Reporte del Proyecto

Rutas

- **Archivo:** admin_dashboard.py
  Propósito: Este código define rutas y funciones para un panel de administración en una aplicación web utilizando Flask. La función `dashboard_admin` verifica si un usuario está autenticado y si es un administrador. Luego, obtiene y muestra la cantidad de registros en las tablas "configuracion_bot" y "logs_errores" de una base de datos Supabase. Si hay errores al realizar estas consultas, se manejan e imprimen mensajes de error. Además, red
  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 tiene como propósito cargar y editar la configuración de un bot llamado "Nora" a través de una interfaz de administración. Utiliza Flask para crear un Blueprint llamado "admin_nora" que define una ruta para editar la configuración de una Nora específica. La configuración se carga desde una base de datos Supabase y se actualiza con los nuevos valores proporcionados a través de un formulario. Además, se valida que se ingresen correctamente el nombre
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, datetime
- **Archivo:** admin_noras.py
  Propósito: Este código tiene como propósito cargar un módulo llamado "admin_noras.py" correctamente. Luego, configura la conexión a una base de datos Supabase, define una ruta para acceder a una vista de administración de Noras y realiza consultas a la base de datos para obtener información sobre las Noras y los tickets pendientes. Finalmente, procesa esta información y la muestra en una plantilla HTML para la vista de administración. Además, maneja
  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 en Python que carga un dashboard para un usuario llamado Nora. Utiliza Flask para crear rutas web y acceder a datos almacenados en una base de datos Supabase. El script carga la configuración, contactos y respuestas asociadas a Nora desde la base de datos Supabase y luego imprime mensajes en la consola para indicar si la carga de datos fue exitosa o si hubo algún error.
  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: El código proporciona una funcionalidad para enviar mensajes de texto a través de WhatsApp utilizando la API de Twilio. Primero, se define una ruta en el blueprint `api_mensajes` para manejar la solicitud de enviar un mensaje.

Luego, se implementa la función `enviar_mensaje_por_twilio` que se encarga de enviar el mensaje utilizando las credenciales de Twilio configuradas en el entorno. El mensaje se envía a través de la API
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:** chat_data.py
  Propósito: Este código es una API construida con Flask que proporciona un endpoint para consultar el historial de conversaciones de un cliente a partir de su número de teléfono. El código importa las librerías necesarias, configura la conexión con Supabase (una base de datos como servicio), define un Blueprint llamado 'chat_data_aura' y una ruta '/chat/' que recibe como parámetro el número de teléfono del cliente. Dentro
  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 define un Blueprint en Flask llamado `debug_bp` que contiene varias rutas para propósitos de depuración y verificación en una aplicación web. Algunas de las funcionalidades proporcionadas por estas rutas incluyen: 1. `debug_verificacion`: Retorna el resultado de una función `revisar_todo()` en formato de texto preformateado. 2. `debug_verificacion_panel`: Renderiza un template HTML llamado "debug_verificacion.html".
  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 para mostrar información sobre variables de entorno críticas. Primero, se definen las claves críticas que se deben buscar en las variables de entorno. Luego, se recorre cada clave para verificar si existe en las variables de entorno y se genera un diccionario con el resultado de la verificación. Además, se obtiene y divide la variable "ADMIN_EMAILS" si está presente en las variables de ent
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: debug_env.html
  Tablas y columnas:
  Archivos .py usados: flask
- **Archivo:** debug_google.py
  Propósito: El código proporcionado es un script de Python que define una ruta de un Blueprint de Flask para verificar la configuración de autenticación con Google. El propósito principal es comprobar si se han configurado correctamente las credenciales de Google OAuth2 para permitir la autenticación de los usuarios a través de Google. El código verifica la presencia de las variables de entorno GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET y GOOGLE_REDIRECT_URI, y muestra mensajes de error si alguna de
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, requests_oauthlib, dotenv
- **Archivo:** debug_login.py
  Propósito: El propósito de este código es definir una ruta en una aplicación Flask para obtener información sobre el

estado de inicio de sesión de un usuario. La ruta "/debug/login_info" devuelve un JSON que indica si un usuario está autenticado o no, y proporciona detalles como el nombre, el email y si el usuario es un administrador. Si no hay ningún usuario autenticado, se devuelve un mensaje indicando que ningún usuario está logueado actualmente.
Rutas: Ninguna
Archivos CSS: Ninguno
Archivos HTML: Ninguno
Tablas y columnas:
Archivos .py usados: flask

- **Archivo:** debug_routes.py
  Propósito: El código proporciona una ruta de depuración en una aplicación Flask que muestra las rutas registradas en la aplicación junto con sus endpoints y métodos HTTP asociados. Al acceder a la ruta "/debug/rutas" mediante un método GET, se generará una página HTML que lista todas las rutas, sus endpoints y los métodos permitidos para cada una de ellas. La información se obtiene a través de la iteración de las reglas de URL registradas en la aplicación
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask
- **Archivo:** debug_test_temp.py
  Propósito: El código define un blueprint en Flask llamado `debug_test_temp_bp` con la ruta `/debug/test_temp`. Cuando se accede a esta ruta, se devuelve un objeto JSON con la clave "ok" establecida en True y un mensaje indicando que el blueprint está funcionando desde rutas. Además, se imprime un mensaje indicando que el archivo `debug_test_temp.py` se ha cargado correctamente.
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask
- **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 objetivo principal es interactuar con una base de datos en Supabase para manejar contactos y envíos programados. El código carga las credenciales de Supabase desde un archivo .env, establece la conexión con Supabase, define rutas para mostrar una interfaz web de envíos programados, obtener una lista de contactos, obtener una
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: panel_envios_programados.html
  Tablas y columnas:
  Archivos .py usados: flask, datetime, supabase, dotenv
- **Archivo:** error_panel.py
  Propósito: Este código es un conjunto de rutas y funciones relacionadas con la gestión de errores en una aplicación web utilizando Flask y Supabase como base de datos. El código incluye la configuración de Supabase utilizando variables de entorno, la creación de un Blueprint llamado "panel_errores" que define dos rutas: 1. "/panel/errores" que muestra los errores almacenados en la tabla "logs_errores" de Supabase.
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: panel_errores.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** main.py
  Propósito: Este código es una aplicación web utilizando Flask que interactúa con una base de datos en Supabase. El código define rutas para la autenticación de usuarios, administración de configuración y actualización de datos en la

base de datos. Algunos puntos clave son: - Define un Blueprint llamado `main_bp` para agrupar rutas relacionadas. - Implementa un middleware `login_requerido` que redirige a la página de inicio de sesión si el
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: Este código define un Blueprint de Flask llamado 'api_mensajes' que maneja una ruta para enviar mensajes a través de la API. La función 'enviar_mensaje_api' recibe un número de teléfono, un mensaje de texto y un archivo adjunto, normaliza el número, guarda el mensaje en un historial, lo envía por Twilio y emite un evento a través de Socket.IO. Además, hay una función 'enviar_mensaje_por_twilio'
  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: Este código es un archivo de Python que define rutas para un panel de cliente en una aplicación web utilizando Flask. - Primero, se configura la conexión a una base de datos Supabase. - Luego, se definen dos rutas: 1. La primera ruta es para acceder al panel del cliente, donde se verifica si el usuario está autenticado y se muestra una lista de módulos disponibles. 2. La segunda ruta es para acc
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** panel_cliente_contactos.py
  Propósito: El propósito de este código es cargar una página web relacionada con los contactos de un cliente en un panel de administración. Utiliza Flask para crear una ruta que maneja las solicitudes GET y POST para mostrar y manejar los contactos respectivamente. Además, se configura una conexión con Supabase, un servicio de base de datos, para consultar la información de configuración del bot y los contactos del cliente en cuestión. También se verifica si el usuario está
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** panel_cliente_respuestas.py
  Propósito: Este código es un script de Python que forma parte de una aplicación web utilizando Flask. Su propósito es manejar respuestas de un bot en una base de datos Supabase. El código carga las dependencias necesarias, como Flask, Supabase, dotenv, y otras, configura la conexión con Supabase utilizando las credenciales obtenidas del archivo .env, y define un Blueprint llamado "panel_cliente_respuestas" que incluye una ruta para manejar resp
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** respuestas.py
  Propósito: Este código define un Blueprint en Flask llamado `respuestas_bp` que maneja rutas relacionadas con respuestas de un bot. En el código se definen dos funciones para cargar respuestas y categorías desde una base de

datos utilizando Supabase. La función `mostrar_respuestas()` responde a la ruta '/respuestas' y carga las respuestas desde la base de datos para mostrarlas en una plantilla HTML llamada 'respuestas.html'. La función `guard
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:** categorias.py
  Propósito: Este código define un Blueprint en Flask llamado 'categorias' que maneja la gestión de categorías en una aplicación web. Utiliza la librería Supabase para interactuar con una base de datos. Las funciones definidas en el código son: - `mostrar_categorias`: Muestra todas las categorías existentes en la base de datos. - `agregar_categoria`: Agrega una nueva categoría a la base de datos. - `eliminar_categoria`: Elimina
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: categorias.html, editar_categoria.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, utils.config
- **Archivo:** contactos.py
  Propósito: Este código define un Blueprint de Flask llamado 'contactos' que contiene varias funciones para interactuar con una base de datos de Supabase. Las funciones definidas incluyen: - `leer_historial(telefono)`: Lee el historial de conversaciones para un número de teléfono específico desde la tabla 'historial_conversaciones' en Supabase. - `actualizar_contacto(telefono, data)`: Actualiza un contacto en la tabla 'contact
  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:** debug_verificar.py
  Propósito: Este código es un script de Python que carga un módulo Flask para verificar la configuración de diversas herramientas y servicios utilizados en una aplicación. Algunas de las verificaciones que realiza incluyen la conexión a Supabase, la configuración de OpenAI, OAuthLib y Google Login, así como la existencia de variables de entorno necesarias. Además, verifica la presencia de datos en ciertas tablas de la base de datos de Supabase. En res
  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:** panel_chat.py
  Propósito: Este código es un script de Python que carga un panel de chat para la aplicación Aura. En resumen, el código realiza las siguientes acciones: 1. Imprime un mensaje indicando que el archivo `panel_chat.py` se ha cargado correctamente. 2. Importa varios módulos necesarios para el funcionamiento del script. 3. Configura la conexión con la base de datos Supabase y la API de OpenAI. 4. Define una función `leer_contactos
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: panel_chat.html
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv, clientes.aura.utils.normalizador

- **Archivo:** panel_cliente_envios.py
  Propósito: El código es un script de Python que carga un módulo llamado "panel_cliente_envios.py" y configura un Blueprint de Flask llamado "panel_cliente_envios_bp". El propósito principal del código es permitir a los clientes programar y gestionar envíos a través de una interfaz web. Utiliza la librería Supabase para interactuar con una base de datos y almacenar la información de los envíos programados. La función
  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 forma parte de un proyecto web utilizando Flask. Su propósito es manejar la configuración de un bot de inteligencia artificial para clientes en un panel de control. El código carga la configuración del bot desde una base de datos Supabase, permite al usuario activar o desactivar la inteligencia artificial a través de una interfaz web, y actualiza la configuración en la base de datos. Además, requiere que el usuario est
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: flask, supabase, dotenv
- **Archivo:** utils.py
  Propósito: El código proporciona funciones para cargar y guardar datos en una base de datos de Supabase utilizando su API. 1. La función `cargar_json(archivo, default=None)` carga un archivo JSON y devuelve su contenido como un diccionario. Si no puede cargar el archivo, devuelve un diccionario vacío o un valor predeterminado especificado. 2. La función `guardar_json(archivo, data)` guarda datos en un archivo JSON. Crea el directorio
  Rutas: Ninguna
  Archivos CSS: Ninguno
  Archivos HTML: Ninguno
  Tablas y columnas:
  Archivos .py usados: datetime, supabase, dotenv
- **Archivo:** whatsapp.py
  Propósito: Este código implementa un webhook de WhatsApp utilizando Flask y Twilio para recibir y procesar mensajes de WhatsApp. El código incluye funciones para interactuar con la base de datos de Supabase para obtener y actualizar información de contactos. Además, se normalizan los números de teléfono antes de consultar la base de datos. También se manejan posibles errores durante la ejecución de las consultas a la base de datos.
  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

HTML

- **Archivo:** debug_openai.py
  Propósito: El código proporciona un Blueprint de Flask llamado `debug_openai_bp` que contiene dos funciones. La primera función, `test_openai()`, crea una solicitud a la API de OpenAI para obtener una respuesta a una pregunta específica proporcionada como parámetro en la URL. La respuesta se formatea y se devuelve como una página HTML. La segunda función, `verificar_openai()`, verifica la versión de la librería de OpenAI instalada y devuelve un
  Rutas: Ninguna
  Archivos CSS: Ninguno

Archivos HTML: Ninguno
Tablas y columnas:
Archivos .py usados: flask, dotenv

Etiquetas

- **Archivo:** etiquetas.py
  Propósito: Este código define una ruta en una aplicación Flask para gestionar etiquetas de clientes. La función `panel_etiquetas` se encarga de mostrar y agregar etiquetas para un cliente específico. Primero, verifica si el usuario está autenticado. Luego, obtiene las etiquetas del cliente desde una base de datos Supabase y las muestra en la página. Si se realiza una solicitud POST (como al enviar un formulario para agregar una nueva etiqueta), se procesa
  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 en Supabase. En resumen, el código incluye la configuración de Supabase, la definición de funciones para obtener la marca de tiempo actual, cargar etiquetas desde la base de datos, y agregar un nuevo contacto a la base de datos. La función `cargar_etiquetas()` busca y carga etiquetas desde la tabla 'etiquetas' en la base de
  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: El código tiene una función llamada `registrar_blueprints_login` que recibe un objeto `app`. Dentro de la función, intenta importar un blueprint de login desde el módulo `clientes.aura.auth.login`, registra ese blueprint en la aplicación `app` y luego imprime un mensaje indicando que el blueprint de login ha sido registrado con éxito. En caso de que ocurra alguna excepción durante el proceso, imprimirá un mensaje de error junto con la descri
  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 en Flask que recibe mensajes POST y procesa interacciones con un bot llamado "aura". Al recibir un mensaje del usuario a través de Twilio, normaliza el número del remitente, guarda el mensaje en un historial, procesa el mensaje para generar una respuesta del bot, guarda la respuesta en el historial y finalmente devuelve la respuesta generada con un código de estado HTTP 200 si todo se procesa correctamente. En caso de error,
  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