Arquitectura para CoProof

Prefijo base para todos los endpoints: `/api/v1`

1. Función: Gestión de Usuarios, Autenticación y Acceso

Responsable de la seguridad, perfiles y sesiones de usuario. Se asume un sistema de autenticación basado en Tokens JWT.

Método	Endpoint	Descripción	Casos de Uso
POST	/auth/register	**Crear una cuenta.** Recibe nombre, correo y contraseña. Inicia el proceso de verificación de correo electrónico.	CDU-06
POST	/auth/verify-email	**Verificar Correo Electrónico.** Recibe un token de verificación enviado al correo del usuario para activar la cuenta.	RF-019
POST	/auth/login	**Iniciar Sesión.** Autentica al usuario con correo y contraseña, y devuelve un token JWT para autorizar futuras solicitudes.	CDU-05
POST	/auth/logout	**Cerrar Sesión.** Invalida el token de sesión del usuario en el servidor (refresh tokens).	CDU-25
POST	/auth/forgot-password	**Recuperar Contraseña.** Recibe el correo del usuario	RF-018

Método	Endpoint	Descripción	Casos de Uso
		y le envía un enlace para restablecer su contraseña.	
POST	/auth/reset-password	**Restablecer Contraseña.** Recibe un token de restablecimiento y la nueva contraseña.	RF-018
GET	/users/me	**Obtener Perfil de Usuario.** Devuelve la información del usuario autenticado actualmente (nombre, correo, etc.).	CDU-26
PATCH	/users/me	**Actualizar Perfil de Usuario.** Permite al usuario modificar su información personal, como su nombre.	CDU-26
PATCH	/users/me/password	**Cambiar Contraseña.** Permite al usuario autenticado cambiar su contraseña actual por una nueva.	RF-064
GET	/users/me/settings	**Obtener Configuración del Entorno.** Devuelve las preferencias del usuario (idioma, tema, configuración de IA).	CDU-27
PATCH	/users/me/settings	**Actualizar Configuración del Entorno.** Permite al	CDU-27

Método	Endpoint	Descripción	Casos de Uso
		usuario guardar sus preferencias de entorno y procesamiento.	

2. Función: Entorno de Trabajo, Proyectos y Colaboración

Gestiona la creación, acceso, modificación y colaboración en proyectos.

Método	Endpoint	Descripción	Casos de Uso
POST	/projects	**Crear un Proyecto.** Crea un nuevo proyecto público o privado, asignando al creador como líder.	CDU-07, CDU-08
GET	/projects/{projectId}	**Obtener Detalles del Proyecto.** Devuelve la información principal de un proyecto (metadata, líder, etc.) para abrirlo.	CDU-10, CDU-11
GET	/projects/{projectId}/graph	**Obtener Grafo del Proyecto.** Carga y devuelve la estructura completa del grafo de un proyecto (nodos y dependencias).	CDU-15
GET	/projects/{projectId}/premises	**Obtener Premisas Globales.** Consulta la base axiomática (definiciones, axiomas) de un proyecto.	RF-039

Método	Endpoint	Descripción	Casos de Uso
PATCH	/projects/{projectId}/premises	**Modificar Premisas Globales.** Permite a un usuario con permisos modificar las premisas. Genera una solicitud de autorización.	CDU-14
POST	/projects/{projectId}/collaborators	**Invitar Colaborador.** Permite al líder de un proyecto privado invitar a otros usuarios a colaborar.	RF-023
DELETE	/projects/{projectId}/collaborators/{userId}	**Eliminar Colaborador.** Permite al líder de un proyecto eliminar a un colaborador.	RF-023
POST	/projects/{projectId}/save-request	**Solicitar Guardado de Cambios.** Envía los cambios de una sesión (individual o colaborativa) para su aprobación por el líder.	CDU-12, CDU-13
GET	/projects/{projectId}/requests	**Consultar Solicitudes de Cambios.** Permite al líder del proyecto ver la lista de solicitudes de autorización pendientes.	RF-060
POST	/projects/{projectId}/requests/{requestId}/authorize	**Autorizar Cambios.** Permite al líder aprobar una solicitud de cambio, integrando las	CDU-24

Método	Endpoint	Descripción	Casos de Uso
		modificaciones al proyecto.	
POST	/projects/{projectId}/requests/{requestId}/reject	**Rechazar Cambios.** Permite al líder rechazar una solicitud de cambio, notificando al colaborador.	CDU-24
PATCH	/projects/{projectId}/nodes/{nodeId}	**Modificar un Nodo.** Permite realizar cambios en un nodo (proveer demostración, plan, datos numéricos). Genera una solicitud de autorización.	CDU-16, CDU-18, CDU-20

Colaboración en Tiempo Real (CDU-11, RF-033): La sincronización en tiempo real se manejará a través de WebSockets. Desde el cliente de Angular estableciendo una conexión a un endpoint como:

- wss://api.coproof.com/v1/projects/{projectId}/collaborate A través de esta conexión se envían eventos como "edición de nodo", "movimiento del cursor", "nuevo colaborador", etc.

3. Función: Procesamiento y Verificación de Demostraciones

Encapsula la ingesta y validación de demostraciones externas.

Método	Endpoint	Descripción	Casos de Uso
POST	/proofs/upload-external	**Cargar Demostración Externa.** Sube un archivo (PDF, LaTeX, imagen) para su análisis. Inicia un	CDU-01, CDU-02

Método	Endpoint	Descripción	Casos de Uso
		trabajo asíncrono y devuelve un `jobId`.	
GET	`/jobs/{jobId}`	**Consultar Estado de un Trabajo.** Verifica el estado de un trabajo de larga duración (validación, traducción, IA, etc.).	N/A
GET	`/jobs/{jobId}/result`	**Obtener Resultado de un Trabajo.** Obtiene el resultado final de la validación o traducción (e.g., código Lean, estado de validez).	`CDU-01`, `CDU-02`
POST	`/projects/{projectId}/import-external`	**Integrar Demostración Externa.** Agrega una demostración externa ya validada (usando su `jobId`) a un proyecto activo.	`RF-006`

4. Función: Asistencia por Inteligencia Artificial (Agentes CoProof)

Gestiona las interacciones con los agentes de IA para asistir en las demostraciones.

Método	Endpoint	Descripción	Casos de Uso
POST	`/projects/{projectId}/nodes/{nodeId}/agent/generate-proof`	**Solicitar Demostración a Agente.** Pide a un agente de IA que genere una demostración	`CDU-17`

Método	Endpoint	Descripción	Casos de Uso
		completa para un nodo. Es asíncrono.	
POST	`/projects/{projectId}/nodes/{nodeId}/agent/generate-plan`	**Solicitar Plan de Demostración a Agente.** Pide a un agente que proponga un plan (pasos) para demostrar un nodo. Es asíncrono.	CDU-19
POST	`/projects/{projectId}/nodes/{nodeId}/agent/explore-logical`	**Solicitar Exploración Lógica.** Pide a un agente que genere y analice casos pequeños para un nodo. Es asíncrono.	CDU-22

5. Función: Cómputo Distribuido y Análisis Numérico (Clúster)

Orquesta tareas computacionalmente intensivas en el clúster.

Método	Endpoint	Descripción	Casos de Uso
POST	`/projects/{projectId}/nodes/{nodeId}/cluster/run-experiment`	**Solicitar Demostración Numérica.** Pide al sistema diseñar y ejecutar un experimento en el clúster para validar un nodo numérico. Es asíncrono.	CDU-21
POST	`/projects/{projectId}/nodes/{nodeId}/cluster/explore-numerical`	**Solicitar Exploración Numérica Orientada.** Pide al	CDU-23

Método	Endpoint	Descripción	Casos de Uso
		sistema una exploración numérica con parámetros específicos. Es asíncrono.	

6. Función: Consulta, Visualización y Trazabilidad de Conocimiento

Endpoints para buscar, visualizar y exportar el conocimiento almacenado en el sistema.

Método	Endpoint	Descripción	Casos de Uso
GET	`/projects/public`	**Consultar Proyectos Públicos.** Devuelve una lista paginada de todos los proyectos públicos. Permite filtrar por área, líder, etc. (`?area=...`).	CDU-09
GET	`/search/projects`	**Buscar Proyectos Públicos.** Permite buscar proyectos por nombre o palabra clave (`?q=...`).	RF-028
GET	`/search/proofs`	**Buscar Demostraciones Internas.** Busca teoremas/lemas en todos los proyectos accesibles por nombre, autor, etc. (`?q=...`).	CDU-03
GET	`/proofs/{proofId}`	**Consultar Demostración Interna.** Devuelve el	CDU-03

Método	Endpoint	Descripción	Casos de Uso
		contenido de un teorema/lema específico, tanto en lenguaje natural como en Lean.	
GET	/proofs/{proofId}/lineage	**Consultar Linaje de un Resultado.** Devuelve la estructura de datos para visualizar qué otros resultados usan este teorema y cuáles usa él.	CDU-04
GET	/proofs/{proofId}/export	**Exportar Demostración o Linaje.** Permite descargar una demostración o su linaje en diferentes formatos (?format=pdf, ?format=json).	RF-011, RF-014

Esquema Entidad-Relación para CoProof

1. Entidad: **Usuario**

Almacena la información de cada persona registrada en el sistema.

- **Propiedades:**
  - `id` (UUID, Clave Primaria): Identificador único para el usuario.
  - `nombre` (String): Nombre completo del usuario.
  - `correo` (String, Único): Dirección de correo electrónico, usada para el inicio de sesión.
  - `hash_contrasena` (String): Hash de la contraseña del usuario (nunca se almacena en texto plano).
  - `correo_verificado` (Boolean): Indica si el usuario ha verificado su dirección de correo.
  - `token_verificacion` (String, Nullable): Token temporal para la verificación de correo.
  - `token_reseteo_pass` (String, Nullable): Token temporal para el restablecimiento de contraseña.
  - `fecha_expiracion_token` (DateTime, Nullable): Fecha de caducidad de los tokens temporales.
  - `fecha_creacion` (DateTime): Fecha y hora de creación de la cuenta.
- **Relaciones:**
  - **lidera** 1 `--` * **Proyecto**: Un usuario puede ser líder de muchos proyectos.
  - **colabora en** * `--` * **Proyecto** (a través de la tabla `Colaborador`): Un usuario puede colaborar en múltiples proyectos y un proyecto puede tener múltiples colaboradores.
  - **posee** 1 `--` 1 **ConfiguracionUsuario**: Cada usuario tiene una única configuración de entorno.
  - **crea** 1 `--` * **SolicitudCambio**: Un usuario puede crear múltiples solicitudes de cambio.
  - **sube** 1 `--` * **DemostracionExterna**: Un usuario puede subir múltiples demostraciones externas.
  - **inicia** 1 `--` * **TrabajoAsincrono**: Un usuario puede iniciar múltiples trabajos asíncronos (IA, Clúster, etc.).

2. Entidad: **Proyecto**

Representa un espacio de trabajo para una o más demostraciones, con su propio grafo y configuración.

- **Propiedades:**
    - `id` (UUID, Clave Primaria): Identificador único para el proyecto.
    - `nombre` (String): Nombre del proyecto.
    - `descripcion` (Text): Descripción detallada del objetivo del proyecto.
    - `visibilidad` (Enum: `publico`, `privado`): Define si el proyecto es accesible para todos o solo para colaboradores.
    - `id_lider` (UUID, Clave Foránea -> Usuario.id): Identificador del usuario que lidera el proyecto.
    - `fecha_creacion` (DateTime): Fecha y hora de creación del proyecto.
- **Relaciones:**
    - **es liderado por** `*` `--` 1 **Usuario**: Muchos proyectos pueden ser liderados por un mismo usuario.
    - **tiene** `*` `--` `*` **Usuario** (a través de la tabla `Colaborador`): Relación de colaboración.
    - **contiene** 1 `--` `*` **NodoGrafo**: Un proyecto está compuesto por muchos nodos en su grafo.
    - **contiene** 1 `--` `*` **PremisaGlobal**: Un proyecto tiene un conjunto de premisas globales.
    - **recibe** 1 `--` `*` **SolicitudCambio**: Un proyecto puede tener muchas solicitudes de cambio pendientes o resueltas.

3. Entidad: **Colaborador** (Tabla de Unión)

Modela la relación muchos-a-muchos entre Usuarios y Proyectos.

- **Propiedades:**
    - `id_usuario` (UUID, Clave Primaria Compuesta, Clave Foránea -> Usuario.id): ID del usuario colaborador.
    - `id_proyecto` (UUID, Clave Primaria Compuesta, Clave Foránea -> Proyecto.id): ID del proyecto en el que colabora.
    - `fecha_invitacion` (DateTime): Fecha en que el usuario fue agregado al proyecto.
- **Relaciones:**
    - **se refiere a** `*` `--` 1 **Usuario**: Muchas entradas de colaboración pueden referirse al mismo usuario.
    - **pertenece a** `*` `--` 1 **Proyecto**: Muchas entradas de colaboración pertenecen al mismo proyecto.

4. Entidad: **NodoGrafo**

Representa un nodo en el grafo de demostración de un proyecto (teorema, lema, meta, etc.).

- **Propiedades:**
  - `id` (UUID, Clave Primaria): Identificador único del nodo.
  - `id_proyecto` (UUID, Clave Foránea -> Proyecto.id): Proyecto al que pertenece el nodo.
  - `titulo` (String): Título o nombre del resultado (e.g., "Teorema de Pitágoras").
  - `enunciado_nl` (Text): El enunciado del teorema en lenguaje natural.
  - `codigo_lean` (Text, Nullable): La representación formal del enunciado y su demostración en Lean.
  - `estado` (Enum: `pendiente`, `en_revision`, `demostrado`, `error`): Estado actual del nodo dentro del flujo de trabajo.
  - `tipo` (Enum: `meta_global`, `teorema`, `lema`, `corolario`, `evaluacion_numerica`): El tipo de resultado que representa el nodo.
  - `datos_numericos` (JSON, Nullable): Para almacenar resultados de demostraciones numéricas.
- **Relaciones:**
  - **pertenece a** `*--1` **Proyecto**: Muchos nodos pertenecen a un único proyecto.
  - **depende de** `*--*` **NodoGrafo** (a través de la tabla `DependenciaNodo`): Un nodo puede depender de varios nodos (prerrequisitos) y varios nodos pueden depender de él (linaje).

5. Entidad: **DependenciaNodo** (Tabla de Unión)

Define las aristas del grafo, representando el linaje y las dependencias entre nodos.

- **Propiedades:**
  - `id_nodo_dependiente` (UUID, Clave Primaria Compuesta, Clave Foránea -> NodoGrafo.id): El nodo que utiliza otro como prerrequisito.
  - `id_nodo_prerequisito` (UUID, Clave Primaria Compuesta, Clave Foránea -> NodoGrafo.id): El nodo que es requerido por otro para su demostración.
- **Relaciones:**
  - Define la relación `*--*` autorreferencial de la entidad **NodoGrafo**.

6. Entidad: **SolicitudCambio**

Registra una solicitud de un colaborador para modificar un proyecto, la cual debe ser aprobada por el líder.

- **Propiedades:**
  - `id` (UUID, Clave Primaria): Identificador único de la solicitud.
  - `id_proyecto` (UUID, Clave Foránea -> Proyecto.id): Proyecto al que aplica la solicitud.
  - `id_solicitante` (UUID, Clave Foránea -> Usuario.id): Usuario que generó la solicitud.
  - `estado` (Enum: `pendiente`, `aprobado`, `rechazado`): Estado de la solicitud de autorización.
  - `descripcion` (Text): Resumen de los cambios propuestos.
  - `fecha_creacion` (DateTime): Fecha de envío de la solicitud.
  - `fecha_revision` (DateTime, Nullable): Fecha en que el líder la revisó.
- **Relaciones:**
  - **pertenece a** `*` `--` 1 **Proyecto**: Muchas solicitudes pueden pertenecer a un proyecto.
  - **es creada por** `*` `--` 1 **Usuario**: Muchas solicitudes pueden ser creadas por un usuario.
  - **agrupa** 1 `--` `*` **CambioPropuesto**: Una solicitud puede contener múltiples cambios específicos.

7. Entidad: **CambioPropuesto**

Detalla un cambio específico dentro de una SolicitudCambio (e.g., la modificación de un nodo).

- **Propiedades:**
  - `id` (UUID, Clave Primaria): Identificador único del cambio.
  - `id_solicitud` (UUID, Clave Foránea -> SolicitudCambio.id): Solicitud a la que pertenece este cambio.
  - `id_entidad_afectada` (UUID): ID del `NodoGrafo` o `PremisaGlobal` afectado.
  - `tipo_entidad` (Enum: `nodo`, `premisa_global`): Indica a qué tabla se refiere `id_entidad_afectada`.
  - `datos_anteriores` (JSON): Estado de la entidad antes del cambio, para auditoría y visualización de diferencias.
  - `datos_nuevos` (JSON): Estado propuesto para la entidad.
- **Relaciones:**
  - **es parte de** `*` `--` 1 **SolicitudCambio**: Muchos cambios específicos pertenecen a una única solicitud.

8. Entidad: **PremisaGlobal**

Almacena un axioma o definición base para un proyecto específico.

- **Propiedades:**
  - `id` (UUID, Clave Primaria): Identificador único de la premisa.
  - `id_proyecto` (UUID, Clave Foránea -> Proyecto.id): Proyecto al que pertenece.
  - `contenido_lean` (Text): La definición formal en Lean.
  - `descripcion_nl` (Text): Explicación en lenguaje natural.
- **Relaciones:**
  - **pertenece a** `*` `--` `1` **Proyecto**: Muchas premisas pertenecen a un único proyecto.

9. Entidad: **TrabajoAsincrono**

Realiza un seguimiento de las tareas de larga duración solicitadas por el usuario (IA, Clúster, validaciones).

- **Propiedades:**
  - `id` (UUID, Clave Primaria): Identificador único del trabajo (job).
  - `id_usuario` (UUID, Clave Foránea -> Usuario.id): Usuario que inició el trabajo.
  - `tipo_trabajo` (Enum: `validacion_externa`, `agente_demostracion`, `cluster_experimento`, etc.): Clasifica el tipo de tarea.
  - `id_recurso_asociado` (UUID, Nullable): ID de la entidad relacionada (e.g., `NodoGrafo.id` o `DemostracionExterna.id`).
  - `estado` (Enum: `encolado`, `en_progreso`, `completado`, `fallido`).
  - `resultado` (JSON, Nullable): Almacena el resultado final del trabajo.
  - `logs` (Text, Nullable): Registros de ejecución, especialmente en caso de error.
  - `fecha_creacion` (DateTime): Cuándo se inició el trabajo.
  - `fecha_finalizacion` (DateTime, Nullable): Cuándo terminó el trabajo.
- **Relaciones:**
  - **es iniciado por** `*` `--` `1` **Usuario**: Muchos trabajos pueden ser iniciados por un usuario.

10. Entidad: **DemostracionExterna**

Almacena metadatos sobre un archivo de demostración subido desde fuera del sistema.

- **Propiedades:**

    - `id` (UUID, Clave Primaria): Identificador único del archivo subido.
    - `id_usuario` (UUID, Clave Foránea -> Usuario.id): Usuario que subió el archivo.
    - `nombre_archivo_original` (String): Nombre del archivo en la máquina del usuario.
    - `path_almacenamiento` (String): Ruta al archivo en el sistema de almacenamiento (e.g., S3 bucket).
    - `formato` (Enum: `pdf`, `latex`, `imagen`, `lean`).
    - `estado_validacion` (Enum: `pendiente`, `procesando`, `valida`, `invalida`, `error`).
    - `fecha_carga` (DateTime).
- **Relaciones:**
    - **es subida por** `*` `--` 1 **Usuario**: Muchos archivos pueden ser subidos por el mismo usuario.

11. Entidad: **ConfiguracionUsuario**

Almacena las preferencias personales de un usuario para el entorno de trabajo.

- **Propiedades:**
    - `id_usuario` (UUID, Clave Primaria, Clave Foránea -> Usuario.id): Vincula la configuración al usuario.
    - `idioma` (String): Idioma preferido para la interfaz.
    - `tema_visual` (String): Tema de la UI (e.g., 'claro', 'oscuro').
    - `config_agentes_ia` (JSON): Parámetros para los modelos de IA (modelo a usar, reintentos, etc.).
    - `config_cluster` (JSON): Parámetros para la conexión y uso del clúster.
- **Relaciones:**
    - **pertenece a** 1 `--` 1 **Usuario**: Relación uno a uno estricta con la entidad Usuario.

Estructura Alternativa (NoSQL Orientada a Documentos)

En un modelo NoSQL, en lugar de "tablas", pensamos en **"colecciones"** de nivel superior. Cada elemento dentro de una colección es un **"documento"** (un objeto JSON) que puede contener datos complejos y anidados.

1. Colección: `usuarios`

Almacena la información de cada usuario y sus configuraciones. Se optimiza para obtener todo lo relacionado con un usuario en una sola lectura.

- **Documento (un documento por usuario, `id` es el `userId`):**

```
{
  "nombre": "String",
  "correo": "String", // Indexado para búsquedas
  "hash_contrasena": "String",
  "correo_verificado": "Boolean",
  "fecha_creacion": "Timestamp",
  "configuracion": { // Objeto anidado, 1-a-1
    "idioma": "String",
    "tema_visual": "String",
    "config_agentes_ia": "Map",
    "config_cluster": "Map"
  },
  "proyectos": { // Mapa para acceso rápido a los proyectos del usuario
    "proyectoId_1": "lider",
    "proyectoId_2": "colaborador"
  }
}
```

- **Justificación del Diseño:**
  - **Consulta Primaria:** "Obtener los datos y la configuración de un usuario" (`/users/me`). Al anidar la configuración, se obtiene todo con una sola lectura del documento del usuario.
  - **Consulta Primaria:** "¿A qué proyectos pertenece este usuario?". El mapa `proyectos` permite verificar permisos y construir el dashboard del usuario sin

necesidad de consultar otra colección. La clave es el ID del proyecto y el valor es el rol.

2. Colección: `proyectos`

Es la colección principal y el **agregado raíz** del modelo. Un solo documento de proyecto contendrá casi toda la información necesaria para renderizar el espacio de trabajo, promoviendo lecturas atómicas y rápidas.

- **Documento (un documento por proyecto, `id` es el `projectId`):**

```
{
  "nombre": "String",
  "descripcion": "Text",
  "visibilidad": "String ('publico' o 'privado')", // Indexado para la consulta de proyectos públicos
  "fecha_creacion": "Timestamp",
  "lider": { // Datos del líder desnormalizados
    "id_usuario": "String",
    "nombre": "String"
  },
  "colaboradores": { // Mapa de colaboradores para búsquedas rápidas
    "userId_1": { "nombre": "String", "fecha_invitacion": "Timestamp" },
    "userId_2": { "nombre": "String", "fecha_invitacion": "Timestamp" }
  },
  "premisas_globales": [ // Array de objetos anidados
    { "id": "UUID", "contenido_lean": "Text", "descripcion_nl": "Text" }
  ],
  "grafo": { // El grafo completo anidado como un mapa de nodos
```

```
    "nodoId_1": {
      "titulo": "String",
      "enunciado_nl": "Text",
      "codigo_lean": "Text",
      "estado": "String ('pendiente', 'demostrado', ...)",
      "tipo": "String ('teorema', 'evaluacion_numerica', ...)",
      "datos_numericos": "Map",
      "dependencias": ["nodoId_A", "nodoId_B"], // Nodos que ESTE necesita
      "dependientes": ["nodoId_X", "nodoId_Y"] // Nodos que necesitan a ESTE (para linaje)
    },
    "nodoId_2": { ... }
  }
}
```

- **Justificación del Diseño:**
  - **Consulta Primaria:** "Abrir un proyecto en el espacio de trabajo" (`/projects/{projectId}`). Al anidar el `grafo` y las `premisas`, una sola lectura de este documento es suficiente para cargar todo el entorno de trabajo. Esto es extremadamente eficiente.
  - **Consulta Primaria:** "Visualizar el linaje de un nodo" (`/proofs/{proofId}/lineage`). Gracias al array desnormalizado `dependientes` en cada nodo, encontrar qué otros resultados dependen de uno es una simple lectura de un campo, en lugar de una consulta costosa a toda la base de datos.
  - **Escrituras:** Las escrituras son más complejas. Por ejemplo, al agregar una dependencia de `nodo_1` a `nodo_A`, se debe actualizar el array `dependencias` en `nodo_1` y el array `dependientes` en `nodo_A`. Esto debe hacerse en una transacción para garantizar la consistencia. Este es el trade-off clásico de NoSQL: escrituras más complejas a cambio de lecturas ultrarrápidas.

3. Colección: `solicitudes_cambio`

Se mantiene como una colección separada porque tiene un ciclo de vida propio y los líderes necesitan consultarlas de forma aislada.

- **Documento (una solicitud por documento):**

```
{
  "id_proyecto": "String", // Indexado para buscar solicitudes por proyecto
  "nombre_proyecto": "String", // Desnormalizado para la UI
  "id_solicitante": "String",
  "nombre_solicitante": "String", // Desnormalizado para la UI
  "estado": "String ('pendiente', 'aprobado', 'rechazado')", // Indexado
  "fecha_creacion": "Timestamp",
  "cambios_propuestos": [ // Array de cambios detallados
    {
      "id_entidad_afectada": "String", // e.g., "nodoId_1"
      "tipo_entidad": "String ('nodo', 'premisa_global')",
      "datos_nuevos": "Map"
    }
  ]
}
```

- **Justificación del Diseño:**
  - **Consulta Primaria:** "Mostrar al líder todas las solicitudes pendientes de un proyecto". Se realiza una consulta simple en esta colección con `WHERE id_proyecto == '...' AND estado == 'pendiente'`. Mantener esto

fuera del documento principal del proyecto evita cargar datos innecesarios al abrir el espacio de trabajo.

4. Colección: `trabajos_asincronos`

Similar al modelo relacional, ya que estas son entidades con un ciclo de vida claro e independiente.

- **Documento (un trabajo por documento, `id` es el `jobId`):**

  ```
  {
    "id_usuario": "String",
    "tipo_trabajo": "String ('validacion_externa', 'agente_demostracion', ...)",
    "id_recurso_asociado": "String", // e.g., "nodoId_1" o "demostracionExternaId"
    "id_proyecto_contexto": "String", // Para saber a qué proyecto pertenece el recurso
    "estado": "String ('encolado', 'completado', 'fallido')",
    "resultado": "Map",
    "logs": "Text",
    "fecha_creacion": "Timestamp"
  }
  ```

- **Justificación del Diseño:** El frontend puede consultar el estado de un trabajo por su ID de forma periódica (`/jobs/{jobId}`) sin impactar el rendimiento de otras colecciones.

5. Colección: `indice_busqueda_demostraciones` (Para Búsqueda)

La mejor práctica es utilizar un servicio de búsqueda dedicado (como Algolia, Elasticsearch o Typesense). Esta colección sería el "índice" que se sincroniza con ese servicio.

- **Documento (un documento por cada nodo PÚBLICO):**

```
{
  // id del documento es el id del nodo original
  "id_nodo": "String",
  "id_proyecto": "String",
  "nombre_proyecto": "String",
  "titulo_nodo": "String",
  "enunciado_nl_nodo": "Text"
}
```

- **Justificación del Diseño:**
  - **Consulta Primaria:** "Buscar demostraciones por palabra clave" (`/search/proofs`). La búsqueda se realiza contra el servicio externo que usa este índice. El resultado devuelve los IDs necesarios (`id_nodo`, `id_proyecto`) para que el frontend pueda luego obtener el documento completo del proyecto y mostrar el nodo correcto. Esto desacopla la capacidad de búsqueda de la base de datos principal, permitiendo escalarla de forma independiente.