Documento de arquitectura y desarrollo del sistema SID

Henry Ricaurte Mora

Abril de 2025

Índice

1. Introducción

En esta sección se brinda una visión introductoria de los conceptos relacionados con la descripción del sistema del SID. Es importante establecer una estructura clara de los componentes que lo conforman.

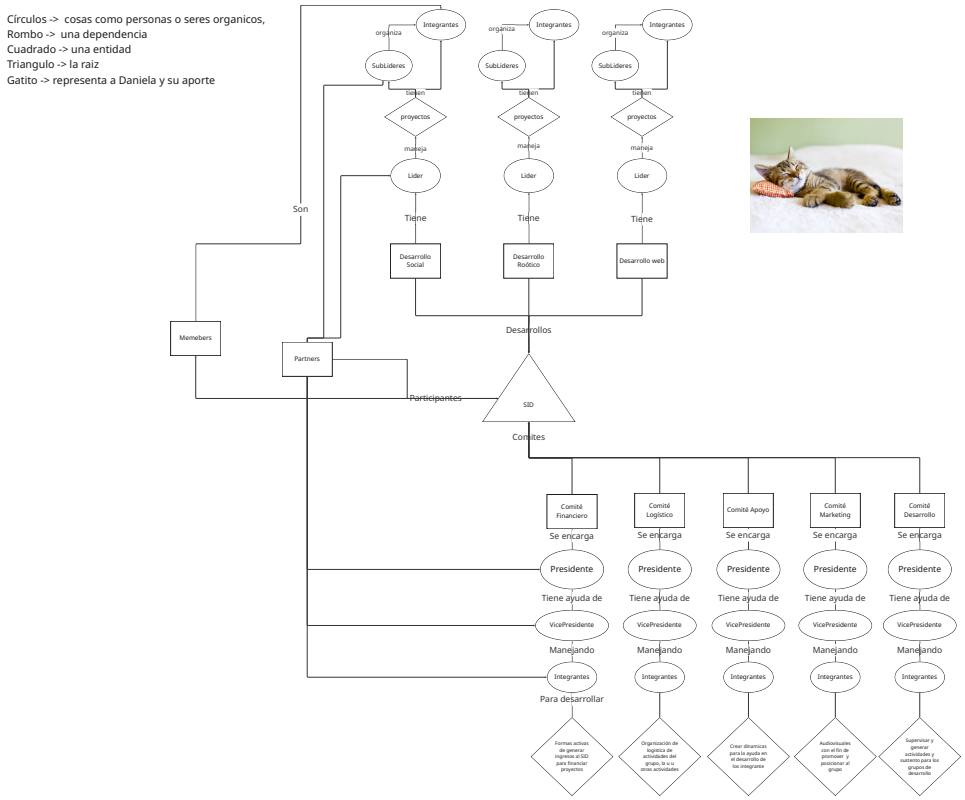

Figura 1: Diagrama de organización del SID

El SID está conformado por tres ramas principales:

- **Desarrollos:** Social, Robótico y Web.
- **Comités:** Financiera, Logística, Apoyo, Marketing y Desarrollo.
- **Participantes:** *Partners* y *Members*.

1.1. Comités

Los comités son responsables del funcionamiento del SID. Cada uno tiene un presidente y un vicepresidente, quienes lideran al equipo para desarrollar estrategias como:

- Generar ingresos para financiar proyectos.
- Organizar la logística de las actividades.
- Establecer dinámicas de formación para los integrantes.
- Crear contenido audiovisual para promoción.
- Supervisar y respaldar a los grupos de desarrollo.

1.2. Desarrollos

Cada desarrollo (Social, Robótico y Web) cuenta con un líder, sublíderes y un equipo. El líder coordina el proyecto general, los sublíderes gestionan apartados específicos y los integrantes trabajan bajo su guía.

Por ejemplo, este documento forma parte de un proyecto en el desarrollo Web.

Los *Partners* incluyen a presidentes, vicepresidentes, líderes y sublíderes. Los *Members* son integrantes que participan activamente en cada uno de los apartados de desarrollo.

2. Descripción General del Sistema

Se desarrollará una plataforma integral para la gestión y difusión de información del grupo SID. Esta incluirá herramientas para la presentación institucional, la oferta de cursos de formación y la divulgación de conocimiento especializado.

2.1. Componentes Principales

- **Portal Web Principal:** Será el sitio informativo del grupo, donde se mostrará quiénes somos y cuáles son nuestras áreas de trabajo. Este portal incluirá secciones dedicadas a los distintos desarrollos del grupo, cada una con su propia página que detallará a sus integrantes, proyectos y líderes. Asimismo, se presentarán los comités con una estructura similar. También se incluirá información general sobre el SID, sus actividades, proyectos, equipos, colaboraciones, cursos, noticias e integrantes, además de un formulario de inscripción para nuevos miembros o interesados.
- **Panel Administrativo:** Permitirá la gestión del contenido visible en el portal principal, así como la administración de los datos de los usuarios provenientes de los formularios de inscripción.
- **Sistema de Gestión Académica:** Este componente permitirá la inscripción y administración de los cursos ofrecidos. Incluirá funcionalidades específicas para tres tipos de usuarios: administrador, profesor y estudiante/usuario. También facilitará la asignación y gestión de profesores por curso.
- **SID Academy:** Proyecto futuro que se plantea como una plataforma educativa de cursos breves, similar a QtAcademy, con el objetivo de fortalecer el aprendizaje especializado dentro de la comunidad.

3. Requerimientos del Sistema

3.1. Requerimientos Funcionales:

3.1.1. Portal Web Principal

RF-001 El sistema debe mostrar información institucional del grupo SID.

RF-002 El sistema debe presentar secciones para cada desarrollo del grupo (Web, Robótica, Social) con sus integrantes, proyectos y líderes.

RF-003 El sistema debe mostrar información sobre los comités existentes.

RF-004 El sistema debe listar actividades realizadas por el grupo.

RF-005 El sistema debe mostrar proyectos desarrollados por el grupo.

RF-006 El sistema debe exponer los cursos ofrecidos por el grupo.

RF-007 El sistema debe presentar los equipos de trabajo.

RF-008 El sistema debe publicar noticias relacionadas con el grupo.

RF-009 El sistema debe mostrar información sobre fundaciones y colaboraciones.

RF-010 El sistema debe incluir un formulario de inscripción para nuevos miembros.

3.1.2. Panel Administrativo (Dashboard)

RF-011 El sistema debe permitir a los administradores gestionar el contenido del portal web.

RF-012 El sistema debe proporcionar funcionalidades para administrar los datos de usuarios.

RF-013 El sistema debe permitir la gestión de actividades (crear, modificar, eliminar).

RF-014 El sistema debe permitir la gestión de proyectos (crear, modificar, eliminar).

RF-015 El sistema debe permitir la gestión de cursos (crear, modificar, eliminar).

RF-016 El sistema debe permitir la gestión de equipos (crear, modificar, eliminar).

RF-017 El sistema debe permitir la gestión de noticias (crear, modificar, eliminar).

RF-018 El sistema debe permitir la gestión de fundaciones (crear, modificar, eliminar).

RF-019 El sistema debe implementar control de acceso basado en roles.

RF-020 El sistema debe permitir la administración de permisos por rol.

3.1.3. API de Servicios

RF-021 El sistema debe proporcionar endpoints RESTful para listar contenido (actividades, proyectos, etc.).

RF-022 El sistema debe implementar paginación en todos los endpoints que devuelven listas.

RF-023 El sistema debe manejar formatos estándar para solicitudes y respuestas (JSON).

RF-024 El sistema debe implementar autenticación mediante tokens JWT.

RF-025 El sistema debe proporcionar endpoints para la gestión de contenidos protegidos por autenticación.

3.2. Requerimientos No Funcionales

3.2.1. Seguridad

RF-026 El sistema debe implementar autenticación OAuth 2.0 con proveedores de identidad externos (Microsoft Entra ID/Azure AD / Google).

RF-027 El sistema debe validar y procesar tokens JWT emitidos por el proveedor de identidad.

RF-028 El sistema debe implementar control de acceso basado en roles mediante la base de datos

RF-029 El sistema debe proteger los endpoints administrativos con autorización basada en roles

RF-030 El sistema debe usar HTTPS para todas las comunicaciones (TLS 1.2+). | Toca entender que es a profundidad

RF-031 El sistema debe implementar protección contra ataques comunes (CSRF, XSS, SQL Injection).

3.2.2. Rendimiento

RF-032 El sistema debe responder a las solicitudes API en menos de 1 segundo bajo carga normal.

RF-033 El sistema debe soportar al menos 100 usuarios concurrentes.

RF-034 Las consultas a la base de datos deben optimizarse para minimizar el tiempo de respuesta.

3.2.3. Escalabilidad

RF-035 La arquitectura debe permitir escalar componentes individuales según demanda.

RF-036 El sistema debe ser compatible con despliegue en contenedores (Docker/Kubernetes).

3.2.4. Mantenibilidad

RF-037 El código debe seguir los principios SOLID y patrones de diseño apropiados.

RF-038 El sistema debe implementar logging estructurado (JSON) para diagnóstico.

RF-039 El código debe incluir documentación JavaDoc y OpenAPI para las APIs.

3.2.5. Compatibilidad

RF-040 Las APIs deben seguir el estándar RESTful nivel 2 de Richardson.

RF-041 El sistema debe proporcionar documentación interactiva mediante Swagger UI.

3.2.6. Interoperabilidad

RF-044 El sistema debe exponer APIs con versionado semántico (v1, v2). | Acordar si mandarlo en header o en la api. (obligatorio tag github)

RF-045 Las integraciones deben usar protocolos estándar (OAuth 2.0, OpenID Connect).

4. Especificación Modulos:

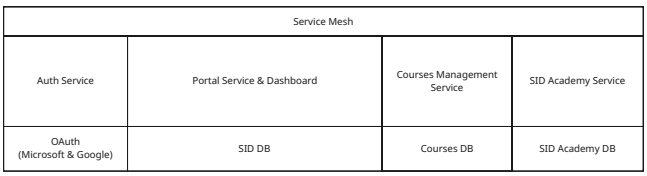

Figura 2: Diagrama de microservicios del proyecto

Modulos propuestos

4.1. Servicio de Autenticación y Usuarios

- Gestión completa del ciclo de vida de usuarios
- Autenticación y autorización
- Gestión de tokens y sesiones

4.2. Servicio de Estructura Organizacional

- Visualización de desarrollos (Web, Robótica, etc.)
- Visualización de comités (Financiero, Social, etc.)
- Visualización de noticias y actualizaciones
- Visualización de información sobre fundaciones y trabajos realizados
- Visualización de integrantes y Formulario
- Administración y gestión de desarrollos (Web, Robótica, etc.)
- Administración y gestión de comités (Financiero, Social, etc.)
- Administración y gestión de noticias y actualizaciones
- Administración y gestión de información sobre fundaciones y trabajos realizados
- Administración y gestión de integrantes
- Administracion de integrantes mediante Formulario.

4.3. Servicio de Gestión Académica

- Catálogo e inscripción a cursos
- Gestión de profesores y alumnos
- Seguimiento y evaluación

4.4. Servicio QtAcademy (futuro)

- Gestión de contenidos educativos cortos
- Sistema de aprobación y curación
- Análisis de métricas de consumo

Módulo	Función principal	Métodos API	Seguridad
AuthService	Autenticación y autorización Gestión de usuarios y roles Administración de tokens y sesiones	▪ POST ▪ GET ▪ PUT	JWT (Solo login es público)
PortalService y Dashboard control	Visualización pública del SID (desarrollos, comités, noticias) Administración de desarrollos Gestión de comités y fundaciones	▪ GET ▪ POST ▪ PUT ▪ PATCH ▪ DELETE	Mixto: • Público (lectura) • JWT + roles (escritura)
CoursesService	Gestión de cursos e inscripciones Administración de profesores/alumnos Seguimiento y evaluación académica	▪ GET ▪ POST ▪ PUT	JWT + roles (basado en perfiles académicos)
SIDAcademy *(futuro)*	Contenidos educativos cortos Curación de material didáctico Análisis de métricas de aprendizaje	▪ GET ▪ POST ▪ PUT	JWT + roles (según nivel de acceso)

Cuadro 1: Arquitectura de modulos propuesta para el sistema SID

5. Arquitectura de Base de Datos:

Como base de datos se elige una NewSQL como Neon la cual opera con postgresql, elegido por comidad de diseño y ademas brinda lo necesario para el proyecto. La base de datos generada es:

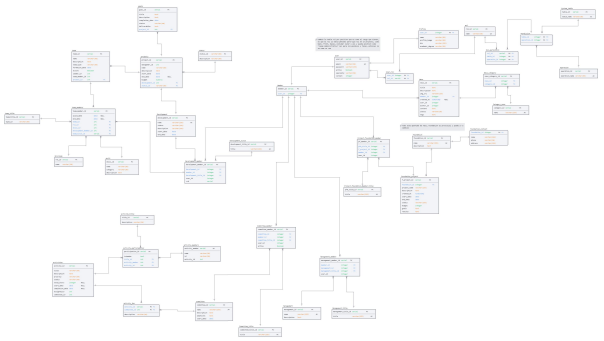

Figura 3: Diagrama de base de datos relacional

6. Arquitectura de modulos y microservicios:

Vamos a abordar todo el tema de los módulos y microservicios ademas de su arquitectura paso por paso. El único microservicio actual es AuthService, luego estan los módulos monoliticos como el **Portal Content Service**

7. Portal Content Service

Este servicio de portal de contenido que se generará es la dicha "Página del SID", desde el lado del backend tenemos que suplir información relevante, esta información está dividida en:

- Actividades
- Proyectos
- Cursos
- Equipos
- Noticias
- Aliados | Fundaciones

7.1. Vision General

El diseño del sistema sigue un patrón de arquitectura estándar de múltiples capas comúnmente utilizado en aplicaciones de SpringBoot. El sistema está diseñado para mantener el desacoplamiento de funciones para garantizar la mantenibilidad, la comprobabilidad (Testing) y la escalabilidad

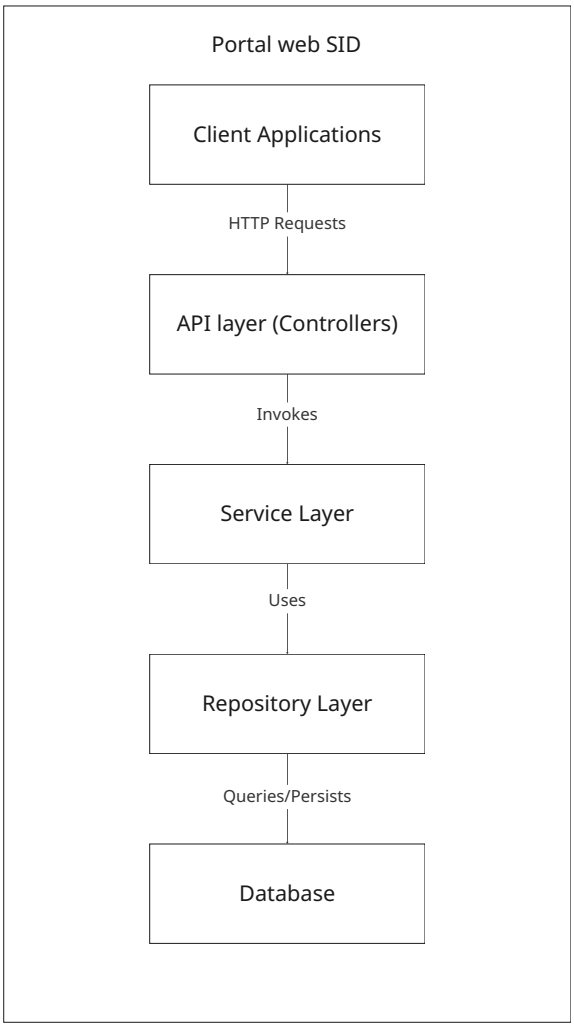

Figura 4: Diagrama de la arquitectura por capas

Capa	**Descripción**	**Responsabilidades principales**
Capa API	Punto de entrada para las solicitudes del cliente	Manejo de solicitudes, formateo de respuestas, validación de entradas
Capa de Servicio	Contiene la lógica de la aplicación	Orquesta operaciones, maneja transacciones y logica de la aplicación
Capa de Repositorio	Gestiona el acceso a los datos	Proporciona una abstracción sobre el almacenamiento, maneja operaciones CRUD
Capa de Dominio	Contiene las entidades del dominio	Encapsula los datos y la lógica del negocio

Cuadro 2: Responsabilidades por capa del sistema

7.2. Capa API (Controladores)

La capa API es responsable de manejar las solicitudes HTTP entrantes, validar los datos de entrada y devolver respuestas apropiadas. Esta capa está compuesta por clases controladoras que definen los endpoints REST expuestos por la aplicación.

Los controladores siguen la convención REST tanto para el nombramiento de endpoints como para el uso de métodos HTTP:

- **GET** para obtener recursos
- **POST** para crear recursos
- **PUT** para actualizar recursos completamente
- **PATCH** para actualizar parcialmente recursos
- **DELETE** para eliminar recursos

7.3. Capa de Servicio

La capa de servicio implementa la lógica central de la aplicación. Se ubica entre los controladores y los repositorios, orquestando operaciones entre múltiples entidades del dominio y aplicando las reglas de negocio.

7.4. Capa de Repositorio

La capa de repositorio proporciona una abstracción sobre el mecanismo de acceso a datos. Utiliza Spring Data JPA para simplificar las operaciones con la base de datos y reducir el código repetitivo. También en los casos necesarios utiliza el patrón repositorio para hacer llamados directos a la base de datos de ser necesario

7.5. Capa de Dominio

La capa de dominio contiene las clases de entidad que representan el dominio del negocio. Estas clases son aquellas que sirven para encapsular la logica del negocio, y se pasa a service para ser manejada.

7.6. Capa de Entidad

La capa de entidad proporciona aquellas clases que representan la abstracción más cercana a la base de datos. Son quienes se usaran mediante los JPA para hacer llamados directos a la base de datos y tambien usada en los repositorios manuales

7.7. Flujo de peticiones

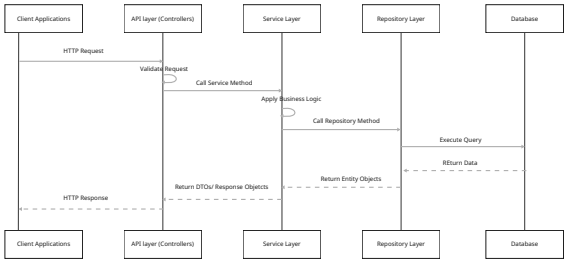

Figura 5: Diagrama de el flujo de las petciones por capas

7.8. Endpoints REST

Con base en los elementos anteriores, definimos los siguientes endpoints REST con paginación de Spring Boot:

7.8.1. Actividades

7.8.2. Proyectos

7.8.3. Cursos

7.8.4. Equipos

7.8.5. Noticias

GET Todos (Paginado)

GET Noticias (paginado)

```
GET /api/news?page=0&size=10&sort=publishDate,desc
```

Estructura de respuesta

```
{
    "content": [
        {
            "id": integer,                  // ID unico de la noticia
            "title": string,                // Titulo de la noticia
            "content": string(HTML),        // Contenido completo en HTML
            "publishDate": string(ISO-8601), // Fecha de publicacion
            "author": string,               // Nombre del autor
            "imageUrl": string(URL),        // Ruta a la imagen destacada
            "tags": string[]                // Array de etiquetas
        }
        // ... mas elementos
    ],
    "pageable": {
        "sort": object,       // Informacion de ordenamiento
        "pageSize": integer,  // Tamano de pagina solicitado
        "pageNumber": integer // Numero de pagina actual
        // ... mas metadatos de paginacion
    },
    "totalElements": integer, // Total de resultados
    "totalPages": integer,    // Total de paginas disponibles
    "first": boolean,         // Si es la primera pagina
    "last": boolean,          // Si es la ultima pagina
    "size": integer           // Tamano de la pagina actual
}
```

GET Individual

GET /api/news/id - Obtener noticia específica

```
GET /api/news/{id}
```

Estructura de respuesta

```
{
    "id": integer,                    // ID unico de la noticia
    "title": string,                  // Titulo de la noticia
    "content": string(HTML),          // Contenido completo en HTML
    "publishDate": string(ISO-8601), // Fecha de publicacion
    "author": string,                 // Nombre del autor
    "imageUrl": string(URL),          // Ruta a la imagen destacada
    "tags": string[]                  // Array de etiquetas
}
```

POST Creación

POST /api/news - Crear nueva noticia

```
POST /api/news
Content-Type: application/json
Authorization: Bearer <token>

{
    "title": string,       // Requerido: Titulo de la noticia
    "content": string(URL:.MD) // Requerido: Ruta para contenido posiblemente
        HTML o .MD , en su defecto el contenido en un string
    "tags": string[]  //  Array de etiquetas
    "imageURL": String(URL) // Ruta a la Imagen
}
```

Estructura de respuesta

```
{
    "id": integer,                    // ID generado automaticamente
    "title": string,                  // Titulo proporcionado
    "status": string(enum),           // Estado inicial: DRAFT
    "createdAt": string(ISO-8601)     // Timestamp de creacion
}
```

PUT (Reemplazo completo)

PUT /api/news/id - Actualizar noticia completa

```
PUT /api/news/{id}
Content-Type: application/json
Authorization: Bearer <token>

{
    "title": string,        // Requerido: Nuevo titulo
    "content": string(HTML) // Requerido: Nuevo contenido
    "imageURL": string(URL) // Ruta de la nueva imagen
    // Todos los campos son requeridos en PUT ,
    // se pueden dejar en blanco si no se quiere actualizar ese apartado
}
```

Estructura de respuesta

```
{
    "id": integer,                  // ID de la noticia actualizada
    "title": string,                // Titulo actualizado
    "lastUpdated": string(ISO-8601)  // Timestamp de actualizacion
}
```

PATCH (Actualización parcial)

PATCH /api/news/id - Actualizar campos específicos

```
PATCH /api/news/{id}
Content-Type: application/json-patch+json
Authorization: Bearer <token>

[
    { "op": string(enum),
      "path": string,
      "value": any
    }, // Operacion JSON Patch

    // Operaciones permitidas: "replace", "add", "remove"
    // Ejemplo: { "op": "replace", "path": "/title", "value": "Nuevo titulo" }
]
```

Estructura de respuesta

```
{
    "id": integer,     // ID de la noticia modificada
    "changes": string[], // Lista de campos modificados
    "version": integer   // Nueva version del documento
}
```

DELETE

DELETE /api/news/id - Eliminar noticia

```
DELETE /api/news/{id}
Authorization: Bearer <token>
```

Respuesta

```
HTTP/1.1 204 No Content
// No se devuelve cuerpo de respuesta
```

7.8.6. Aliados y Fundaciones

GET Todos (Paginado)

GET Aliados (paginado)

```
GET /api/allies?page=0&size=10&type=FOUNDATION
```

Estructura de respuesta

```
{
    "content": [
        {
            "id": integer,                    // ID unico del aliado
            "name": string,                   // Nombre de la organizacion
            "type": string(enum),             // Tipo: FOUNDATION, COMPANY, NGO,
    etc.
            "description": string,            // Descripcion corta
            "logoUrl": string(URL),           // Ruta al logo
            "website": string(URL),           // Sitio web
            "partnerSince": string(ISO-8601) // Fecha de inicio de alianza
        }
        // ... mas elementos
    ],
    "pageable": {
        "sort": object,       // Informacion de ordenamiento
        "pageSize": integer,  // Tamanio de pagina solicitado
        "pageNumber": integer // Numero de pagina actual
        // ... mas metadatos de paginacion
    },
    "totalElements": integer, // Total de resultados
    "totalPages": integer,    // Total de paginas disponibles
    "first": boolean,         // Si es la primera pagina
    "last": boolean           // Si es la ultima pagina
}
```

GET Individual

GET /api/allies/id - Obtener aliado específico

```
GET /api/allies/{id}
```

Estructura de respuesta

```
{
    "id": integer,                    // ID unico del aliado
    "name": string,                   // Nombre de la organizacion
    "type": string(enum),             // Tipo: FOUNDATION, COMPANY, NGO, etc.
    "description": string,            // Descripcion corta
    "logoUrl": string(URL),           // Ruta al logo
    "website": string(URL),           // Sitio web
    "partnerSince": string(ISO-8601), // Fecha de inicio de alianza
    "contactInfo": {                  // Informacion de contacto
        "email": string(email),
        "phone": string
    },
    "projects": [                     // Proyectos asociados
        {
            "id": integer,
            "name": string,
            "status": string(enum)    // ACTIVE, COMPLETED, PLANNED
        }
    ],
    "socialMedia": {                  // Redes sociales
        "facebook": string(URL),
        "twitter": string(URL),
        "linkedin": string(URL)
    }
}
```

POST Creación

POST /api/allies - Crear nuevo aliado

```
POST /api/allies
Content-Type: application/json
Authorization: Bearer <token>

{
    "name": string,                 // Requerido: Nombre de la organizacion
    "type": string(enum),           // Requerido: FOUNDATION, COMPANY, NGO, etc.
    "description": string,          // Requerido: Descripcion
    "website": string(URL),         // Opcional: Sitio web
    "contactInfo": {                // Opcional: Informacion de contacto
        "email": string(email),
        "phone": string
    },
    "socialMedia": {                // Opcional: Redes sociales
        "facebook": string(URL),
        "twitter": string(URL),
        "linkedin": string(URL)
    }
}
```

Estructura de respuesta

```
{
    "id": integer,                  // ID generado automaticamente
    "name": string,                 // Nombre proporcionado
    "type": string(enum),           // Tipo proporcionado
    "createdAt": string(ISO-8601),  // Timestamp de creacion
    "status": string(enum)          // Estado inicial: PENDING_REVIEW
}
```

PUT (Reemplazo completo)

PUT /api/allies/id - Actualizar aliado completo

```
PUT /api/allies/{id}
Content-Type: application/json
Authorization: Bearer <token>

{
    "name": string,                  // Requerido: Nombre de la organizacion
    "type": string(enum),            // Requerido: FOUNDATION, COMPANY, NGO, etc.
    "description": string,           // Requerido: Descripcion
    "website": string(URL),          // Opcional: Sitio web
    "contactInfo": {                 // Opcional: Informacion de contacto
        "email": string(email),
        "phone": string
    },
    "socialMedia": {                 // Opcional: Redes sociales
        "facebook": string(URL),
        "twitter": string(URL),
        "linkedin": string(URL)
    }
}
```

Estructura de respuesta

```
{
    "id": integer,                   // ID del aliado actualizado
    "name": string,                  // Nombre actualizado
    "lastUpdated": string(ISO-8601), // Timestamp de actualizacion
    "status": string(enum)           // Estado: ACTIVE, PENDING_REVIEW, etc.
}
```

PATCH (Actualización parcial)

PATCH /api/allies/id - Actualizar campos específicos

```
PATCH /api/allies/{id}
Content-Type: application/json-patch+json
Authorization: Bearer <token>

[
    { "op": string(enum), "path": string, "value": any }, // Operacion JSON Patch
    // Operaciones permitidas: "replace", "add", "remove"
    // Ejemplo: { "op": "replace", "path": "/description", "value": "Nueva
        descripcion" }
]
```

Estructura de respuesta

```
{
    "id": integer,         // ID del aliado modificado
    "changes": string[],   // Lista de campos modificados
    "version": integer,    // Nueva version del documento
    "modifiedAt": string(ISO-8601) // Timestamp de modificacion
}
```

DELETE

DELETE /api/allies/id - Eliminar aliado

```
DELETE /api/allies/{id}
Authorization: Bearer <token>
```

Respuesta

```
HTTP/1.1 204 No Content
// No se devuelve cuerpo de respuesta
```