, y adaptabilidad a diversos contextos de uso.

Objetivo del manual

Este manual técnico tiene como propósito fundamental proporcionar una documentación exhaustiva y detallada del sistema CallTech, una aplicación web desarrollada para la gestión integral de contactos. El documento está específicamente diseñado para servir como guía técnica completa para desarrolladores, administradores de sistemas y personal técnico que requiera comprender, mantener, modificar o extender las funcionalidades del sistema.

El manual abarca desde los aspectos más básicos de la instalación y configuración, hasta los detalles más complejos de la arquitectura interna, patrones de diseño implementados y estrategias de optimización. Cada sección ha sido cuidadosamente estructurada para proporcionar tanto una visión general como detalles específicos de implementación.

Alcance del proyecto

CallTech es un sistema de gestión de contactos que utiliza tecnologías web modernas, incluyendo Flask para el backend y HTML5, CSS3, y JavaScript para el frontend. Este sistema está diseñado para ser escalable, manejando desde pequeñas listas hasta bases de datos extensas. También es mantenible, con código limpio y bien documentado que facilita futuras modificaciones. La seguridad es una prioridad, con validaciones y medidas de seguridad apropiadas implementadas. Además, su interfaz es responsive, adaptándose a diferentes dispositivos y tamaños de pantalla. Actualmente, CallTech ofrece funcionalidades CRUD completas para contactos, un sistema de categorización, búsqueda avanzada, generación de códigos QR y gestión de perfiles de usuario

Público objetivo

Este manual está dirigido a Desarrolladores Backend (Que necesiten entender la lógica de negocio y estructura del servidor), Desarrolladores Frontend (Que trabajen en la interfaz de usuario y experiencia del usuario), Administradores de Sistemas (Responsables del despliegue y mantenimiento en producción), Arquitectos de Software (Que evalúen o modifiquen la arquitectura del sistema.), Estudiantes y Académicos (Que utilicen este proyecto como referencia de buenas prácticas.)

Descripción General de la Aplicación

Características Principales y Funcionalidades

CallTech es una aplicación web completa que ofrece un conjunto robusto de funcionalidades para la gestión eficiente de contactos personales y profesionales. Las características principales incluyen:

Gestión Completa de Contactos (CRUD)

- Creación de nuevos contactos con formularios intuitivos y validación en tiempo real.
- Visualización organizada de contactos en formato de lista y tarjetas.
- Edición rápida y eficiente de información existente.
- Eliminación segura con confirmaciones para prevenir pérdidas accidentales.

Sistema de categorización avanzado

- Categorías predefinidas: Familia, Amigos, Trabajo, Compañeros, Clientes, Otros.
- Asignación de colores únicos para identificación visual rápida.
- Filtrado dinámico por categorías con contadores automáticos.
- Estadísticas en tiempo real por categoría.

Búsqueda y filtrado inteligente

- Motor de búsqueda en tiempo real que opera sobre múltiples campos.
- Capacidad de búsqueda por nombre, teléfono, email, empresa y notas.
- Combinación de filtros de categoría con términos de búsqueda.
- Resaltado de términos de búsqueda en los resultados.

Gestión de Perfil Personal

- Creación y mantenimiento de perfil personal completo.
- Campos personalizables: nombre, teléfono, email, empresa, biografía.
- Actualización en tiempo real con validación de datos.

Funcionalidad de Compartir

- Generación automática de códigos QR para perfiles.
- Enlaces directos para compartir información de contacto.
- Compatibilidad con dispositivos móviles para escaneo QR.

Detección Inteligente de Duplicados

- Algoritmos de detección que comparan múltiples campos.
- Alertas preventivas antes de crear contactos duplicados.
- Sugerencias de fusión de contactos similares

Beneficios del Sistema

Para Usuarios Finales:

- Interfaz intuitiva que reduce la curva de aprendizaje.
- Acceso rápido a información de contacto mediante búsqueda avanzada.
- Organización eficiente mediante categorización visual.
- Capacidad de compartir información fácilmente.

Para Desarrolladores:

- Código modular y bien estructurado.
- Documentación completa y comentarios descriptivos.
- Patrones de diseño reconocidos y buenas prácticas.
- Facilidad de extensión y mantenimiento.

Para Organizaciones:

- Solución escalable que crece con las necesidades.
- Bajo costo de implementación y mantenimiento.
- Seguridad de datos implementada desde el diseño.
- Compatibilidad con estándares web modernos.

Casos de Uso Principales

Caso de Uso 1: Profesional Independiente

Un consultor que necesita organizar contactos de clientes, proveedores y colegas. Utiliza las categorías para separar tipos de contactos y la función de búsqueda para encontrar rápidamente información específica durante reuniones.

Caso de Uso 2: Pequeña Empresa

Un startup que requiere un sistema centralizado para gestionar contactos de clientes potenciales, socios y empleados. Aprovecha la funcionalidad de compartir perfiles para facilitar el networking en eventos.

Caso de Uso 3: Uso Personal

Un individuo que desea organizar sus contactos personales de manera más eficiente que las aplicaciones tradicionales, utilizando categorías personalizadas y la capacidad de generar códigos QR para compartir su información.

Arquitectura del Proyecto

Estructura de carpetas

La arquitectura del proyecto CallTech sigue una estructura modular y organizada que facilita el mantenimiento y la escalabilidad. La organización de carpetas está diseñada siguiendo las mejores prácticas de desarrollo web con Flask:

```
calltech/
├── app.py                    # Aplicación principal Flask - Punto de ent
├── models.py                 # Definición de modelos de datos SQLAlchemy
├── forms.py                  # Formularios WTForms para validación
├── utils.py                  # Funciones utilitarias y helpers
├── config.py                 # Configuraciones de la aplicación
├── wsgi.py                   # Punto de entrada WSGI para producción
├── requirements.txt          # Dependencias del proyecto
├── README.md                 # Documentación principal del proyecto
├── .gitignore                # Archivos ignorados por Git
├── static/                   # Archivos estáticos del frontend
│   ├── css/
│   │   └── style.css         # Estilos personalizados
│   ├── js/
│   │   └── main.js           # JavaScript principal
│   └── images/               # Recursos gráficos
├── templates/                # Plantillas HTML Jinja2
│   ├── base.html             # Plantilla base con layout común
│   ├── index.html            # Página principal - lista de contactos
│   ├── add_contact.html      # Formulario para agregar contactos
│   ├── edit_contact.html     # Formulario para editar contactos
│   ├── view_contact.html     # Vista detallada de contacto
│   ├── my_profile.html       # Página de perfil personal
│   ├── share_profile.html    # Página para compartir perfil
│   ├── about.html            # Página "Quienes Somos"
│   └── register.html         # Formulario de registro (futuro)
└── instance/                 # Datos específicos de la instancia
    ├── calltech.db           # Base de datos SQLite
    └── config.py             # Configuración específica de instancia
```

Descripción detallada de componentes

Archivos de configuración

- app.py: Contiene la lógica principal de la aplicación, definición de rutas y configuración inicial.
- config.py: Centraliza todas las configuraciones, permitiendo diferentes entornos (desarrollo, producción).
- wsgi.py: Proporciona el punto de entrada para servidores WSGI en producción

Capa de Datos

- models.py: Define la estructura de datos usando SQLAlchemy ORM.
- instance/calltech.db: Base de datos SQLite para desarrollo local.

Capa de Presentación

- templates/: Contiene todas las plantillas HTML usando el motor Jinja2.
- static/: Recursos estáticos (CSS, JavaScript, imágenes).

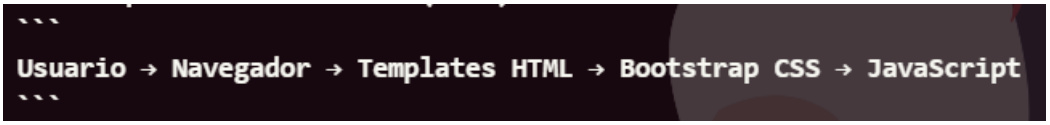
Utilidades y Helpers

- utils.py: Funciones auxiliares reutilizables.
- forms.py: Definición de formularios con validación.

Flujo de datos

El flujo de datos en CallTech sigue el patrón MVC (Model-View-Controller) adaptado para aplicaciones web Flask

Capa de presentación (view)



- Los usuarios interactúan con la interfaz web.
- Las plantillas Jinja2 renderizan contenido dinámico.
- Bootstrap proporciona estilos responsive.
- JavaScript maneja interactividad del lado cliente.

Capa de Aplicación (Controller)

- Las rutas Flask reciben y procesan peticiones HTTP.
- La lógica de negocio valida y procesa datos.
- Se generan respuestas apropiadas (HTML, JSON, redirects).

Capa de Datos (Model)

- Los modelos SQLAlchemy definen la estructura de datos.
- SQLite proporciona persistencia local.
- Las operaciones CRUD se manejan a través del ORM

Flujo Típico de una Operación:

- Solicitud del Usuario: El usuario hace clic en "Agregar Contacto".
- Routing: Flask dirige la solicitud a la ruta /add.
- Procesamiento: Se carga el formulario y se validan los datos.
- Persistencia: Si es válido, se guarda en la base de datos.
- Respuesta: Se redirige al usuario con un mensaje de confirmación.
- Renderizado: Se actualiza la vista con el nuevo contacto.

Patrones de Diseño Implementados

Patrón MVC (Model-View-Controller)

- Model: models.py - Gestión de datos y lógica de negocio.
- View: templates/ - Presentación e interfaz de usuario.
- Controller: app.py - Lógica de aplicación y coordinación.

Patrón Repository

- Abstracción de acceso a datos a través de SQLAlchemy.

- Separación clara entre lógica de negocio y persistencia.
- Facilita testing y cambios de base de datos.

Patrón Factory

- Configuración de la aplicación Flask usando factory pattern.
- Permite múltiples configuraciones (desarrollo, testing, producción).
- Facilita la inicialización de componentes.

Patrón Template Method

- Plantilla base (base.html) define estructura común.
- Plantillas específicas extienden y personalizan contenido.
- Reutilización de código y consistencia visual.

Tecnologías utilizadas

Uso de Python/Flask

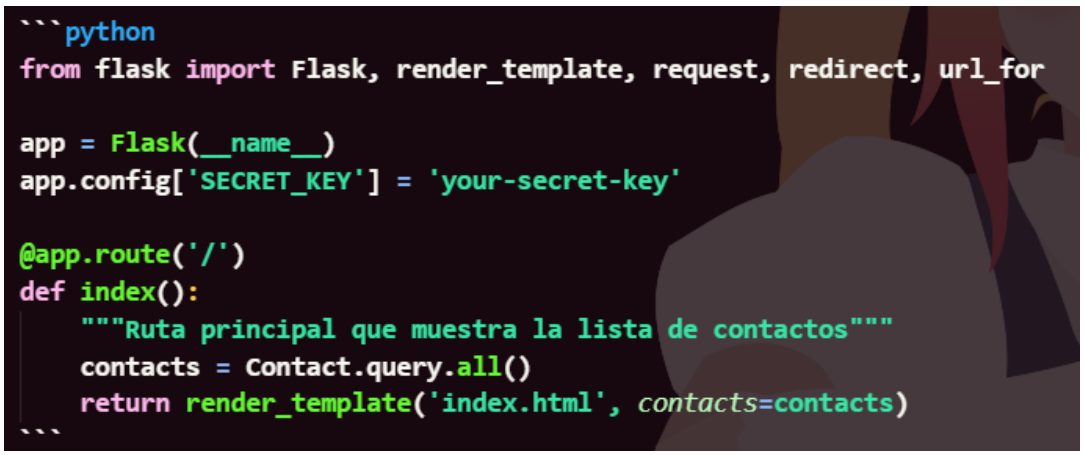
Python 3.8+

Python fue seleccionado como lenguaje principal debido a su:

- Sintaxis clara y legible que facilita el mantenimiento.
- Amplio ecosistema de librerías para desarrollo web.
- Excelente soporte para prototipado rápido y desarrollo ágil.
- Comunidad activa y documentación extensa.

Flask Framework

Flask es un microframework web que proporciona:



```python
from flask import Flask, render_template, request, redirect, url_for

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key'

@app.route('/')
def index():
    """Ruta principal que muestra la lista de contactos"""
    contacts = Contact.query.all()
    return render_template('index.html', contacts=contacts)
```

Ventajas de Flask

- Flexibilidad para estructurar la aplicación según necesidades específicas.
- Curva de aprendizaje suave comparado con frameworks más complejos.
- Excelente integración con SQLAlchemy para manejo de bases de datos.
- Sistema de plantillas Jinja2 potente y flexible.

Extensiones Flask Utilizadas

- Flask-SQLAlchemy: ORM para manejo de base de datos.
- Flask-WTF: Manejo de formularios y validación CSRF.
- Werkzeug: Utilidades WSGI y herramientas de desarrollo

Base de Datos SQLite

SQLite como Motor de Base de Datos

SQLite fue elegido para el desarrollo por:

- No requiere instalación o configuración de servidor.
- Base de datos embebida ideal para desarrollo y testing.
- Soporte completo de SQL con transacciones ACID.
- Facilidad de backup y migración.

SQLAlchemy como ORM

```python
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Contact(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(120), unique=True)
    phone = db.Column(db.String(20))
    category_id = db.Column(db.Integer, db.ForeignKey('category.id'))

    def to_dict(self):
        """Convierte el objeto a diccionario para JSON"""
        return {
            'id': self.id,
            'name': self.name,
            'email': self.email,
            'phone': self.phone
        }
```

Beneficios del ORM

- Abstracción de diferencias entre motores de base de datos.
- Prevención de inyección SQL mediante consultas parametrizadas.
- Migraciones automáticas y versionado de esquema.
- Relaciones entre modelos fáciles de definir y usar

Frontend con Bootstrap

Bootstrap 5

Framework CSS que proporciona:

- Sistema de *grid responsive* de 12 columnas.
- Componentes predefinidos (botones, formularios, modales).
- Utilidades CSS para espaciado, colores y tipografía.
- JavaScript para componentes interactivos

```html
<!-- Ejemplo de uso de Bootstrap en templates -->
<div class="container">
    <div class="row">
        <div class="col-md-6">
            <div class="card">
                <div class="card-body">
                    <h5 class="card-title">{{ contact.name }}</h5>
                    <p class="card-text">{{ contact.email }}</p>
                </div>
            </div>
        </div>
    </div>
</div>
```

HTML5 y CSS3

- Semántica moderna con elementos como section, article, nav.
- CSS Grid y Flexbox para layouts avanzados.
- Variables CSS para temas personalizables.
- Animaciones y transiciones suaves

JavaScript Vanilla

```javascript
// Búsqueda en tiempo real
function searchContacts() {
    const searchTerm = document.getElementById('search').value.toLowerCase();
    const contacts = document.querySelectorAll('.contact-card');

    contacts.forEach(contact => {
        const text = contact.textContent.toLowerCase();
        contact.style.display = text.includes(searchTerm) ? 'block' : 'none';
    });
}
```

Otras herramientas y dependencias

Librerías Python principales

```python
# requirements.txt
Flask==2.3.3
Flask-SQLAlchemy==3.0.5
Flask-WTF==1.1.1
WTForms==3.0.1
qrcode==7.4.2
Pillow==10.0.1
email-validator==2.0.0
```

qrcode + Pillow

Para generación de códigos QR:

```python
import qrcode
from PIL import Image

def generate_qr_code(data):
    """Genera código QR para compartir perfil"""
    qr = qrcode.QRCode(version=1, box_size=10, border=5)
    qr.add_data(data)
    qr.make(fit=True)

    img = qr.make_image(fill_color="black", back_color="white")
    return img
```

WTForms

Para validación de formularios

```python
from wtforms import StringField, EmailField, SelectField
from wtforms.validators import DataRequired, Email, Optional

class ContactForm(FlaskForm):
    name = StringField('Nombre', validators=[DataRequired()])
    email = EmailField('Email', validators=[Optional(), Email()])
    phone = StringField('Teléfono')
    category = SelectField('Categoría', coerce=int)
```

Herramientas de desarrollo

- Git: Control de versiones distribuido.
- VS Code: Editor de código con extensiones para Python y Flask.
- Flask Debug Toolbar: Herramientas de debugging en desarrollo.

- Postman: Testing de APIs y endpoints

Configuración del proyecto

Requisitos del Sistema:

Hardware Mínimo:

- Procesador: 1 GHz o superior.
- RAM: 512 MB disponibles.
- Espacio en disco: 100 MB libres.
- Conexión a internet: Para instalación de dependencias.

Hardware Recomendado:

- Procesador: 2 GHz dual-core o superior.
- RAM: 2 GB disponibles.
- Espacio en disco: 1 GB libres.
- SSD: Para mejor rendimiento de base de datos.

Software requerido

Python 3.8 o superior

```bash
# Verificar versión de Python
python --version
# o
python3 --version

# Debe mostrar Python 3.8.x o superior
```

pip (Gestor de paquetes de Python):

```bash
# Verificar pip
pip --version
# o
python -m pip --version
```

Git (Control de versiones):

```bash
# Verificar Git
git --version
```

Navegador Web Moderno:

- Chrome 80+ (recomendado).
- Firefox 75+.
- Safari 13+.
- Edge 80+.

Sistemas Operativos Soportados:

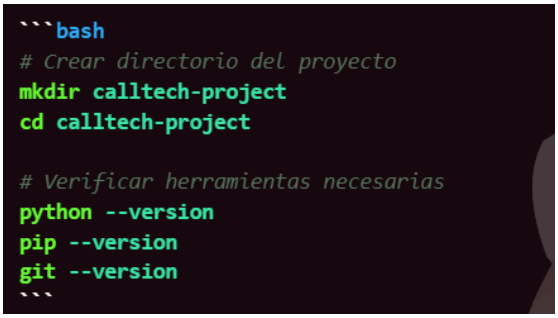
- Windows 10/11.
- macOS 10.14 (Mojave) o superior.
- Ubuntu 18.04 LTS o superior.
- Debian 10 o superior.
- CentOS 7/8.
- Fedora 30+.

Herramientas de Desarrollo Opcionales:

- VS Code con extensiones Python.
- PyCharm Community/Professional.
- Sublime Text con paquetes Python.

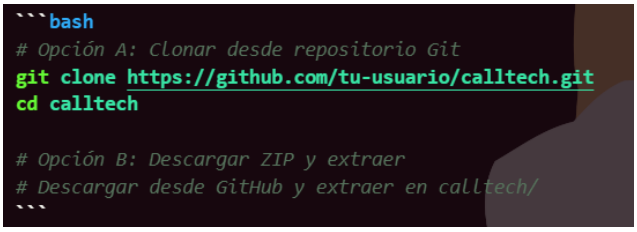
Instrucciones de Instalación

Paso 1: Preparación del Entorno



```bash
# Crear directorio del proyecto
mkdir calltech-project
cd calltech-project

# Verificar herramientas necesarias
python --version
pip --version
git --version
```

Paso 2: Clonar el Repositorio



```bash
# Opción A: Clonar desde repositorio Git
git clone https://github.com/tu-usuario/calltech.git
cd calltech

# Opción B: Descargar ZIP y extraer
# Descargar desde GitHub y extraer en calltech/
```

Paso 3: Crear Entorno Virtual

```bash
# Crear entorno virtual
python -m venv venv

# Activar entorno virtual
# En Windows:
venv\Scripts\activate

# En macOS/Linux:
source venv/bin/activate

# Verificar activación (debe mostrar (venv) en el prompt)
which python  # macOS/Linux
where python   # Windows
```

Paso 4: Instalar Dependencias

```bash
# Actualizar pip a la última versión
python -m pip install --upgrade pip

# Instalar dependencias del proyecto
pip install -r requirements.txt

# Verificar instalación
pip list
```

Paso 5: Verificar Instalación

```bash
# Verificar que Flask esté instalado
python -c "import flask; print(flask.__version__)"

# Verificar SQLAlchemy
python -c "import sqlalchemy; print(sqlalchemy.__version__)"

# Verificar otras dependencias críticas
python -c "import qrcode, PIL; print('QR y PIL OK')"
```

Inicialización del Proyecto

Configuración Inicial de la Base de Datos:

```python
# El archivo app.py incluye inicialización automática
def create_tables():
    """Crear tablas y datos iniciales"""
    with app.app_context():
        db.create_all()
        init_default_categories()

        # Crear perfil de usuario por defecto
        if not UserProfile.query.first():
            default_profile = UserProfile(
                name="Mi Perfil CallTech",
                phone="",
                email="",
                company="",
                bio="Comparte tu información de contacto fácilmente"
            )
            db.session.add(default_profile)
            db.session.commit()
```

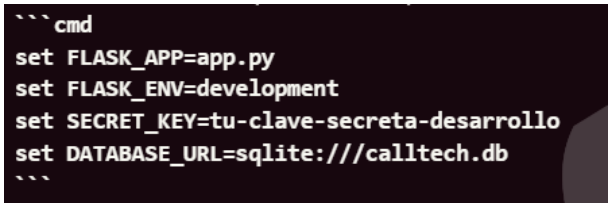
Primera Ejecución

Verificación de Funcionamiento:

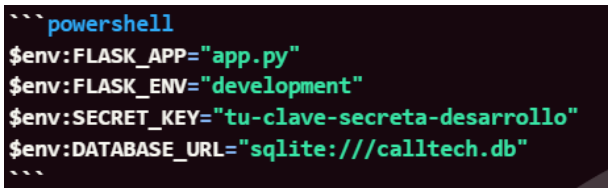
- Abrir navegador en http://localhost:5000.
- Verificar que la página principal carga correctamente.
- Probar navegación a "Agregar Contacto".
- Verificar que la base de datos se crea en instance/calltech.db.

Configuración de variables de entorno

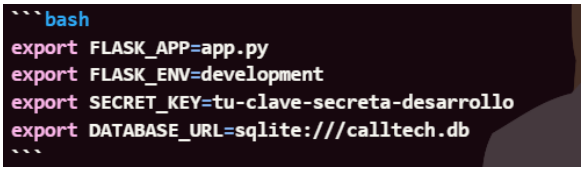
Para Desarrollo (Windows CMD)



Para Desarrollo (Windows PowerShell)



Para Desarrollo (macOS/Linux)



Archivo .env (Recomendado)

```bash
# Crear archivo .env en la raíz del proyecto
FLASK_APP=app.py
FLASK_ENV=development
SECRET_KEY=tu-clave-secreta-muy-segura
DATABASE_URL=sqlite:///calltech.db
DEBUG=True
```

Para Producción

```bash
export FLASK_ENV=production
export SECRET_KEY=clave-super-secreta-produccion
export DATABASE_URL=postgresql://usuario:password@localhost/calltech
export DEBUG=False
```

Configuración en config.py

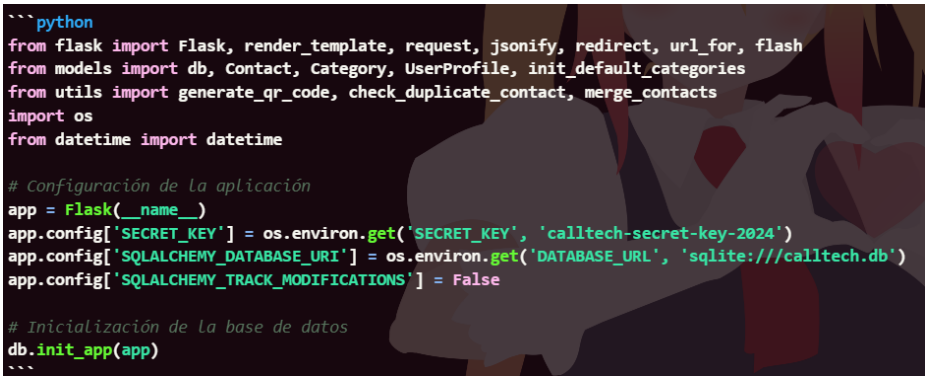
```python
import os
from dotenv import load_dotenv

load_dotenv()

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'dev-secret-key'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or 'sqlite:///calltech.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    DEBUG = os.environ.get('DEBUG', 'False').lower() == 'true'

class DevelopmentConfig(Config):
    DEBUG = True
    SQLALCHEMY_DATABASE_URI = 'sqlite:///calltech-dev.db'

class ProductionConfig(Config):
    DEBUG = False
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL')

config = {
    'development': DevelopmentConfig,
    'production': ProductionConfig,
    'default': DevelopmentConfig
}
```

Detalles de la Implementación

Módulo Principal (app.py)

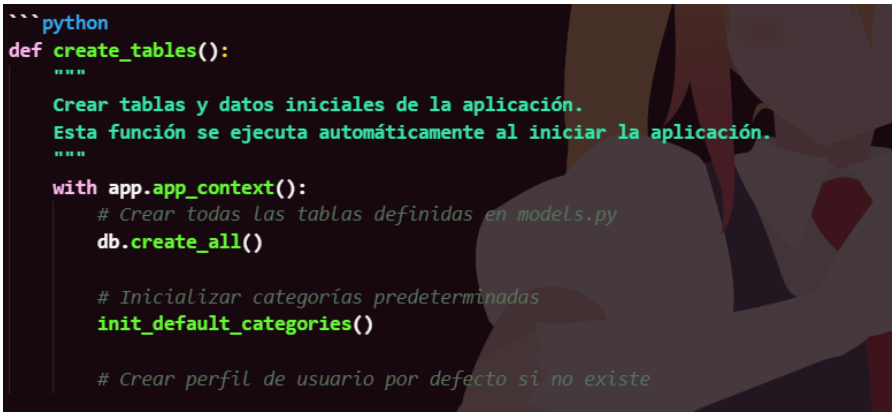
El archivo app.py constituye el núcleo de la aplicación CallTech, conteniendo la configuración principal, definición de rutas y lógica de negocio central.

Estructura General



```python
from flask import Flask, render_template, request, jsonify, redirect, url_for, flash
from models import db, Contact, Category, UserProfile, init_default_categories
from utils import generate_qr_code, check_duplicate_contact, merge_contacts
import os
from datetime import datetime

# Configuración de la aplicación
app = Flask(__name__)
app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY', 'calltech-secret-key-2024')
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL', 'sqlite:///calltech.db')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Inicialización de la base de datos
db.init_app(app)
```

Función de Inicialización



```python
def create_tables():
    """
    Crear tablas y datos iniciales de la aplicación.
    Esta función se ejecuta automáticamente al iniciar la aplicación.
    """
    with app.app_context():
        # Crear todas las tablas definidas en models.py
        db.create_all()

        # Inicializar categorías predeterminadas
        init_default_categories()

        # Crear perfil de usuario por defecto si no existe
```