Reporte del Proyecto

Rutas

- **Archivo:** admin_dashboard.py
  Propósito: Este código define una ruta de Flask para el panel de administración de un sistema. Al acceder a la ruta "/admin", se verifica si el usuario tiene una sesión activa y si es un administrador. Luego, se realizan consultas a una base de datos Supabase para contar el número de registros en las tablas "configuracion_bot" y "logs_errores", mostrando mensajes de información sobre los resultados obtenidos o posibles errores en el proceso. En
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: admin_dashboard.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** admin_nora.py
  Propósito: Este código es un script de Python que utiliza el framework Flask para crear una ruta de edición de configuración para un bot llamado "Nora". La ruta "/admin/nora//editar" permite editar la configuración de un bot específico. El script carga la configuración del bot desde una base de datos en Supabase, valida los datos enviados en un formulario y actualiza la configuración en la base de datos. También maneja errores en caso
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, datetime
- **Archivo:** admin_noras.py
  Propósito: El código tiene como propósito cargar un módulo de Python llamado "admin_noras.py" y configurar un Blueprint en Flask para manejar rutas relacionadas con la administración de Noras (posiblemente un tipo de entidad o configuración). Utiliza la librería Supabase para interactuar con una base de datos y cargar información sobre Noras y tickets pendientes. La función `vista_admin()` consulta las Noras y los tickets pendientes desde
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: admin_noras.html, admin_noras.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, datetime
- **Archivo:** admin_nora_dashboard.py
  Propósito: El código es un script de Python que carga un dashboard para un usuario llamado Nora. Utiliza Flask para crear una ruta que carga la información del dashboard desde una base de datos en Supabase. El código carga la configuración, contactos, respuestas y tickets relacionados con el usuario Nora desde diferentes tablas en la base de datos de Supabase. Muestra mensajes de estado (éxito, advertencia o error) durante el proceso de carga de la información.
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: admin_nora_dashboard.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, datetime
- **Archivo:** api_mensajes.py
  Propósito: Este código define un Blueprint de Flask llamado `api_mensajes` que contiene una ruta `/api/enviar_mensaje` para enviar mensajes de texto por WhatsApp a través de Twilio. La función `enviar_mensaje_por_twilio` se encarga de enviar mensajes utilizando la API de Twilio, mientras que la función

`enviar_mensaje_api` es la que se encarga de procesar las solicitudes POST para enviar mensajes. En resumen, el propósito de este
Rutas: Ninguna
Archivos CSS: Ninguno
Archivos HTML: Ninguno
Tablas y columnas:
Archivos .py usados: flask, werkzeug.utils, utils.normalizador, utils.historial, utils.error_logger, twilio.rest

- **Archivo:** categorias.py
  Propósito: Este código define un Blueprint de Flask llamado 'categorias' que maneja las operaciones relacionadas con categorías en una aplicación web. El Blueprint incluye tres rutas: 1. `/categorias`: Muestra todas las categorías almacenadas en la base de datos de Supabase. 2. `/categorias/agregar`: Agrega una nueva categoría a la base de datos de Supabase. 3. `/categorias/eliminar/`: Elimina una categoría específica de
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: categorias.html, editar_categoria.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, utils.config
- **Archivo:** chat_data.py
  Propósito: Este código es un endpoint de una API construida con Flask que devuelve el historial de conversaciones de un cliente específico. El endpoint espera un número de teléfono como parámetro, normaliza el número utilizando una función de normalización importada, consulta la base de datos Supabase para obtener el historial de conversaciones asociado a ese número, y devuelve los datos en formato JSON. En caso de que no se encuentren datos o ocurra algún error, se devuelve un
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, clientes.aura.utils.normalizador
- **Archivo:** debug.py
  Propósito: Este código es un script de depuración para una aplicación web Flask que proporciona rutas para verificar y manipular datos en una base de datos de Supabase. - Importa varios módulos y funciones necesarios. - Configura la conexión con Supabase utilizando las credenciales almacenadas en variables de entorno. - Define un Blueprint llamado "debug" que contiene varias rutas: 1. "/debug/verificar" que llama a la función
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: debug_verificacion.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, clientes.aura.utils.debug_integracion, clientes.aura.utils.twilio_sender, clientes.aura.utils.normalize, twilio.rest
- **Archivo:** debug_env.py
  Propósito: Este código define una ruta en una aplicación Flask que muestra información sobre variables de entorno críticas para el funcionamiento de la aplicación. Al acceder a la ruta "/debug/env" mediante una solicitud GET, se comprueba la presencia de ciertas claves críticas en las variables de entorno del sistema. Si alguna de estas claves está ausente, se marca como faltante. Además, se verifica y se divide la variable "ADMIN_EMAILS" en correos
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: debug_env.html
  Tablas y columnas:
  Archivos .py usados: flask
- **Archivo:** debug_google.py
  Propósito: El código proporciona una ruta de depuración para verificar la configuración de autenticación con Google.

Primero, se comprueba la presencia de las credenciales de cliente de Google (ID, secreto y URI de redirección). Luego, se genera una URL de autorización OAuth2 para iniciar el flujo de autenticación con Google. Finalmente, se comprueba si la URL generada coincide con la URI esperada y se almacena un estado
Rutas: Ninguna
Archivos CSS: Ninguno
Archivos HTML: Ninguno
Tablas y columnas:
Archivos .py usados: flask, requests_oauthlib, dotenv

- **Archivo:** debug_login.py
  Propósito: El código proporciona una ruta en una aplicación Flask para obtener información de inicio de sesión de un usuario. Si el usuario está en sesión, devuelve un JSON con detalles como si está conectado, su nombre, correo electrónico y si es administrador. Si no hay usuario en sesión, devuelve un JSON indicando que no hay usuario conectado actualmente.
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask
- **Archivo:** debug_openai.py
  Propósito: El propósito de este código es proporcionar rutas para probar la integración con la API de OpenAI a través de un servidor Flask. La ruta "/debug/openai" permite enviar una pregunta como parámetro y obtener una respuesta generada por el modelo GPT-3.5 de OpenAI. Si la conexión con OpenAI falla, se devuelve un mensaje de error. Además, hay una función auxiliar llamada "verificar_openai" que verifica la versión de
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, dotenv
- **Archivo:** debug_routes.py
  Propósito: El código proporciona un conjunto de rutas de depuración para una aplicación web Flask. Al acceder a la ruta "/debug/rutas" mediante un método GET, se mostrará una tabla HTML que contiene información sobre las rutas registradas en la aplicación, incluyendo la ruta en sí, el endpoint asociado y los métodos permitidos para cada ruta. El propósito principal de este código es permitir a los desarrolladores visualizar y depurar las rutas registradas
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask
- **Archivo:** debug_test_temp.py
  Propósito: El código es un script de Python que carga un archivo llamado "debug_test_temp.py" correctamente e importa las funciones necesarias de Flask, como Blueprint y jsonify. Luego, crea un Blueprint llamado "debug_test_temp_bp" con una ruta "/debug/test_temp" que devuelve un objeto JSON con un valor booleano "ok" y un mensaje. En resumen, el propósito de este código es definir una ruta en una aplicación Flask que devuelve un objeto JSON con un
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask
- **Archivo:** debug_verificar.py
  Propósito: El código proporciona una ruta de depuración en una aplicación web Flask llamada "debug_verificar" que

verifica la configuración de varios servicios y variables de entorno necesarias para el correcto funcionamiento de la aplicación. Realiza comprobaciones como verificar la conexión con OpenAI, OAuthLib, Google Login, y la presencia de ciertas variables de entorno. Además, verifica la existencia de datos en tablas específicas en una base de datos Supabase.
Rutas: Ninguna
Archivos CSS: Ninguno
Archivos HTML: debug_verificacion.html
Tablas y columnas:
Archivos .py usados: flask, dotenv, supabase, clientes.aura.routes.debug_openai, clientes.aura.routes.debug_oauthlib, clientes.aura.routes.debug_google, twilio.rest, urllib.parse

- **Archivo:** envios_programados.py
  Propósito: Este código es un script de Python que utiliza el framework Flask para crear una API y una interfaz web. El código establece una conexión con la base de datos Supabase y define diferentes rutas para la API y las vistas web. - Se carga la configuración de Supabase desde variables de entorno. - Se definen rutas para mostrar una vista de envíos programados, obtener contactos desde la base de datos, obtener envíos programados y programar
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: panel_envios_programados.html
  Tablas y columnas:
  Archivos .py usados: flask, datetime, supabase, dotenv
- **Archivo:** main.py
  Propósito: Este código es una aplicación web construida con Flask que interactúa con una base de datos en Supabase. La aplicación incluye las siguientes funcionalidades: 1. Configuración de Supabase: Carga las credenciales de Supabase desde variables de entorno y crea un cliente Supabase. 2. Middleware `login_requerido`: Función que verifica si el usuario está logueado antes de permitir el acceso a ciertas rutas. 3.
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: login.html, panel_conversaciones.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, utils.config, app
- **Archivo:** normalizador.py
  Propósito: El código proporciona una API para enviar mensajes de texto a través de Twilio. La función `enviar_mensaje_por_twilio` se encarga de enviar un mensaje de texto a un número de teléfono utilizando la plataforma Twilio. La API definida en la ruta `/api/enviar_mensaje` recibe un número de teléfono, un mensaje de texto y opcionalmente un archivo adjunto. El número de teléfono se normaliza, se guarda el mensaje en un histor
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, werkzeug.utils, utils.normalizador, utils.historial, utils.error_logger, twilio.rest
- **Archivo:** panel_cliente.py
  Propósito: El código es un script de Python que define rutas para un panel de cliente en una aplicación web utilizando Flask. También configura la conexión con la base de datos de Supabase para cargar y almacenar información relacionada con la configuración de un bot de inteligencia artificial. En resumen, el código carga la configuración de Supabase, define rutas para acceder al panel del cliente y al área de entrenamiento del bot, verifica la sesión del usuario y renderiza plant
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv

- **Archivo:** panel_cliente_contactos.py
  Propósito: Este código es un script de Python que carga un módulo de Flask para manejar rutas relacionadas con los contactos de un cliente en un panel de administración. El código primero establece una conexión con la base de datos Supabase utilizando las credenciales proporcionadas en variables de entorno. Luego define una ruta `/panel_cliente/contactos/` que carga los contactos de un cliente específico y verifica si el módulo de IA está hab
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** panel_cliente_envios.py
  Propósito: El código es un script de Python que carga un archivo de configuración, configura una conexión a una base de datos Supabase y define un Blueprint de Flask para manejar las rutas relacionadas con los envíos de un cliente. El Blueprint incluye una función que permite visualizar los envíos programados para un cliente y programar nuevos envíos. Además, se verifica si el usuario está autenticado antes de acceder a la funcionalidad de envi
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** panel_cliente_ia.py
  Propósito: El código es un script de Python que carga un archivo llamado "panel_cliente_ia.py" y configura un Blueprint de Flask llamado "panel_cliente_ia_bp". Define una ruta "/panel_cliente/ia/" que maneja tanto peticiones GET como POST. En la función "panel_ia", verifica si hay un usuario en sesión y redirige a la página de inicio de sesión si no lo hay. Luego, carga la configuración de un
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** panel_cliente_respuestas.py
  Propósito: El código es un script de Python que carga un archivo llamado "panel_cliente_respuestas.py" y configura un Blueprint en Flask para manejar respuestas de un panel de cliente. Utiliza la librería Supabase para interactuar con una base de datos y almacenar información sobre palabras clave y respuestas. El Blueprint define una ruta para manejar solicitudes GET y POST en "/panel_cliente/respuestas/". La función `panel_respuestas` verifica si
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** respuestas.py
  Propósito: Este código es un script en Python que utiliza el framework Flask para crear una aplicación web. Su propósito es interactuar con una base de datos en Supabase para cargar y guardar respuestas de un bot, así como también categorías relacionadas. El código define un Blueprint llamado 'respuestas' que contiene dos rutas: '/respuestas' para mostrar las respuestas almacenadas y '/guardar' para guardar nuevas respuestas en la base de datos. Además, se def
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: respuestas.html, editar.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, utils.config

Base de Datos

- **Archivo:** contactos.py
  Propósito: El código proporcionado es un script Python que utiliza el framework Flask para crear un Blueprint llamado 'contactos'. Este Blueprint contiene varias funciones que interactúan con una base de datos de Supabase para manejar información de contactos y conversaciones. Las funciones incluidas en el código son: 1. `leer_historial(telefono)`: Lee el historial de conversaciones de un contacto específico. 2. `actualizar_contacto(telefono, data)`:
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: panel_cliente_contactos.html, editar_contacto.html
  Tablas y columnas:
  Archivos .py usados: flask, datetime, googleapiclient.discovery, google.oauth2, supabase, dotenv
- **Archivo:** error_panel.py
  Propósito: Este código utiliza Flask para crear un Blueprint llamado "panel_errores" que maneja la visualización y limpieza de errores almacenados en una tabla llamada "logs_errores" en Supabase, un servicio de base de datos en la nube. La función `ver_errores()` muestra los errores almacenados en la tabla, mientras que `limpiar_errores()` elimina todos los errores de la tabla. Además, hay una función auxiliar
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: panel_errores.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** panel_chat.py
  Propósito: El código es un script de Python que carga un panel de chat para una aplicación web utilizando Flask. - Se importan diferentes módulos y librerías necesarios para la funcionalidad del panel de chat. - Se configura una conexión con una base de datos en Supabase y se verifica si las credenciales están configuradas correctamente. - Se establece una clave de API de OpenAI. - Se define una función `leer_contactos()` que lee los contact
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: panel_chat.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, clientes.aura.utils.normalizador
- **Archivo:** utils.py
  Propósito: El código proporcionado es un script de Python que interactúa con una base de datos en Supabase utilizando su API. El script incluye funciones para cargar datos desde un archivo JSON, guardar datos en un archivo JSON, cargar datos de una tabla en Supabase y guardar datos en una tabla de Supabase. El código comienza importando los módulos necesarios, como `json`, `os`, `re`, `datetime`, `supabase` y `dotenv`.
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: datetime, supabase, dotenv
- **Archivo:** whatsapp.py
  Propósito: Este código define un Blueprint en Flask llamado 'whatsapp', el cual maneja las interacciones con un webhook de WhatsApp. La función `obtener_contacto` busca un contacto en una base de datos utilizando el número de teléfono normalizado, y si no existe, crea un nuevo contacto con cierta información predeterminada. La función `actualizar_contacto` actualiza la información de un contacto existente en la base de datos. Ambas funciones interactúan con una base
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno

Tablas y columnas:
Archivos .py usados: flask, twilio.twiml.messaging_response, twilio.rest, utils.config, utils.historial, utils.normalizador, datetime, supabase, dotenv

Etiquetas

- **Archivo:** etiquetas.py
  Propósito: Este código es un fragmento de una aplicación web desarrollada con Flask. Su propósito es gestionar la visualización y creación de etiquetas para los clientes en una base de datos utilizando Supabase como servicio de almacenamiento. El código comienza importando los módulos necesarios, configurando la conexión con Supabase a través de las variables de entorno `SUPABASE_URL` y `SUPABASE_KEY`, y definiendo un Blueprint llamado `et
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, clientes.aura.utils.config
- **Archivo:** services.py
  Propósito: Este código es parte de una aplicación web desarrollada con Flask que interactúa con una base de datos de Supabase para gestionar contactos y etiquetas. Primero, se configura la conexión con Supabase utilizando las credenciales proporcionadas en variables de entorno. Luego se definen funciones para obtener la fecha y hora actual, cargar etiquetas desde la base de datos y agregar un contacto. La función `cargar_etiquetas()` consulta la tabla 'et
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: agregar_contacto.html, editar_contacto.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, .utils, datetime

Otros

- **Archivo:** registro_login.py
  Propósito: Este código tiene como propósito registrar un blueprint de login en una aplicación Flask. Primero intenta importar el blueprint de login desde el módulo `clientes.aura.auth.login`, luego lo registra en la aplicación utilizando `app.register_blueprint(login_bp)`. Si el registro es exitoso, imprime un mensaje indicando que el blueprint de login ha sido registrado correctamente. En caso de que ocurra algún error durante el proceso de registro, imprime un mensaje de error junto con la
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: clientes.aura.auth.login
- **Archivo:** webhook.py
  Propósito: Este código define un webhook utilizando Flask que recibe mensajes POST. El objetivo principal del código es procesar mensajes recibidos de Twilio, guardarlos en un historial, procesar la respuesta correspondiente y guardarla también en el historial. Además, se normaliza el número de teléfono del remitente y se manejan posibles errores durante el proceso. En resumen, el código se encarga de la comunicación entre un bot (en este caso "aura") y los
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, clientes.aura.handlers.process_message, clientes.aura.utils.historial, clientes.aura.utils.normalizador, datetime