Diseño Completo de Endpoints de la API (RESTful)
La API del backend será el puente entre la aplicación de escritorio y la infraestructura en la nube, permitiendo la gestión de toda la información del consultorio. Se diseñará siguiendo los principios de RESTfulness para garantizar la coherencia, la escalabilidad y la facilidad de uso.
Para Gestión de Pacientes
El módulo de Gestión Integral del Expediente del Paciente es crucial para centralizar toda la información crítica y permitir un acceso inmediato y un seguimiento eficiente. Los siguientes endpoints facilitarán estas operaciones:

- 1. GET /patients
  - Método HTTP: GET
  - Ruta exacta: /api/v1/patients
  - Descripción: Permite obtener una lista de todos los pacientes registrados en el sistema. Soporta parámetros para búsqueda, filtrado, paginación y ordenamiento, facilitando la "Búsqueda Avanzada". Los filtros pueden incluir fecha, estado o tipo de consulta.
- 2. GET /patients/{id}
  - Método HTTP: GET
  - Ruta exacta: /api/v1/patients/{id} (donde {id} es el identificador único del paciente)
  - Descripción: Recupera los detalles completos de un paciente específico, incluyendo su perfil y la información general de su historial. Esto es útil para la función "Ver Detalles" o para abrir la ficha completa del paciente.
- 3. POST /patients
  - Método HTTP: POST
  - Ruta exacta: /api/v1/patients
  - Descripción: Crea un nuevo registro de paciente en el sistema con todos sus datos demográficos y de contacto.
- 4. PUT /patients/{id}
  - Método HTTP: PUT
  - Ruta exacta: /api/v1/patients/{id}
  - Descripción: Actualiza los datos de un paciente existente identificado por su ID. Permite modificar cualquier campo del perfil del paciente.
- 5. DELETE /patients/{id}
  - Método HTTP: DELETE
  - Ruta exacta: /api/v1/patients/{id}
  - Descripción: Realiza un "borrado suave" (soft delete) de un paciente. Esto significa que el registro no se elimina permanentemente de la base de datos, sino que se marca como inactivo o archivado (ej., is_deleted = true). El sistema está diseñado para que el registro se marque como inactivo, no se borre permanentemente.
- 6. POST /patients/{id}/archive
  - Método HTTP: POST
  - Ruta exacta: /api/v1/patients/{id}/archive
  - Descripción: Archiva un paciente específico por su ID. Esta acción complementa el borrado suave, marcando el paciente como archivado o inactivo. Esta funcionalidad puede aplicarse a múltiples pacientes seleccionados para acciones en lote.

Documento: Arq. Backend y APIs
Diseño Completo de Endpoints de la API (RESTful) (Continuación)
La API del backend continúa su diseño siguiendo los principios de RESTfulness, garantizando la coherencia, la escalabilidad y la facilidad de uso para todas las operaciones del "Ecosistema Digital Inteligente para Cirugía Especial". Los datos estructurados se almacenan en la Base de Datos Relacional (Cloud SQL), mientras que los archivos digitales residen en Cloud Storage, ambos con cifrado en reposo para asegurar la confidencialidad de la información del paciente. Todas las interacciones con la API se realizarán de forma segura a través de HTTPS (TLS 1.3).
Para Gestión de Historial Clínico
El registro cronológico y detallado del historial clínico es una funcionalidad central para el seguimiento eficiente del paciente. Los siguientes endpoints permiten la interacción con esta información:
- 1. GET /patients/{id}/history
  - Método HTTP: GET
  - Ruta exacta: /api/v1/patients/{id}/history (donde {id} es el identificador único del paciente)
  - Descripción: Permite obtener el historial clínico completo de un paciente específico, recuperando todas las entradas cronológicas de consultas, diagnósticos, notas de evolución, procedimientos y tratamientos asociados.
- 2. POST /patients/{id}/history
  - Método HTTP: POST
  - Ruta exacta: /api/v1/patients/{id}/history
  - Descripción: Permite agregar una nueva entrada al historial clínico de un paciente específico. Esta acción es el equivalente digital a registrar una nueva consulta o procedimiento, incluyendo campos como fecha de consulta, diagnóstico y notas.
- 3. PUT /history/{id}
  - Método HTTP: PUT
  - Ruta exacta: /api/v1/history/{id} (donde {id} es el identificador único de una entrada específica del historial)
  - Descripción: Actualiza los datos de una entrada existente en el historial clínico, permitiendo la corrección o modificación de detalles como diagnósticos o notas.
- 4. DELETE /history/{id}
  - Método HTTP: DELETE
  - Ruta exacta: /api/v1/history/{id}
  - Descripción: Realiza un "borrado suave" de una entrada específica del historial clínico por su ID. Esto significa que el registro se marca como inactivo o archivado en la base de datos, en lugar de ser eliminado permanentemente, preservando la integridad histórica de los datos.

Para Gestión de Citas
El módulo de agenda y citas es fundamental para optimizar la programación y el control del flujo de pacientes, reduciendo los procesos manuales. Los endpoints asociados son:
- 1. GET /appointments
  - Método HTTP: GET
  - Ruta exacta: /api/v1/appointments
  - Descripción: Obtiene la lista de todas las citas programadas en el consultorio. Soporta parámetros para filtrar por fecha (ej., "Citas del Día"), por paciente o por estado (ej., "Confirmada", "Cancelada", "Atendida"), así como paginación y ordenamiento.
- 2. GET /appointments/{id}
  - Método HTTP: GET
  - Ruta exacta: /api/v1/appointments/{id} (donde {id} es el identificador único de la cita)
  - Descripción: Recupera los detalles completos de una cita específica por su ID, incluyendo el paciente asociado, fecha, hora, tipo y estado.
- 3. POST /appointments
  - Método HTTP: POST
  - Ruta exacta: /api/v1/appointments
  - Descripción: Crea una nueva cita en el calendario del consultorio. Requiere la información necesaria como el paciente, la fecha, la hora y el tipo de cita.
- 4. PUT /appointments/{id}
  - Método HTTP: PUT
  - Ruta exacta: /api/v1/appointments/{id}

    ◦ Descripción: Actualiza los datos de una cita existente identificada por su ID. Permite modificar su fecha, hora, tipo, o cambiar su estado (ej., de "Programada" a "Confirmada" o "Cancelada").
• 5. DELETE /appointments/{id}
    ◦ Método HTTP: DELETE
    ◦ Ruta exacta: /api/v1/appointments/{id}
    ◦ Descripción: Realiza un "borrado suave" de una cita por su ID. El registro de la cita se marca como inactivo en la base de datos, lo que permite mantener un historial completo de las citas incluso si son canceladas o no se realizan.

Para Gestión Documental

La gestión documental avanzada centraliza y asegura el almacenamiento de todos los archivos digitales del paciente, eliminando la dispersión y mejorando la seguridad. Los archivos se almacenan en Cloud Storage con cifrado en reposo.

• 1. POST /patients/{id}/documents
    ◦ Método HTTP: POST
    ◦ Ruta exacta: /api/v1/patients/{id}/documents
    ◦ Descripción: Permite subir un nuevo archivo digital (PDF, imagen, video corto, etc.) y asociarlo directamente al expediente de un paciente específico. El archivo se cargará a Cloud Storage y sus metadatos se registrarán en la base de datos.
• 2. GET /patients/{id}/documents
    ◦ Método HTTP: GET
    ◦ Ruta exacta: /api/v1/patients/{id}/documents
    ◦ Descripción: Obtiene una lista de todos los documentos digitales (metadatos como nombre, tipo, URL) asociados a un paciente específico.
• 3. GET /documents/{id}
    ◦ Método HTTP: GET
    ◦ Ruta exacta: /api/v1/documents/{id} (donde {id} es el identificador único del documento)
    ◦ Descripción: Recupera un documento específico por su ID. Esta acción puede devolver una URL pre-firmada o un stream de datos que permita la descarga o visualización directa del archivo desde Cloud Storage.
• 4. DELETE /documents/{id}
    ◦ Método HTTP: DELETE
    ◦ Ruta exacta: /api/v1/documents/{id}
    ◦ Descripción: Realiza un "borrado suave" de un documento digital por su ID. Solo se marca el registro del documento como inactivo en la base de datos, sin eliminar el archivo físico de Cloud Storage de forma inmediata, manteniendo así un registro auditable.

Documento: Arq. Backend y APIs
Diseño Completo de Endpoints de la API (RESTful) (Continuación)
La definición de los endpoints RESTful es crucial para la interacción segura y eficiente entre el Dashboard de Escritorio (Núcleo Administrativo Central) y el backend en la nube (Google Cloud Platform). Estos endpoints facilitan la autenticación de usuarios, la gestión de permisos, el registro de actividades para auditoría y la integración de capacidades de Inteligencia Artificial para automatización y asistencia. Todas las comunicaciones se realizarán a través de HTTPS (TLS 1.3), garantizando el cifrado de datos en tránsito.
Para Autenticación y Autorización
La autenticación y la autorización son la base de la seguridad del "Ecosistema Digital Inteligente para Cirugía Especial", asegurando que solo el personal autorizado acceda a la información y funcionalidades.

- 1. POST /login
  - Método HTTP: POST
  - Ruta exacta: /api/v1/auth/login
  - Descripción: Este endpoint es el punto de entrada para que un usuario inicie sesión en el sistema. Recibe las credenciales (nombre de usuario y contraseña), las valida contra la base de datos, y si son correctas, devuelve un token de autenticación que será utilizado para futuras solicitudes. Es fundamental para aplicar el control de acceso basado en roles. La comunicación se realiza cifrada vía HTTPS (TLS 1.3).
- 2. Endpoints para Gestión de Usuarios
  - La gestión de usuarios es una funcionalidad clave del "Núcleo Administrativo Central", accesible para roles como "Administrador" y "Admin Principal".
  - GET /users
    - Método HTTP: GET
    - Ruta exacta: /api/v1/users
    - Descripción: Obtiene una lista de todos los usuarios registrados en el sistema. Este endpoint permite a los administradores visualizar el personal del consultorio y sus roles asociados.
  - GET /users/{id}
    - Método HTTP: GET
    - Ruta exacta: /api/v1/users/{id} (donde {id} es el identificador único del usuario)
    - Descripción: Recupera los detalles completos del perfil de un usuario específico, incluyendo su información de contacto y el rol asignado.
  - POST /users
    - Método HTTP: POST
    - Ruta exacta: /api/v1/users
    - Descripción: Permite crear una nueva cuenta de usuario en el sistema. Esta operación incluye la asignación de un rol inicial (ej., "Asistente", "Admin Secundario").
  - PUT /users/{id}
    - Método HTTP: PUT
    - Ruta exacta: /api/v1/users/{id}
    - Descripción: Actualiza los datos de un usuario existente, como su nombre, información de contacto o, fundamentalmente, su rol en el sistema.
  - DELETE /users/{id}
    - Método HTTP: DELETE
    - Ruta exacta: /api/v1/users/{id}
    - Descripción: Realiza un "borrado suave" de una cuenta de usuario por su ID. El registro del usuario se marca como inactivo en la base de datos, manteniendo la integridad histórica y facilitando posibles recuperaciones.
- 3. Endpoints para Gestión de Roles
  - El sistema cuenta con un robusto "Sistema de Roles", con roles como "Admin Principal", "Admin Secundario", "Asistente", "Invitado (Solo Lectura)". La información de roles se almacena en la tabla ROLES.
  - GET /roles
    - Método HTTP: GET
    - Ruta exacta: /api/v1/roles
    - Descripción: Obtiene una lista de todos los roles de usuario definidos en el sistema, junto con sus nombres y las capacidades generales asociadas.
  - GET /roles/{id}

        - Método HTTP: GET
        - Ruta exacta: /api/v1/roles/{id} (donde {id} es el identificador único del rol)
        - Descripción: Recupera los detalles específicos y el conjunto de permisos detallados asociados a un rol particular por su ID.
    - POST /roles
        - Método HTTP: POST
        - Ruta exacta: /api/v1/roles
        - Descripción: Permite la creación de una nueva definición de rol en el sistema, especificando su nombre y los permisos que este tendrá sobre las diferentes funcionalidades y datos.
    - PUT /roles/{id}
        - Método HTTP: PUT
        - Ruta exacta: /api/v1/roles/{id}
        - Descripción: Actualiza el nombre o los permisos de un rol existente. Esta es una operación crítica que debe ser manejada con extrema precaución.
    - DELETE /roles/{id}
        - Método HTTP: DELETE
        - Ruta exacta: /api/v1/roles/{id}
        - Descripción: Realiza un "borrado suave" de un rol, marcándolo como inactivo. Los roles inactivos no pueden ser asignados a nuevos usuarios, pero su registro se mantiene para auditoría y referencias históricas.
- 4. Verificación de Permisos por Rol
    - Descripción: La verificación de permisos basados en el rol del usuario es una lógica interna del backend que se ejecuta en el Servidor de Aplicación. Después de que un usuario es autenticado y su rol se consulta de la base de datos ROLES, el backend determina si el usuario tiene permiso para acceder a una funcionalidad o dato específico antes de procesar la solicitud. No se expone como un endpoint RESTful dedicado.

Para Logs de Auditoría

El "Servicio de Logs de Auditoría (Cloud Logging / Audit Logs)" es fundamental para la seguridad, registrando cronológicamente toda actividad y cambios en el sistema. Estos logs son inmutables (Write Once, Read Many - WORM).

- 1. POST /logs
    - Método HTTP: POST
    - Ruta exacta: /api/v1/logs
    - Descripción: Este endpoint es utilizado internamente por el backend para registrar de forma automática y asíncrona todas las acciones clave realizadas en el sistema. Esto incluye intentos de login (exitosos o fallidos), modificaciones de datos de pacientes, creación/actualización de citas, subida de documentos, y cualquier otra operación crítica que requiera un rastro auditable.
- 2. GET /logs
    - Método HTTP: GET
    - Ruta exacta: /api/v1/logs
    - Descripción: Obtiene un listado cronológico de los registros de auditoría del sistema. Este endpoint es accesible solo para roles con permisos elevados (como el "Admin Principal"), permitiéndoles revisar el historial de actividades, monitorear la seguridad y rastrear operaciones específicas.

Para Integración con IA

La "Inteligencia y Automatización Integrada" utiliza el modelo Gemma 3 (o similar a Gemini) para potenciar el flujo de trabajo del consultorio. La IA se integra directamente en el backend mediante APIs.

- 1. POST /ai/generate-email
    - Método HTTP: POST
    - Ruta exacta: /api/v1/ai/generate-email
    - Descripción: Envía una solicitud al modelo de IA para generar un borrador de correo electrónico. Se proporcionarían parámetros como el propósito del correo, el destinatario y puntos clave a incluir, y la IA devolvería el texto del borrador.
- 2. POST /ai/summarize
    - Método HTTP: POST
    - Ruta exacta: /api/v1/ai/summarize
    - Descripción: Recibe un bloque de texto (ej., notas de consulta, informes) y utiliza el modelo de IA para generar un resumen conciso de dicho contenido.
- 3. POST /ai/chat
    - Método HTTP: POST

    - Ruta exacta: /api/v1/ai/chat
    - Descripción: Permite enviar mensajes de texto al chat interno del sistema, donde el modelo de IA procesa la entrada y devuelve una respuesta textual. Es utilizado para consultas rápidas o para asistir en la redacción de informes. Cabe destacar que este chat funciona únicamente con texto y no genera archivos.
- 4. POST /ai/generate-template
    - Método HTTP: POST
    - Ruta exacta: /api/v1/ai/generate-template
    - Descripción: Envía una solicitud a la IA para generar diversas plantillas, como hojas membretadas, basándose en la información o el formato requerido.
- 5. POST /ai/create-reminder
    - Método HTTP: POST
    - Ruta exacta: /api/v1/ai/create-reminder
    - Descripción: Utiliza la IA para interpretar una solicitud de usuario y generar un recordatorio dentro del sistema, posiblemente con detalles como fecha, hora y descripción.

Documento: Arq. Backend y APIs
Especificación de Solicitudes y Respuestas
Para asegurar la interoperabilidad y una comunicación fluida y predecible entre el Dashboard de Escritorio (frontend) y el Servidor de Aplicación (backend), es fundamental definir contratos claros para las APIs. Estos contratos especifican la estructura JSON esperada tanto en las solicitudes (Request Body) enviadas al backend como en las respuestas (Response Body) recibidas. Todas las comunicaciones de datos sensibles se realizarán de forma segura a través de HTTPS (TLS 1.3).
1. Estructura JSON para Solicitudes (Request Body)
Las solicitudes al backend generalmente contendrán objetos JSON con los datos necesarios para realizar una operación específica (creación, actualización, etc.). Los ejemplos a continuación ilustran estructuras genéricas:
• Para Autenticación (Ej: POST /api/v1/auth/login)
    ◦ Método: POST
    ◦ Descripción: Envía las credenciales del usuario para iniciar sesión.
    ◦ Estructura JSON:
    ◦ Consideraciones: Las contraseñas se envían como texto plano en el cuerpo de la solicitud, pero la seguridad está garantizada por el cifrado TLS 1.3 en tránsito.
• Para Creación de Recursos (Ej: POST /api/v1/patients)
    ◦ Método: POST
    ◦ Descripción: Crea un nuevo perfil de paciente en el sistema.
    ◦ Estructura JSON:
    ◦ Nota: datos_contacto se almacenaría como JSON en la base de datos.
• Para Actualización de Recursos (Ej: PUT /api/v1/patients/{id})
    ◦ Método: PUT
    ◦ Descripción: Actualiza los datos de un paciente existente.
    ◦ Estructura JSON:
    ◦ Consideraciones: Solo los campos que necesitan ser modificados deben ser incluidos en la solicitud.
• Para Agendamiento de Citas (Ej: POST /api/v1/appointments)
    ◦ Método: POST
    ◦ Descripción: Permite programar una nueva cita.
    ◦ Estructura JSON:
• Para Interacción con IA (Ej: POST /api/v1/ai/generate-email)
    ◦ Método: POST
    ◦ Descripción: Envía datos para que el modelo de IA (Gemma 3 o similar a Gemini) genere un borrador de correo electrónico.
    ◦ Estructura JSON:
    ◦ Nota: Otros endpoints de IA como /ai/summarize o /ai/chat esperarían campos como text_to_summarize o message respectivamente.
2. Estructura JSON para Respuestas (Response Body)
Las respuestas del backend indicarán el resultado de la operación solicitada. Las respuestas de éxito suelen contener los datos del recurso solicitado o modificado.
• Para Recuperación de un Recurso Único (Ej: GET /api/v1/patients/{id})
    ◦ Método: GET
    ◦ Descripción: Devuelve los detalles completos de un paciente específico.
    ◦ Estructura JSON (HTTP 200 OK):
• Para Listado de Recursos con Paginación (Ej: GET /api/v1/patients)
    ◦ Método: GET
    ◦ Descripción: Devuelve una lista de pacientes, con soporte para paginación.
    ◦ Estructura JSON (HTTP 200 OK):
• Para Creación Exitosa (Ej: POST /api/v1/patients)
    ◦ Método: POST
    ◦ Descripción: Confirma la creación exitosa del recurso, incluyendo su identificador único.
    ◦ Estructura JSON (HTTP 201 Created):
• Para Login Exitoso (Ej: POST /api/v1/auth/login)
    ◦ Método: POST
    ◦ Descripción: Devuelve un token de autenticación para futuras solicitudes, junto con información básica del usuario y su rol.
    ◦ Estructura JSON (HTTP 200 OK):
• Para Respuesta de IA (Ej: POST /api/v1/ai/generate-email)

    - Método: POST
    - Descripción: Devuelve el texto generado por el modelo de IA.
    - Estructura JSON (HTTP 200 OK):

3. Manejo de Archivos Grandes (Cloud Storage y URLs)

Es importante destacar que los archivos grandes, como PDFs de historiales, imágenes (ej. RX) y videos cortos, no se enviarán directamente en el cuerpo de las solicitudes o respuestas JSON. En su lugar, se utilizará Google Cloud Storage (equivalente a S3/Blob) para el almacenamiento seguro de estos archivos. Los datos sensibles almacenados en Cloud Storage estarán cifrados en reposo.

- Para Subida de Archivos: La aplicación de escritorio primero enviará el archivo directamente a Cloud Storage. Una vez que el archivo esté en la nube, el backend recibirá una URL (o un identificador) del archivo, que luego se almacenará en la base de datos relacional (Cloud SQL) junto con los metadatos relevantes del paciente o del historial clínico.
- Para Acceso a Archivos: Cuando se necesite visualizar un archivo, el backend proporcionará una URL firmada o un enlace seguro de Cloud Storage en la respuesta JSON. El frontend utilizará esta URL para acceder y mostrar el archivo directamente desde Cloud Storage, evitando cargar el backend con transferencias de grandes volúmenes de datos.

Manejo de Errores

Un manejo de errores estandarizado es vital para la robustez y la facilidad de depuración de la API, proporcionando retroalimentación clara tanto al frontend como a los desarrolladores.

1. Códigos de Estado HTTP Estándar

El backend utilizará códigos de estado HTTP estándar para indicar el resultado de cada solicitud:

- 200 OK: La solicitud fue exitosa. Se utiliza para operaciones GET que devuelven datos, y para PUT/DELETE que resultaron en una modificación/eliminación exitosa.
- 201 Created: La solicitud POST fue exitosa y resultó en la creación de un nuevo recurso. La respuesta incluirá el recurso recién creado o su identificador.
- 400 Bad Request: La solicitud del cliente es incorrecta, malformada o contiene datos inválidos. Esto incluye errores de validación de entrada.
- 401 Unauthorized: La solicitud no ha sido aplicada porque le falta credenciales de autenticación válidas para el recurso solicitado. Esto ocurre si el token de autenticación está ausente, es inválido o ha expirado. También se usa cuando las credenciales de login son incorrectas.
- 403 Forbidden: La solicitud es válida, pero el usuario autenticado no tiene los permisos necesarios para acceder al recurso o realizar la acción solicitada. El control de acceso basado en roles es fundamental aquí.
- 404 Not Found: El recurso solicitado no existe en el servidor. Esto se aplica si se busca un ID de paciente que no existe.
- 500 Internal Server Error: Un error genérico del servidor que impide completar la solicitud. Esto indica un problema inesperado en el backend, no relacionado directamente con la solicitud del cliente.

2. Estructura JSON Estándar para Mensajes de Error

Todos los errores, especialmente los de tipo 4xx y 5xx, se devolverán con una estructura JSON estandarizada para facilitar su procesamiento por parte del frontend:

```
{
  "error": "Mensaje conciso del error",
  "code": 400, // Código de estado HTTP
  "details": "Mensaje detallado para el desarrollador o información de validación"
}
```

- error: Una descripción breve y legible por el usuario del problema.
- code: El código de estado HTTP relevante.
- details: Información adicional que puede ser útil para la depuración o para mostrar validaciones específicas al usuario (ej. lista de campos inválidos).

3. Ejemplos de Uso de Errores

- Escenario: Solicitud con Campos Obligatorios Vacíos (400 Bad Request)
    - Descripción: El usuario intenta crear un paciente sin proporcionar el nombre.
    - Respuesta JSON:
    - Citas:. Mensajes como "Campo 'Nombre' requerido" o "Formato de email inválido".
- Escenario: Credenciales de Login Incorrectas (401 Unauthorized)
    - Descripción: Un usuario intenta iniciar sesión con un nombre de usuario o contraseña

incorrectos.
    ◦ Respuesta JSON:
    ◦ Citas:.
• Escenario: Acceso Denegado por Permisos (403 Forbidden)
    ◦ Descripción: Un usuario con rol de "Asistente" intenta acceder a la funcionalidad de "Gestionar Usuarios".
    ◦ Respuesta JSON:
    ◦ Citas: El sistema impone un control estricto de acceso basado en roles.
• Escenario: Recurso No Encontrado (404 Not Found)
    ◦ Descripción: El frontend solicita los detalles de un paciente con un id que no existe en la base de datos.
    ◦ Respuesta JSON:
    ◦ Citas:. Mensajes como "No se encontraron pacientes".
• Escenario: Error Inesperado del Servidor (500 Internal Server Error)
    ◦ Descripción: Un fallo inesperado ocurre en la lógica del backend (ej. un problema con la base de datos, un error en el código no manejado).
    ◦ Respuesta JSON:
    ◦ Citas:. Mensaje sobre "problemas de conexión o indisponibilidad del servicio".

4. Registro de Errores (Logs de Auditoría)

Es crucial que todos los errores, especialmente los errores 4xx y 5xx, sean registrados en el "Servicio de Logs de Auditoría (Cloud Logging / Audit Logs)". Estos logs son inmutables (Write Once, Read Many - WORM), lo que garantiza su integridad y son vitales para la detección de anomalías, la respuesta a incidentes y el cumplimiento normativo. Esto permite un monitoreo proactivo de la seguridad y el rendimiento del sistema.

Implementación de Autenticación y Autorización
Para asegurar la seguridad y el control de acceso en la plataforma de Remex 8, se definirá una estrategia robusta de autenticación y autorización. Dado el modelo de negocio multinivel de Remex 8, que implica la gestión de socios y distribuidores, es fundamental proteger la información y las funcionalidades de la API.

- Estrategia de Autenticación de Usuarios:
  - La autenticación se implementará utilizando tokens web JSON (JWT) [información no proveniente de las fuentes]. Después de que un usuario (cliente o distribuidor) inicie sesión exitosamente proporcionando sus credenciales (nombre de usuario/correo electrónico y contraseña), el backend generará un token JWT. Este token encapsulará información relevante sobre el usuario, como su ID y roles, y será firmado digitalmente para garantizar su integridad y autenticidad [información no proveniente de las fuentes].
  - Este token será enviado al cliente (aplicación web o móvil) y deberá ser incluido en las cabeceras de cada solicitud subsiguiente a la API. Esto permitirá al servidor verificar la identidad del usuario sin necesidad de consultar una base de datos en cada petición, optimizando el rendimiento [información no proveniente de las fuentes].
  - La validez de los tokens se gestionará estableciendo un tiempo de expiración limitado, tras el cual el usuario deberá refrescar su sesión o volver a autenticarse [información no proveniente de las fuentes]. Se podría implementar un mecanismo de refresh token para mejorar la experiencia de usuario, permitiendo renovar el token de acceso sin requerir la reintroducción de credenciales, siempre que el refresh token sea válido [información no proveniente de las fuentes].
- Implementación de Autorización (Control de Acceso Basado en Roles - RBAC):
  - La autorización se basará en un modelo de control de acceso basado en roles (RBAC) [información no proveniente de las fuentes]. Cada usuario en el sistema de Remex 8 (por ejemplo, clientes, distribuidores, administradores) tendrá uno o más roles asignados, que definirán sus permisos dentro de la plataforma [información no proveniente de las fuentes].
  - Antes de procesar cualquier solicitud a un endpoint de la API, el backend verificará el token JWT para extraer los roles del usuario. Luego, se comprobará si el rol del usuario tiene los permisos necesarios para acceder a la funcionalidad o a los datos específicos solicitados [información no proveniente de las fuentes]. Por ejemplo, un distribuidor tendrá acceso a funcionalidades relacionadas con su red y ventas, mientras que un cliente solo podrá acceder a información de productos y realizar compras.
  - Esto garantiza que solo los usuarios con los permisos adecuados puedan realizar ciertas acciones (ej., gestionar la red de afiliados, ver reportes de comisiones, acceder a información de precios de distribuidor) [información no proveniente de las fuentes].

Tecnología del Backend
La elección de la tecnología para el backend es crucial para la escalabilidad, mantenibilidad y eficiencia de la plataforma digital de Remex 8. Considerando que dogma.black se especializa en desarrollo de software y programación, y que se ha mencionado el uso potencial de Modelos de Lenguaje (LLMs) y APIs para automatización, se sugieren las siguientes opciones, las cuales no están explícitamente detalladas en las fuentes de Remex 8:

- Elección del Lenguaje de Programación:
  - Se recomienda Python o Node.js [información no proveniente de las fuentes].
    - Python: Es una excelente opción por su legibilidad, su vasta comunidad y su robusto ecosistema de librerías, especialmente útil si en el futuro se planean integraciones más profundas con análisis de datos o inteligencia artificial (dado el contexto de LLMs mencionados por dogma.black) [4, información no proveniente de las fuentes]. Su versatilidad lo hace apto para el desarrollo rápido y para manejar diversas tareas del backend.
    - Node.js: Ideal para aplicaciones en tiempo real y microservicios, Node.js permite usar JavaScript tanto en el frontend como en el backend, lo que puede agilizar el desarrollo si el equipo ya está familiarizado con JavaScript. Su modelo de E/S no bloqueante es eficiente para aplicaciones con muchas conexiones concurrentes, como una plataforma de redes sociales con alta interacción [1, 2, 4, información no proveniente de las fuentes].
- Elección del Framework de Backend y Justificación:
  - Si se elige Python:
    - Django: Es un framework completo y robusto que incluye muchas funcionalidades "listas para usar", como un ORM (Object-Relational Mapper), un panel de administración y

un sistema de autenticación, lo que acelera el desarrollo de aplicaciones complejas y seguras [información no proveniente de las fuentes]. Sería ideal para una plataforma que gestiona tanto productos como una red de distribuidores con diferentes roles y datos.

        - Flask: Si se prefiere una solución más ligera y modular, Flask es una excelente alternativa. Ofrece mayor flexibilidad y control sobre los componentes a utilizar, siendo ideal para construir APIs RESTful y microservicios, permitiendo escalar funcionalidades específicas según la demanda de Remex 8 [información no proveniente de las fuentes].
    - Si se elige Node.js:
        - Express.js: Es el framework más popular y flexible para Node.js, ideal para construir APIs RESTful de manera rápida y eficiente [información no proveniente de las fuentes]. Su naturaleza unopinionated permite diseñar la arquitectura del backend con gran libertad, integrándose fácilmente con bases de datos y otros servicios necesarios para una plataforma de marketing y ventas multinivel [1, 2, 3, información no proveniente de las fuentes].

Estas elecciones se alinean con la necesidad de construir una infraestructura digital sólida que soporte el crecimiento de Remex 8 y las capacidades de automatización exploradas por dogma.black.

Configuración de Servicios de GCP para el Backend
La infraestructura del backend de Remex 8 se apoyará en Google Cloud Platform (GCP) para proveer la escalabilidad, resiliencia y seguridad necesarias. A continuación, se detalla la configuración de los servicios clave:

- Servidor de Aplicación: Cloud Run
  - Para el despliegue del servidor de aplicación, se recomienda el uso de Cloud Run [información no proveniente de las fuentes]. Esta elección se alinea con la flexibilidad y la capacidad de construir APIs RESTful y microservicios mencionadas en la sección de "Tecnología del Backend" [información no proveniente de las fuentes].
  - Propósito en la Arquitectura: Cloud Run permitirá desplegar los servicios del backend como contenedores sin servidor, lo que significa que la infraestructura subyacente será gestionada automáticamente por GCP [información no proveniente de las fuentes]. Esto facilitará la escalabilidad automática basada en la demanda de tráfico, desde cero hasta miles de solicitudes concurrentes, y optimizará los costos al pagar solo por los recursos consumidos [información no proveniente de las fuentes]. Es ideal para alojar la lógica de negocio de la API, procesar las solicitudes de autenticación/autorización y servir los datos a las aplicaciones cliente.
- API Gateway / Load Balancer: Cloud Load Balancing con integración de Cloud Endpoints o API Gateway
  - Para servir como el punto de entrada seguro y único para todas las solicitudes externas a la API, se configurará un Cloud Load Balancer [información no proveniente de las fuentes]. Este balanceador de carga externo distribuirá de manera eficiente el tráfico hacia las instancias del servidor de aplicación en Cloud Run [información no proveniente de las fuentes].
  - Rol en la Arquitectura: Además de la distribución de tráfico, se podría integrar con Cloud Endpoints o API Gateway [información no proveniente de las fuentes]. Esto proporcionaría capacidades adicionales como la gestión de API, monitoreo, control de acceso basado en claves de API, y la imposición de límites de tasa (rate limiting) para proteger el backend de usos indebidos o ataques de denegación de servicio [información no proveniente de las fuentes]. Garantizará que solo las solicitudes válidas y autorizadas lleguen a los servicios internos.
- Almacenamiento de Archivos: Cloud Storage Buckets
  - Los archivos estáticos y multimedia de la plataforma, como las imágenes y videos de alta calidad utilizados en la estrategia de contenido para redes sociales, y otros documentos necesarios, se almacenarán en Cloud Storage Buckets [información no proveniente de las fuentes].
  - Configuración y Consideraciones: Se configurarán buckets dedicados para diferentes tipos de contenido (ej., uno para imágenes de productos, otro para videos de marketing, otro para documentos internos de distribuidores) [información no proveniente de las fuentes].
  - Seguridad y Acceso: Los buckets se configurarán con permisos de acceso específicos, utilizando políticas de IAM (Identity and Access Management) [información no proveniente de las fuentes]. Los archivos públicos (como imágenes de productos para el sitio web o redes sociales) se harán accesibles públicamente con URLs generadas, mientras que los archivos sensibles (ej., documentos de negocio para distribuidores) tendrán acceso restringido y se servirán a través de URLs firmadas temporalmente o mediante el backend una vez que el usuario haya sido autorizado [información no proveniente de las fuentes]. Se implementará versionamiento para la recuperación de archivos y la protección contra eliminaciones accidentales [información no proveniente de las fuentes].
- Monitoreo y Registro: Cloud Logging y Cloud Monitoring
  - Para la recopilación centralizada y el análisis de registros, se utilizará Cloud Logging [información no proveniente de las fuentes].
  - Configuración: Cloud Logging se configurará automáticamente para recopilar los logs del servidor de aplicación (Cloud Run), así como los logs de otros servicios de GCP como el Load Balancer y Cloud SQL [información no proveniente de las fuentes]. Esto permitirá una visibilidad completa del comportamiento del sistema, facilitando la depuración, la identificación de problemas de seguridad o rendimiento, y la auditoría [información no proveniente de las fuentes].
  - Complementariamente, Cloud Monitoring se utilizará para recopilar métricas de rendimiento (uso de CPU, memoria, latencia de solicitudes, errores) y configurar alertas proactivas sobre el estado y la salud de la aplicación y la infraestructura [información no proveniente de las fuentes].

- Conexión a Base de Datos: Cloud SQL
    - El backend se conectará a la base de datos de la plataforma, que estará alojada en Cloud SQL [información no proveniente de las fuentes].
    - Nota: La configuración detallada de Cloud SQL, incluyendo el tipo de base de datos (PostgreSQL, MySQL), la capacidad y las estrategias de respaldo, se abordará en el Documento 2, "Arq. Base de Datos y Gestión de Datos" [información proveniente de la estructura del documento].
- Consideraciones de Red y Seguridad: VPC y Firewall
    - Para garantizar la comunicación segura y controlada entre los diferentes servicios de GCP, se configurará una Virtual Private Cloud (VPC) [información no proveniente de las fuentes].
    - Firewall Rules: Se implementarán reglas de firewall dentro de la VPC para restringir el tráfico entre los servicios y solo permitir las comunicaciones necesarias [información no proveniente de las fuentes]. Por ejemplo, se permitirá que el servidor de aplicación se conecte a la instancia de Cloud SQL, pero se bloqueará el acceso directo a la base de datos desde internet [información no proveniente de las fuentes]. Esto crea una capa adicional de seguridad y aislamiento para la infraestructura del backend.

Consideraciones Adicionales del Backend
Además de la configuración de servicios fundamental, existen consideraciones críticas para el backend que garantizan la robustez, seguridad y confiabilidad de la plataforma de Remex 8.

- Validación de Datos en el Backend:
  - Es imperativo implementar una rigurosa validación de datos en el backend para todas las entradas recibidas a través de las APIs [información no proveniente de las fuentes, sino una práctica recomendada en arquitectura de software]. Esta validación asegura la integridad de la información, previene el ingreso de datos corruptos o maliciosos, y mantiene la consistencia de la base de datos [información no proveniente de las fuentes]. Esto incluye la verificación de tipos de datos, rangos, formatos, obligatoriedad de campos y la aplicación de reglas de negocio específicas antes de procesar o almacenar cualquier dato. Una validación robusta es la primera línea de defensa contra vulnerabilidades de seguridad y errores de aplicación [información no proveniente de las fuentes].
- Registro de Actividades Clave (Logs de Auditoría):
  - Reafirmando la importancia de lo mencionado previamente en la sección de "Configuración de Servicios de GCP para el Backend" sobre Cloud Logging [información no proveniente de las fuentes], es esencial que el backend genere y recopile logs de auditoría para las actividades clave [información no proveniente de las fuentes]. Estos logs deben registrar acciones significativas como inicios de sesión (exitosos y fallidos), modificaciones de datos de usuarios o productos, transacciones (ventas de productos, registros de nuevos socios), y cualquier intento de acceso no autorizado [información no proveniente de las fuentes]. La configuración de Cloud Logging para recopilar estos logs del servidor de aplicación (Cloud Run) y otros servicios de GCP es crucial [información no proveniente de las fuentes]. Los logs de auditoría son fundamentales para:
    - Seguridad: Permitir la detección temprana de patrones de comportamiento sospechosos o ataques, facilitando la investigación forense en caso de una brecha de seguridad [información no proveniente de las fuentes].
    - Seguimiento y Cumplimiento: Proporcionar un historial inmutable de las operaciones del sistema, lo que es vital para la resolución de problemas, la rendición de cuentas y el cumplimiento de posibles regulaciones futuras [información no proveniente de las fuentes].