DuocUC ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

Novuvitae Labs:
Manual Técnico de Software.

Índice:

1. Introducción

El presente **Manual Técnico del Software "Novuvitae Labs"** tiene como objetivo documentar en profundidad la estructura interna, configuración, arquitectura, funcionamiento operativo y lineamientos de mantenimiento del sistema desarrollado en el marco del Proyecto APT. Este documento constituye la referencia principal para desarrolladores, administradores del sistema y personal técnico involucrado en la continuidad del proyecto, garantizando su correcta implementación, soporte y escalabilidad futura.

Novuvitae Labs es una plataforma web comunitaria diseñada para centralizar recursos, fomentar la participación de los usuarios y facilitar la difusión de información mediante dos módulos principales:

- **Foro** estructurado por categorías, con creación y gestión de posts y comentarios.
- **Módulo de noticias**, administrado desde un panel especializado y orientado a comunicar actualizaciones relevantes a la comunidad.

Además, el sistema incorpora un **mecanismo seguro de autenticación mediante OAuth2 con Discord**, lo que permite simplificar el acceso de los usuarios y automatizar su registro en la base de datos interna. Esta integración asegura un flujo de ingreso rápido, moderno y estandarizado, aprovechando la infraestructura de identificación provista por Discord.

El software cuenta también con un **dashboard administrativo**, desde donde se gestionan los contenidos del sitio (noticias, usuarios, entre otros), y se supervisa el funcionamiento general de la plataforma. Para persistencia de datos se utiliza una **base de datos relacional**, cuyo modelo está diseñado para garantizar integridad referencial, eficiencia en consultas y escalabilidad ante crecimiento del contenido o la cantidad de usuarios.

Este manual tiene por finalidad entregar:

- Una visión clara de la **arquitectura del sistema**.
- La explicación detallada del **modelo de datos** y sus relaciones.
- Las instrucciones de **instalación, configuración, despliegue y mantenimiento**.
- La descripción del funcionamiento interno de cada módulo.
- Las buenas prácticas necesarias para garantizar la **seguridad, estabilidad y evolución técnica** del software.

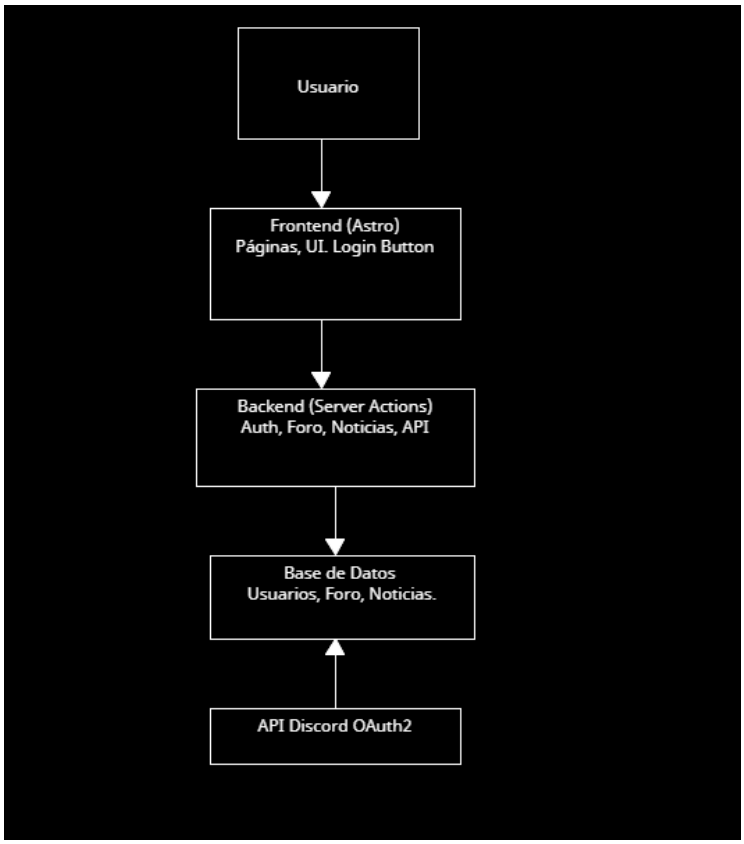
En términos generales, este documento proporciona todos los elementos fundamentales para que cualquier miembro del equipo técnico pueda comprender el funcionamiento de **Novuvitae Labs**, modificarlo, ampliarlo o restaurarlo en caso de ser necesario.

2. Arquitectura del Sistema

La arquitectura de **Novuvitae Labs** se basa en un enfoque modular y escalable que separa claramente las responsabilidades entre el frontend, los servicios de autenticación, la base de datos y las integraciones externas. El diseño se centra en mantener un sistema fácil de mantener, extensible y preparado para futuros desarrollos.

A continuación, se presenta el diagrama conceptual y la explicación de cada uno de sus componentes.

2.1 Diagrama General de Arquitectura



2.2 Componentes de la Arquitectura

2.2.1 Frontend – Astro

El frontend del sistema está desarrollado con **Astro**, un framework moderno orientado al rendimiento y a la generación híbrida de contenidos (SSR/SSG). Sus principales responsabilidades dentro del proyecto son:

- Renderizado de las vistas del foro, noticias y páginas informativas.
- Interacción con los servicios internos mediante endpoints.
- Gestión del flujo visual de autenticación con Discord.
- Consumo de datos provenientes del backend (usuarios, posts, categorías, noticias, etc.).
- Implementación de componentes reutilizables y una estructura escalable para el UI.

Astro permite generar páginas rápidas, ligeras y optimizadas, lo que facilita la experiencia del usuario en dispositivos variados.

2.2.2 Backend / Servicios Auxiliares

Si bien Novuvitae Labs no utiliza un backend tradicional independiente, se apoya en:

- **Endpoints internos de Astro (API Routes)** para manejar:
  - Operaciones de CRUD del foro.
  - Gestión de noticias.
  - Registro automático de usuarios.
  - Validación de sesiones.
  - Consultas a la base de datos.
  - Lógica del panel administrativo.

Estos endpoints funcionan como un backend ligero que encapsula la lógica del negocio y sirve como capa intermediaria entre el frontend y la base de datos.

2.2.3 Base de Datos Relacional

El sistema utiliza una **base de datos relacional (SQL)**, la cual estructura la información en tablas relacionadas para asegurar consistencia e integridad.
El modelo incluye entidades tales como:

- Usuarios
- Comunidades (si aplica)
- Categorías del foro
- Posts
- Comentarios

- Noticias

El diseño favorece:

- Claves primarias auto incrementales.
- Relaciones **1:N** entre usuarios → posts → comentarios.
- Relaciones entre administradores y recursos gestionados.

ORM / Conector

Puede utilizarse un ORM como **Prisma**, PlanetScale o un cliente SQL nativo; en cualquier caso, los endpoints sirven como capa única de acceso a datos.

2.2.4 Integración con Discord OAuth2

La autenticación se realiza mediante el protocolo **OAuth2**, proporcionando:

- Inicio de sesión seguro mediante Discord.
- Obtención del perfil básico del usuario (ID, nombre, email, avatar).
- Inserción automática del usuario en la base de datos si no existe.
- Generación de una sesión válida para navegar en el sitio.
- Gestión de permisos básicos según rol (usuario / admin).

Este sistema evita manejar contraseñas manuales y garantiza un estándar profesional de autenticación.

2.2.5 Servidor / Hosting

Dependiendo del entorno definido por el equipo, el sistema puede desplegarse en:

- **Vercel**, Netlify o plataformas enfocadas en Astro.
- **Railway, Render o Supabase** para la base de datos.
- Hosting tradicional si se requiere un VPS.
- Github

En cualquiera de los casos, la arquitectura está preparada para:

- Despliegue continuo (CI/CD).
- Escalabilidad horizontal del frontend.
- Persistencia segura de datos.

Resumen del objetivo de la arquitectura

La arquitectura de **Novuvitae Labs** busca:

- Mantener bajo acoplamiento entre frontend, lógica y base de datos.
- Facilitar escalabilidad del foro, noticias y futuras expansiones.
- Ofrecer un sistema modular, seguro y sencillo de mantener.

3. Requerimientos Técnicos

Esta sección describe los **requerimientos mínimos** y **recomendados** para la instalación, ejecución y mantenimiento del sistema **Novuvitae Labs**. Incluye tanto los aspectos de hardware como las dependencias software necesarias.

3.1 Requerimientos de Hardware

Aunque el sistema es liviano y puede ejecutarse en equipos de uso general, se recomiendan las siguientes especificaciones:

Servidor de Desarrollo (local)

Componente	Requerimiento Mínimo	Recomendado
CPU	2 núcleos	4 núcleos o más
RAM	4 GB	8 GB
Almacenamiento	2 GB libres	SSD 10 GB
Red	Acceso a Internet	Banda ancha estable

Servidor de Producción (hosting)

Componente	Requerimiento Mínimo	Recomendado
CPU	1 vCPU	2 vCPU
RAM	512 MB	1–2 GB
Almacenamiento	2 GB	5+ GB SSD
Base de datos	Hosting SQL básico	Hosting SQL profesional / DBaaS

Novuvitae Labs puede ejecutarse casi en cualquier entorno moderno debido a su arquitectura ligera basada en Astro.

3.2 Requerimientos de Software

Sistemas Operativos Compatibles

- Windows 10/11
- macOS (Intel y ARM)
- Linux (Ubuntu, Debian, Arch o equivalentes)

El sistema puede instalarse en cualquier OS que soporte Node.js.

3.3 Dependencias Principales de Desarrollo

Node.js

- **Versión mínima:** 18.x
- **Recomendada:** 20.x
  Es indispensable para ejecutar Astro, las rutas API y el entorno de desarrollo.

Astro

Framework principal del frontend y endpoints internos.

Base de Datos

Sistema compatible con SQL, idealmente:

- MySQL
- PostgreSQL
- PlanetScale
- Supabase

ORM / Cliente de Base de Datos

(Según lo que utilicen)

- Prisma (recomendado)
- Mysql2 / pg / cliente SQL nativo

Integración OAuth2 con Discord

Requiere:

- Cuenta de desarrollador en Discord
- Configuración del **Discord Developer Portal**
- Variables de entorno:

- DISCORD_CLIENT_ID
- DISCORD_CLIENT_SECRET
- DISCORD_REDIRECT_URI

3.4 Requerimientos para despliegue (Deployment)

El proyecto puede desplegarse en plataformas como:

Frontend

- Vercel
- Netlify
- GitHub Pages (con limitaciones)
- Cualquier servidor Node.js

Base de datos

- Railway
- Supabase
- PlanetScale
- Render
- Servidor SQL propio

Variables de entorno necesarias

Variable	Uso
DATABASE_URL	Conexión a la base de datos
DISCORD_CLIENT_ID	Autenticación OAuth2
DISCORD_CLIENT_SECRET	Autenticación OAuth2
DISCORD_REDIRECT_URI	URI de retorno post-login
SESSION_SECRET	Gestión de sesiones segura

3.5 Herramientas de desarrollo utilizadas

- **VS Code**

- Extensiones recomendadas:
    - ESLint
    - Prettier
    - Astro Tools
- **Git + GitHub** para control de versiones
- **Terminal** (CMD, PowerShell, Bash, Zsh)

4. Instalación:

Esta sección describe los pasos necesarios para instalar, configurar y ejecutar el sistema **Novuvitae Labs** tanto en un entorno de desarrollo como en un entorno de producción. Los procedimientos están diseñados para que cualquier técnico o desarrollador pueda preparar el proyecto desde cero.

4.1. Clonar el Repositorio

Para obtener el código fuente del proyecto, clone el repositorio desde la plataforma correspondiente (GitHub, GitLab u otra).

git clone <URL-del-repositorio>

cd novuvitae-labs

Asegúrese de estar ubicado dentro de la carpeta raíz del proyecto antes de continuar.

4.2. Instalar Dependencias

El proyecto utiliza Node.js y un conjunto de paquetes gestionados mediante **npm**.

Ejecute:

npm install

Esto instalará todas las librerías necesarias para ejecutar:

- El frontend con Astro.
- Las rutas API internas.
- Integración OAuth con Discord.
- Conexión con la base de datos.

4.3. Configurar Variables de Entorno

El sistema requiere configurar parámetros externos para poder interactuar con Discord y con la base de datos.

1. Crear un archivo llamado:

.env

2. Agregar dentro del archivo las siguientes claves (según configuración real del proyecto):

DATABASE_URL="mysql://usuario:password@host:puerto/base"

DISCORD_CLIENT_ID="XXXXXXXXXXX"

DISCORD_CLIENT_SECRET="XXXXXXXXXXX"

DISCORD_REDIRECT_URI="https://tusitio.com/api/auth/callback"

SESSION_SECRET="cadena-segura-generada"

Notas importantes:

- Las credenciales **no deben compartirse públicamente**.
- Si se usa Prisma, recuerde que DATABASE_URL debe ser compatible con el proveedor configurado.
- Para despliegue en Vercel o Netlify, estas variables se configuran desde el panel de administración.

4.4. Ejecución del Proyecto

Modo Desarrollo

Para levantar el servidor local:

npm run dev

Esto iniciará el entorno en:

http://localhost:4321

Podrá visualizar:

- Página principal
- Login con Discord
- Foro y categorías
- Panel de administración
- Módulo de noticias

Astro recargará automáticamente los cambios.

Modo Producción

Primero, generar la build optimizada:

npm run build

Para visualizar un entorno similar al despliegue final:

npm run preview

Esto sirve para pruebas y validación antes de subir a producción.

Despliegue Final

Dependiendo del hosting elegido, se deben subir:

- Carpetas generadas por el build
- Variables de entorno
- Configuración de base de datos

Recomendado para frontend Astro:
Vercel o **Netlify**
Para la base de datos:
Supabase, Railway o PlanetScale

5. Configuración de Discord OAuth2

El sistema Novuvitae Labs utiliza **Discord OAuth2** como método principal de autenticación, permitiendo a los usuarios iniciar sesión con su cuenta de Discord y vinculando automáticamente sus datos a la base de datos del sistema.

Esta sección explica paso a paso cómo crear la aplicación en Discord, obtener las credenciales necesarias e integrarlas en el proyecto.

5.1. Crear una Aplicación en Discord Developer Portal

1. Ingresar a:
   https://discord.com/developers/applications
2. Hacer clic en **"New Application"**.
3. Asignar un nombre a la aplicación (por ejemplo:
   Novuvitae Labs Auth).
4. Aceptar los términos y crear la aplicación.

Después de esto, accederás al panel principal donde podrás gestionar las credenciales y configuraciones de OAuth2.

5.2. Obtener Credenciales Necesarias

Client ID

1. En el menú lateral, seleccionar **OAuth2** → **General**.
2. Copiar el campo **"Client ID"**.

Ejemplo:

DISCORD_CLIENT_ID="123456789012345678"

Client Secret

1. Dentro del mismo menú (OAuth2 → General), bajar a la sección **Client Secret**.
2. Hacer clic en **Reset Secret** (si aún no existe uno).
3. Copiar el valor generado.

Ejemplo:

DISCORD_CLIENT_SECRET="aBcDeFGhIJKLmNoPQRstuVwXYZ123456"

Advertencia:
Nunca publique este valor en repositorios o sitios públicos.

Redirect URL

El Redirect URL indica dónde Discord enviará al usuario después del inicio de sesión exitoso.

1. En **OAuth2 → Redirects**
2. Dar clic en **Add Redirect**
3. Agregar la URL correspondiente del proyecto:

Para entorno local:

http://localhost:4321/api/auth/callback

Para producción (ejemplo):

https://novuvitaelabs.com/api/auth/callback

4. Guardar cambios.

En el archivo .env, esta ruta se configura como:

DISCORD_REDIRECT_URI="https://novuvitaelabs.com/api/auth/callback"

5.3. Configurar el Alcance (Scopes)

En la sección **OAuth2 → URL Generator**:

1. Seleccionar los *scopes* necesarios:
    - identify (obligatorio para obtener nombre y avatar)
    - email (opcional, si se desea registrar el correo)
2. En "OAuth2 URL Generator", el sistema generará un enlace de autenticación.

Este enlace es el que el sistema utiliza internamente para redirigir al usuario.

5.4. Activar Discord OAuth2 en el Proyecto

Una vez obtenidas todas las credenciales:

1. Abrir el archivo .env
2. Ingresar:

DISCORD_CLIENT_ID="XXX"

DISCORD_CLIENT_SECRET="XXX"

DISCORD_REDIRECT_URI="XXX"

SESSION_SECRET="cadena_unica_segura"

3. El proyecto ya está preparado para usar Discord OAuth2.

5.5. Funcionamiento Interno en Novuvitae Labs

- El usuario hace clic en **"Login con Discord"**.
- Es redirigido a la página oficial de Discord OAuth2.
- Una vez autorizado:
    - El backend recibe un **authorization code**.
    - Se intercambia por un **access token**.
    - Se solicita información del usuario a la API de Discord.
    - Si el usuario no existe en la base de datos:
        - Se registra automáticamente (ID de Discord, nombre, avatar).
    - Si ya existe:
        - Se actualiza si corresponde o simplemente se inicia sesión.

El sistema gestiona todo el flujo sin intervención manual.

6. Base de Datos

La base de datos de **Novuvitae Labs** está diseñada para soportar los módulos principales del sistema: foro, noticias, usuarios y administración.
El modelo sigue un esquema **relacional**, donde las entidades se vinculan mediante claves foráneas para garantizar consistencia, integridad referencial y escalabilidad.

6.1. Modelo de Datos (Relaciones Principales)

Las relaciones entre las tablas son las siguientes:

Relaciones entre usuarios, posts y comentarios (foro):

- **Usuarios (id)**
  1 --- ∞
  Posts (user_id)
- **Usuarios (id)**
  1 --- ∞
  Comentarios (user_id)
- **Posts (id)**
  1 --- ∞
  Comentarios (post_id)

Relaciones entre noticias, categorías y usuarios:

- **Categorías (id)**
  1 --- ∞
  Noticias (categoria_id)
- **Usuarios (id)**
  1 --- ∞
  Noticias (usuario_id)

6.2. Diagrama Conceptual del Modelo de Datos (Descripción Visual)

(Agregar diagrama)

6.3. Descripción Técnica de Cada Tabla

Tabla: usuarios

Campo	Tipo	Descripción
id	int (PK)	Identificador único interno del sistema
discord_id	string	ID del usuario proveniente de Discord
nombre	string	Nombre visible del usuario
avatar	string	URL del avatar obtenido por OAuth2
rol	string	Role del usuario (user/admin)

Notas técnicas:

- discord_id permite vincular la cuenta externa.
- El campo rol activa funciones dentro del panel administrativo.

Tabla: categorias

Campo	Tipo	Descripción
id	int (PK)	Identificador de la categoría
nombre	string	Nombre de la categoría

Uso:

- Compartida por el módulo de foro y noticias.

Tabla: posts

Campo	Tipo	Descripción
id	int (PK)	Identificador del post
usuario_id	int (FK)	Usuario que creó el post
categoria_id	int (FK)	Categoría asignada
titulo	string	Título del post
contenido	text	Cuerpo del mensaje
creado_en	datetime	Timestamp de creación

Relaciones:

- usuario_id → usuarios.id
- categoria_id → categorias.id

Tabla: comentarios

Campo	Tipo	Descripción
id	int (PK)	Identificador del comentario
post_id	int (FK)	Post al que pertenece
usuario_id	int (FK)	Usuario autor del comentario
contenido	text	Texto del comentario

Relaciones:

- post_id → posts.id
- usuario_id → usuarios.id

Notas:

- Un post puede tener múltiples comentarios.
- Se recomienda habilitar “cascade delete” en caso de eliminación de posts.

Tabla: noticias

Campo	Tipo	Descripción
id	int (PK)	Identificador de la noticia
titulo	string	Título de la noticia
contenido	text	Contenido extendido
categoria_id	int (FK)	Categoría asignada
usuario_id	int (FK)	Autor de la noticia (administrador)
creado_en	datetime	Fecha de publicación

Relaciones:

- categoria_id → categorias.id
- usuario_id → usuarios.id

6.4. Consideraciones del Diseño

- Cada entidad está normalizada para evitar redundancia.
- Las relaciones son claras y permiten consultas eficientes.
- Las noticias y posts comparten categorías para simplificar la administración.
- El sistema permite futura expansión:
    - Más roles
    - Reacciones
    - Sistemas de permisos
    - Moderación avanzada

7. Estructura del Software (Frontend)

El frontend de **Novuvitae Labs** está construido con **Astro**, siguiendo una estructura modular que facilita la mantenibilidad, escalabilidad y claridad del código.
A continuación se detalla la estructura principal del directorio src/, junto con la función de cada carpeta y la organización del flujo interno del sistema.

7.1. Árbol General del Proyecto

src/

├─ pages/

├─ components/

├─ layouts/

├─ services/

├─ db/

├─ utils/

└─ assets/

7.2. Descripción de Carpetas y Función de Cada Una

/pages/

Contiene las páginas principales del sitio.
Cada archivo dentro de pages/ se traduce directamente en una ruta del sitio web.

Ejemplos típicos:

- /index.astro → Página principal
- /foro/index.astro → Listado del foro
- /foro/post/[id].astro → Vista de un post
- /noticias/index.astro → Listado de noticias
- /admin/* → Sección administrativa

Función:
Definir las vistas públicas y privadas accesibles desde el navegador.

/components/

Incluye componentes reutilizables del frontend que se integran dentro de las páginas.

Ejemplos:

- Navbar.astro
- Footer.astro
- PostCard.astro
- CommentBox.astro
- AdminSidebar.astro

Función:
Evitar duplicación de código y facilitar la edición visual desde un solo lugar.

/layouts/

Plantillas de diseño que comparten estructura general entre múltiples páginas.

Ejemplos:

- MainLayout.astro
- AdminLayout.astro
- AuthLayout.astro

Función:
Permitir consistencia visual en todo el proyecto y simplificar el markup repetitivo (header, sidebar, footer, etc.).

/services/

Servicios lógicos del frontend, incluyendo:

- Manejadores de autenticación (Discord OAuth2).
- Funciones para interactuar con APIs.
- Consultas a endpoints internos.
- Acceso abstracto a datos del foro y noticias.

Archivos típicos:

- authService.js
- foroService.js
- noticiasService.js

Función:
Centralizar la lógica que se comunica entre frontend ↔ backend/base de datos.

/db/

Contiene:

- Conexión a la base de datos (ej. usando better-sqlite3, Drizzle, Prisma, o similar).
- Modelos del sistema organizados por entidad (usuarios, posts, etc.).
- Funciones CRUD internas.

Ejemplos:

- connection.js
- usuarios.js
- posts.js
- comentarios.js
- noticias.js

Función:
Gestionar la persistencia del sistema de forma ordenada, modular y segura.

/utils/

Utilidades generales del sistema, por ejemplo:

- Formateadores de fechas.
- Validadores.
- Funciones de sanitización.
- Helpers comunes.

Ejemplos:

- dateFormat.js
- sanitizeInput.js
- permissions.js

Función:
Evitar lógica duplicada y agrupar funciones auxiliares.

/assets/ ***(si aplica)***

Imágenes, íconos, estilos globales, fuentes u otros recursos estáticos.

7.3. Organización del Flujo de Código

El flujo de funcionamiento del frontend se organiza de la siguiente manera:

1. El usuario accede a una página

Astro procesa los archivos dentro de /pages/.

2. La página usa un layout

Permite mantener coherencia visual.

3. La página incluye componentes

Navbar, tarjetas, comentarios, etc.

4. Los componentes interactúan con servicios

Por ejemplo:

- Obtener posts
- Guardar comentarios

- Iniciar sesión con Discord

5. Los servicios interactúan con la base de datos

A través de funciones dentro de /db/.

6. Los datos se devuelven a la página

Y se renderizan estática o dinámicamente.

7.4. Beneficios de Esta Estructura

- Modularidad total
- Fácil mantenimiento
- Permite escalar nuevas funciones
- Separación clara entre vistas, lógica y persistencia
- Integración limpia con Discord OAuth2
- Flujo consistente en todo el proyecto

8. Funcionamiento del Sistema

El funcionamiento interno de **Novuvitae Labs** se organiza en flujos técnicos que describen cómo la aplicación procesa las acciones clave: autenticación, gestión de contenido, consultas al servidor y renderización de vistas.
A continuación se detallan los principales procesos técnicos que sostienen el funcionamiento del software.

8.1. Flujo técnico de Login con Discord (OAuth2)

El sistema utiliza el protocolo **OAuth2** para autenticar usuarios mediante Discord.
El flujo técnico es el siguiente:

1. Inicio del login

El cliente ejecuta una redirección hacia la URL de autorización de Discord:

https://discord.com/api/oauth2/authorize

?client_id=XXXXX

&redirect_uri=ENCODED_REDIRECT

&response_type=code

&scope=identify%20email

2. Discord valida al usuario

- Discord presenta la pantalla de autorización.
- El usuario confirma acceso.

3. Discord redirige al servidor del proyecto

Discord envía un **authorization code** al redirect_uri.

Ejemplo:

/auth/discord/callback?code=1234abcd

4. Intercambio del código por un token

El backend/servidor realiza una petición POST a:

https://discord.com/api/oauth2/token

Enviando:

- client_id
- client_secret
- code
- redirect_uri

Discord responde con un **access_token**.

5. Obtención de datos del usuario

Con el token, el sistema consulta:

GET https://discord.com/api/users/@me

y recibe:

- id
- username
- avatar
- email (si tiene permisos)

6. Registro / actualización en la base de datos

El sistema aplica la siguiente lógica:

- Si discord_id no existe → crear usuario nuevo.

- Si ya existe → actualizar datos (nombre, avatar).

7. Creación de sesión local

El sistema genera:

- cookie de sesión
- o token local (según implementación)

Permitiendo al usuario navegar autenticado dentro de la plataforma.

8.2. Flujo Técnico de Creación de un Post

1. Envío del formulario

El usuario autenticado envía un formulario desde el frontend con:

- título
- contenido
- categoría
- id del usuario (extraído de la sesión)

2. Validación en el servidor

Se valida:

- longitud del título
- sanitización del contenido
- existencia del usuario y categoría
- verificación de permisos

3. Inserción en la base de datos

Se ejecuta una operación SQL (o mediante ORM), por ejemplo:

INSERT INTO posts (usuario_id, categoria_id, titulo, contenido, creado_en)

VALUES (...)

El sistema genera un id único para el nuevo post.

4. Regeneración o render del contenido

El servidor devuelve una respuesta al frontend:

- ubicación del nuevo post
- contenido limpio y validado

- metadatos (fecha, autor, categoría)

La página del post se renderiza dinámicamente o se navega automáticamente a su URL final:

/foro/post/{id}

8.3. Flujo Técnico de Creación de Comentarios

1. Envío de datos del comentario

El frontend envía:

- contenido del comentario
- id del post
- id del usuario autenticado

2. Validación

Similar al caso anterior:

- sanitización
- longitud
- existencia del post
- usuario autenticado

3. Inserción en tabla comentarios

INSERT INTO comentarios (post_id, usuario_id, contenido)

VALUES (...)

4. Actualización dinámica

El nuevo comentario se incorpora inmediatamente al render del post.

8.4. Flujo Técnico de Gestión de Noticias (Admin)

1. Acceso al panel de administración

El usuario debe tener rol admin.

2. Envío del formulario de creación o edición

Se envían:

- título

- contenido
- categoría
- usuario creador

3. Validación y seguridad

El sistema verifica:

- rol de administrador
- integridad del contenido
- existencia de la categoría

4. Registro en la tabla noticias

INSERT INTO noticias (...)

5. Actualización automática de la sección pública

La noticia aparece en la sección:

/noticias

o en un módulo destacado de la página principal.

8.5. Flujo Técnico del Dashboard Administrativo

El dashboard carga datos del sistema para mostrar métricas y estado general.

1. Solicitud de métricas

El frontend realiza consultas hacia servicios internos (API o funciones directas en Astro):

- Número de usuarios registrados
- Número de posts
- Número de comentarios
- Número de noticias

2. Consulta en BD

Ejemplo:

SELECT COUNT(*) FROM usuarios;

SELECT COUNT(*) FROM posts;

SELECT COUNT(*) FROM comentarios;

SELECT COUNT(*) FROM noticias;

3. Procesamiento en el servidor

Se formatea la información en objetos JSON o datos crudos.

4. Renderización de gráficos o tarjetas

El dashboard muestra:

- estadísticas globales
- actividad reciente
- datos de crecimiento

(Dependiendo de los componentes implementados.)

8.6. Renderización de Páginas

Astro combina:

- datos provenientes de la base de datos
- layouts
- componentes

para generar:

- páginas estáticas (cuando corresponde)
- o páginas dinámicas usando endpoints y reactividad mínima

El servidor entrega HTML optimizado que integra el contenido actualizado en tiempo real, especialmente en páginas de foro, noticias y panel administrativo.

9. Operación del Sistema

Este apartado describe los procedimientos necesarios para operar, actualizar y monitorear el sistema **Novuvitae Labs** desde el punto de vista técnico. Está dirigido a administradores del sistema, desarrolladores y personal encargado del mantenimiento.

9.1. Inicio del Servidor

El proyecto utiliza los comandos estándar de Astro para su ejecución, tanto en entornos de desarrollo como producción.

9.1.1. Modo Desarrollo

Este modo permite trabajar de forma local con recarga automática y mensajes detallados de error.

npm run dev

El servidor se inicia por defecto en:

http://localhost:4321/

Características del modo dev:

- recarga en tiempo real (HMR)
- logs detallados de compilación
- validación inmediata de cambios

9.1.2. Compilar para Producción

Para generar los archivos optimizados del proyecto:

npm run build

Esto genera la carpeta:

/dist

incluyendo:

- HTML optimizado
- JS y CSS minificado
- assets listos para deploy

9.1.3. Previsualización en modo producción

Antes de subir la aplicación al servidor, es posible ejecutarla localmente con el mismo comportamiento que tendrá en producción:

npm run preview

9.2. Actualización de la Aplicación

Existen dos tipos principales de actualización: **actualización del código fuente** y **actualización de dependencias**.

9.2.1. Actualización del código fuente

1. Obtener los últimos cambios desde el repositorio:

git pull origin main

2. Volver a instalar dependencias (si hubo cambios en package.json):

npm install

3. Volver a construir el proyecto:

npm run build

4. Reiniciar el servicio (según hosting).

9.2.2. Actualización de dependencias

Para revisar si existen versiones más recientes:

npm outdated

Para actualizar paquetes:

npm update

Si se requiere una actualización mayor y manual:

npm install paquete@latest

Tras actualizar dependencias, **siempre reconstruir**:

npm run build

9.3. Reinicio de Servicios

El método de reinicio depende del hosting. Algunas formas comunes:

9.3.1. En entorno local

Simplemente detener el proceso:

- CTRL + C
  y volver a ejecutarlo:

npm run dev

o para producción:

npm run preview

9.3.2. En servidores basados en Node con PM2

Si se usa PM2:

pm2 restart novuvitae

Ver estado:

pm2 list

Ver logs:

pm2 logs novuvitae

9.4. Logs y Monitoreo

El monitoreo básico de errores y comportamiento del sistema se puede realizar mediante:

9.4.1. Logs del servidor de desarrollo

Astro muestra logs en consola cuando se ejecuta:

npm run dev

Incluye:

- errores de compilación
- errores de rutas
- advertencias de dependencias

9.4.2. Logs en producción

Dependiendo del hosting (PM2, Docker, VPS), se pueden usar:

Con PM2:

pm2 logs

pm2 logs novuvitae

Con Docker:

docker logs <container>

9.4.3. Monitoreo del estado de la aplicación

Se recomienda agregar:

- **Ping automático** (cron job o uptime monitor)
- **Monitoreo de respuesta HTTP (200/400/500)**
- **Monitoreo de carga del servidor** (CPU, RAM, disco)
- **Monitoreo de la base de datos** (conexiones activas, tamaño)

Servicios recomendados (opcionales):

- UptimeRobot
- Grafana + Prometheus
- PM2 Monitoring Dashboard

9.5. Respaldo y seguridad (recomendado)

Para operación estable se sugiere:

Backups

- Copias diarias de la base de datos
- Backup semanal del repositorio (si es privado)
- Backup de archivos de configuración (.env) en entorno seguro

Seguridad

- Regenerar client secret de Discord si se sospecha exposición
- Mantener dependencias actualizadas
- Restringir el acceso al panel admin por rol

10. Seguridad

La seguridad es un componente fundamental en la plataforma **Novuvitae Labs**, especialmente considerando que gestiona usuarios autenticados, información de perfil obtenida desde Discord y contenido generado por la comunidad.

Este apartado describe las medidas implementadas y las buenas prácticas recomendadas para mantener la integridad y confiabilidad del sistema.

10.1. Manejo del Login OAuth (Discord)

El sistema utiliza el protocolo **OAuth2** proporcionado por Discord, lo que permite una autenticación segura sin almacenar contraseñas en la plataforma.

Medidas de seguridad aplicadas:

✓ Tokens no se almacenan en cliente.

Solo se usa el token devuelto por Discord durante el proceso de validación.
No se guarda access tokens en localStorage ni cookies públicas.

✓ Verificación del usuario en servidor/servicio interno.

Después de obtener el código de Discord, el servidor solicita:

- access_token
- datos del usuario (/users/@me)

Se validan campos esenciales como:

- discord_id
- email (si se requiere)
- avatar
- nombre

✓ Enmascaramiento de credenciales

Dentro del proyecto:

- DISCORD_CLIENT_ID
- DISCORD_CLIENT_SECRET
- DISCORD_REDIRECT_URI

se guardan únicamente en **variables de entorno (.env)**.
Nunca se incluyen en el repositorio.

✓ Validación contra suplantación

Los usuarios se registran o actualizan únicamente si:

- el ID proviene directamente de Discord,
- el token está validado por el servidor de Discord.

Esto impide que un usuario falsifique identidad.

10.2. Protección de Rutas

El sistema utiliza middleware o validaciones internas para controlar el acceso a ciertas áreas.

Rutas protegidas:

- Panel de administración
- Panel de noticias
- Gestión de categorías
- Dashboard
- Creación/moderación de posts
- Gestión de usuarios (si aplica)

Medidas aplicadas:

✓ Autenticación obligatoria

Toda ruta crítica verifica la existencia de:

- sesión activa,
- usuario presente en la base de datos.

Si no cumple, se redirige a login.

✓ Protección por roles

Antes de acceder a una ruta:

```
if (user.role !== 'admin') {
  deny_access()
}
```

Roles disponibles:

- **admin** → acceso completo
- **moderador** (opcional) → administración de foro
- **usuario** → uso normal del sistema

10.3. Prevención de Spam en el Foro

El foro admite la producción de contenido abierto, por lo que se implementan medidas básicas para evitar abuso.

Medidas aplicadas o recomendadas:

✓ Límite de frecuencia de publicación

Evita que un usuario cree múltiples posts/comentarios en segundos.

Ejemplo:

- 1 post cada 10 segundos
- 1 comentario cada 3 segundos

✓ Filtrado de contenido vacío

Se impide enviar:

- posts sin título
- comentarios sin contenido
- mensajes con solo espacios

✓ Sanitización de HTML

Antes de guardar contenido, se filtran etiquetas peligrosas:
<script>, <iframe>, <object>, etc.

✓ Longitud máxima

Para evitar flooding:

- Título: máximo 150 caracteres
- Contenido: máximo 3000 caracteres
- Comentarios: máximo 800 caracteres

✓ Protección contra spam automatizado

Opcionalmente se puede activar:

- verificación anti-bot
- challenge ligero tipo "rate-limit por IP"
- corrección de patrones repetitivos

10.4. Manejo de Roles

El sistema implementa un esquema básico pero eficaz de control de acceso.

Roles definidos:

Rol	Permisos
admin	Gestión total del sistema, panel de control, categorías, noticias, usuarios.
moderador (opcional)	Gestión del foro, eliminación de posts, edición de contenido.
usuario	Crear posts, comentar, editar perfil.

Asignación de roles:

La tabla usuarios incluye:

rol: string // 'admin', 'moderador', 'usuario'

Reglas de seguridad:

✓ Un usuario **no puede cambiar su propio rol** desde la interfaz.
✓ La asignación se hace manualmente desde la BD o panel admin (si se implementa).
✓ Las rutas se verifican tanto en frontend como backend.

10.5. Recomendaciones Adicionales

Aunque no obligatorias, se sugieren las siguientes prácticas:

- Forzar HTTPS en producción.
- Regenerar client secret de Discord cada cierto tiempo.
- Limitar intentos de login (si se implementa flujo propio).
- Revisar logs periódicamente en búsqueda de patrones de abuso.
- Aplicar parches de seguridad en dependencias con npm audit fix.

11. Mantenimiento y Escalabilidad

Este apartado describe cómo mantener y escalar la plataforma **Novuvitae Labs** a medida que el proyecto crece. Incluye lineamientos para agregar nuevas funcionalidades, comunidades, módulos, extensiones en la base de datos y manejo de nuevas opciones de autenticación en el futuro.

11.1. Agregar Nuevas Comunidades

El proyecto está diseñado para permitir la incorporación de nuevas comunidades temáticas, categorías o subgrupos.

Pasos recomendados:

✓ 1. Agregar nueva categoría en la BD

Insertar en categorias:

INSERT INTO categorias (nombre) VALUES ('Nueva Comunidad');

✓ 2. Crear página o vista en el frontend

Dentro de src/pages/:

- Crear archivo relacionado a la categoría (ej: /comunidad-x.astro)
- O bien usar rutas dinámicas ya implementadas si el sistema las soporta.

✓ 3. Ajustar servicios de consulta

En src/services/:

- Agregar endpoint o extender función de consulta si la comunidad requiere contenido propio.

✓ 4. Añadir componentes específicos

Solo si la comunidad requiere interfaces personalizadas, por ejemplo:

- filtros
- tabs
- módulos propios
- banners o landing

11.2. Agregar Nuevos Módulos

La arquitectura basada en **Astro + componentes independientes** hace posible extender el sistema sin romper lo existente.

Ejemplos de módulos que podrían añadirse:

- Sistema de eventos
- Tienda / marketplace
- Sistema de tickets internos
- Módulo de reportes de usuarios
- Portafolio de proyectos

Estructura sugerida al agregar módulos:

En src/:

```
modules/
  └─ nombre-modulo/
      ├─ pages/
      ├─ components/
      ├─ services/
      └─ utils/
```

Buenas prácticas:

✓ Mantener módulos aislados y reutilizables.
✓ Crear endpoints internos específicos para cada módulo.
✓ Mantener un archivo de configuración centralizado si un módulo requiere flags o toggles.
✓ Documentar nuevas rutas y dependencias.

11.3. Extender la Base de Datos

La BD está diseñada para ser ampliada sin afectar el funcionamiento base.

Pasos seguros para extender la BD:

1. Crear nueva tabla

Ejemplo:

```
CREATE TABLE eventos (
  id INT PRIMARY KEY AUTO_INCREMENT,
  titulo VARCHAR(255),
  fecha DATETIME,
  descripcion TEXT
```

);

2. Crear relaciones si aplica

Ejemplo: eventos asociados a usuarios

ALTER TABLE eventos ADD COLUMN usuario_id INT;

ALTER TABLE eventos ADD FOREIGN KEY (usuario_id) REFERENCES usuarios(id);

3. Actualizar ORM / servicios

En src/db/ o src/services/:

- Crear nuevos métodos CRUD.
- Actualizar consultas existentes si dependen de nuevas estructuras.

4. Ajustar validaciones en frontend

En formularios y páginas donde se crea/edita contenido.

5. Migraciones (si el proyecto escala)

Se recomienda integrar una herramienta de migración como:

- Prisma Migrate
- Drizzle Kit
- Sequelize CLI
- Knex migrations

11.4. Cambiar Proveedores OAuth en el Futuro

El sistema actualmente utiliza Discord OAuth2, pero es escalable hacia otros proveedores:

- Google
- GitHub
- Microsoft
- Steam
- Facebook
- OAuth genérico

Requisitos para soportar nuevos proveedores:

1. Abstraer el módulo de autenticación

Crear un servicio como:

src/services/auth/

├─ discord.js

├─ google.js

├─ github.js

└─ auth-handler.js

2. Mantener estructura consistente en la BD

Agregar campos opcionales:

proveedor: 'discord' | 'google' | 'github'

oauth_id: string

email: string? (si el proveedor lo entrega)

3. Adaptar flujo OAuth

Cada proveedor usa:

- autorización
- intercambio de código por token
- obtención de datos

Se reemplaza solo el endpoint, no el flujo completo.

4. Permitir múltiples proveedores

El login se convierte en un componente flexible:

Botones:

- **Login con Discord**
- **Login con Google**
- etc.

5. Mantener compatibilidad

La aplicación debe:

- Registrar usuarios nuevos según proveedor
- Conectar cuentas existentes mediante oauth_id
- Evitar duplicados basándose en email cuando aplique

11.5. Recomendaciones generales de mantenimiento

- Ejecutar npm update trimestralmente.
- Revisar dependencias vulnerables con:
- npm audit
- npm audit fix
- Respaldar la base de datos regularmente.
- Mantener un changelog para cada versión.
- Documentar rutas nuevas en el manual técnico.
- Revisar logs de servidor semanalmente.
- Monitorear errores con herramientas como Sentry (si se integra).

12. Respaldo y Recuperación

Este apartado describe los procedimientos para realizar respaldos seguros de la base de datos y del proyecto completo, así como los pasos necesarios para restaurar el sistema en caso de falla, corrupción, pérdida de información o migración de servidor.
El objetivo es asegurar la continuidad operativa de **Novuvitae Labs** y minimizar riesgos ante incidentes.

12.1. Respaldo de Base de Datos

La base de datos contiene los datos más sensibles del sistema: usuarios, posts, comentarios y noticias.
Se recomienda realizar **respaldos automáticos diarios** y un respaldo manual antes de cada despliegue o actualización del proyecto.

Comando para generar respaldo

Desde la terminal del servidor:

pg_dump database > respaldo.sql

Buenas prácticas del respaldo

- Guardar el archivo en una carpeta protegida (no pública).
- Mantener una política de retención (mínimo 7 días, ideal 30 días).
- Comprimir el respaldo si es muy grande:
- gzip respaldo.sql

12.2. Restauración de Base de Datos

En caso de pérdida total o parcial de la información, o durante una migración a un nuevo servidor, se puede restaurar un respaldo previamente generado.

Comando para restaurar

psql database < respaldo.sql

Recomendaciones al restaurar

- Asegurarse de que la base de datos destino esté vacía o correctamente preparada.
- Verificar permisos del usuario que realiza la restauración.
- Realizar pruebas posteriores:

- Inicio de sesión
- Creación de posts
- Panel admin
- Consultas generales

12.3. Respaldo del Proyecto (Código)

El respaldo del proyecto debe incluir únicamente los archivos fuente esenciales.

Directorios y archivos a respaldar

Elemento	Motivo
/src	Contiene todo el código del frontend, servicios, componentes y lógica del sistema.
/public	Contiene archivos estáticos, assets e imágenes necesarias para el funcionamiento del sitio.
package.json	Lista todas las dependencias y scripts del proyecto.
.env	Contiene variables privadas necesarias para el funcionamiento (no debe compartirse públicamente).

Archivos que se deben *excluir* del respaldo

- /node_modules (se regenera con npm install)
- /dist (se genera con npm run build)
- Archivos temporales o logs

Ejemplo de respaldo comprimido

zip -r respaldo_proyecto.zip src public package.json .env

12.4. Restauración del Proyecto

Para restaurar el proyecto:

1. Descomprimir o clonar los archivos del respaldo.
2. Instalar las dependencias:
3. npm install
4. Restaurar la base de datos si es necesario.
5. Verificar que el archivo .env contenga las credenciales correctas.

6. Iniciar el proyecto:
7. npm run dev

12.5. Recomendaciones Generales de Respaldo

- Mantener **dos copias**: una local y una remota (Google Drive, AWS S3, servidor FTP).
- Probar la restauración al menos una vez al mes.
- Automatizar respaldos con cron jobs si el servidor lo permite.
- Mantener copias encriptadas si contienen datos personales.

13. Anexos

Los anexos incluidos en esta sección complementan la información técnica presentada en el manual y permiten una mejor comprensión del funcionamiento interno del sistema. Su objetivo es proporcionar material visual y de referencia que facilite futuras implementaciones, auditorías, mejoras o mantenimientos del software.

13.1. Esquemas de Arquitectura

Se incluyen diagramas que representan la estructura general del sistema, permitiendo visualizar:

- **Relación entre Frontend, Backend y Base de Datos**
- **Flujo del usuario en el proceso de autenticación con Discord OAuth2**
- **Comunicación entre módulos internos del sistema y servicios externos**

Estos esquemas permiten comprender de manera macro cómo se organiza y comunica el software.

13.2. Endpoints Internos (si se agregan servicios API)

Aunque el proyecto *Novuvitae Labs* está construido principalmente sobre un frontend Astro con acciones y servicios integrados, si en el futuro se incorporan servicios API centralizados, esta sección debe documentar:

Estructura sugerida para la documentación de endpoints:

Ejemplo de formato:

Endpoint: /api/posts/create
Método: POST
Descripción: Crea un nuevo post en el foro.
Parámetros:

- titulo (string)
- contenido (string)
- categoria_id (int)
  Respuestas:
- 200 OK → Post creado exitosamente

- 400 Bad Request → Validación fallida
- 401 Unauthorized → Usuario no autenticado

Este apartado debe ampliarse conforme se agreguen nuevas APIs o servicios internos.

13.3. Diagramas de Flujo

Se incluyen diagramas de flujo que representan la lógica operativa del sistema, tales como:

Flujo de autenticación con Discord

- Inicio de sesión
- Redirección a Discord
- Obtención de token
- Verificación del usuario
- Registro/actualización en la base de datos

Flujo de creación de un post

- Recepción del formulario
- Validación de campos
- Inserción en base de datos
- Renderización del post

Flujo de publicación de noticias

- Acceso de administrador
- Registro de datos
- Validación
- Publicación en la portada

Flujo del Dashboard

- Consulta de métricas
- Procesamiento de datos
- Despliegue visual

Estos diagramas permiten visualizar procesos complejos de forma clara y sirven como base para nuevas implementaciones y para la capacitación de personal técnico.

Lista de componentes:

Componentes:

1. Arquitectura del Sistema

1. Arquitectura Cliente–Servidor
2. Separación Frontend / Backend integrada dentro de Astro.
3. Servicios internos para autenticación y lógica de negocio.
4. Integración externa con Discord OAuth2.
5. Comunicación mediante HTTP/REST entre frontend y backend.

2. Gestión de Autenticación

6. Implementación de OAuth2 con Discord (flujo Authorization Code).
7. Validación de tokens y obtención segura de datos del usuario.
8. Registro automático del usuario en la base de datos al iniciar sesión.
9. Control de sesiones y manejo de cookies seguras.
10. Protección de rutas privadas para el panel admin.

3. Base de Datos

11. Modelo de datos relacional diseñado para escalabilidad.
12. Tablas principales: Usuarios, Noticias, Categorías, Posts, Comentarios.
13. Relaciones 1:N y N:1 entre las tablas según funcionalidad del foro.
14. Integridad referencial mediante claves foráneas.
15. Consultas optimizadas para lectura y escritura frecuente.
16. Inserción automática de datos tras login con Discord.

4. Backend (Lógica del Sistema)

17. Servicios para CRUD de noticias.
18. Servicios para CRUD de posts y comentarios del foro.
19. Verificación de permisos (admin / usuario general).
20. Manejo de rutas dinámicas para contenido del foro.
21. Control de errores y respuestas estandarizadas.

5. Frontend

22. Renderizado híbrido con Astro (SSR + Islas interactivas según necesidad).
23. Vistas estructuradas por módulos (inicio, noticias, foro, admin).
24. Componentes reutilizables para posts, comentarios y tarjetas de noticias.
25. Estilos gestionados con Tailwind o CSS modular.

6. Seguridad

26. Uso de HTTPS (según entorno de despliegue).
27. Protección contra inyección SQL mediante ORM/queries sanitizadas.
28. Sanitización de contenido en posts y comentarios (XSS).
29. Uso de tokens de Discord de forma temporal y segura.
30. Validación del rol para acceso al panel administrativo.

7. Panel Administrativo

31. Dashboard con información consolidada del sistema.
32. Gestión de noticias desde interfaz dedicada.
33. Gestión básica de usuarios y contenido (si aplica).
34. Rutas del panel restringidas por autenticación + rol admin.

8. Foro

35. Sistema jerárquico de categorías, posts y comentarios.
36. Ordenamiento de posts por fecha de creación.

37. Render dinámico por URL del post.
38. Control de usuarios para creación de contenido.
39. Persistencia de datos en la base de datos.

9. Noticias

40. Creación, edición y eliminación desde panel admin.
41. Renderizado automático en la página principal.
42. Organización por fecha y categoría.
43. Editor basado en formularios simple y claro.

10. Integraciones

44. Discord OAuth2 para autenticación.
45. API de Discord para obtener avatar, nombre y email del usuario.
46. Webhook interno opcional (si se implementó).

11. Despliegue

47. Estructura preparada para deploy en hosting estático + servidor Node.
48. Variables de entorno para claves de Discord y conexión a base de datos.
49. Scripts de build y preview de Astro.

12. Documentación y Manual Técnico

50. Instrucciones de instalación.
51. Instrucciones de configuración de OAuth con Discord.
52. Guía de conexión a la base de datos.
53. Guía de rutas y endpoints internos.