

## DOCUMENTACIÓN PROYECTO API DOCUMENTOS GOLANG

Esta documentación tiene como objetivo explicar de manera detallada la estructura, responsabilidades y flujo de trabajo de APIDOCUMENTOS que actúa como intermediaria entre los usuarios y un sistema de gestión documental (actualmente aplicado a un servicio externo gestionado con **Alfresco**). Está diseñada para ser comprensible tanto para desarrolladores que deseen integrarse con la API como para aquellos que busquen entender su funcionamiento interno.

### 1. ESTRUCTURA DEL PROYECTO

El proyecto está organizado siguiendo los principios de **Clean Architecture** y **Domain-Driven Design (DDD)**, con una clara separación de responsabilidades en capas. A continuación, se describe cada capa y su interacción:

#### 1.1. CAPA DE APLICACIÓN (application/)

##### Responsabilidad:

Esta capa implementa los casos de uso de la aplicación, orquestando la lógica de negocio mediante servicios que actúan como intermediarios entre la capa de infraestructura (API) y la capa de dominio.

##### Componentes:

- **application/documento/servicio\_documento.go**: Implementa la lógica de aplicación para operaciones como subir, listar y descargar documentos. Valida las reglas de negocio y delega las operaciones al dominio.
- **application/documento/servicio\_dto.go**: Define los Data Transfer Objects (DTOs) específicos para la comunicación con el servicio externo, realizando el mapeo entre JSON y entidades de dominio.

##### Interacción:

- Recibe solicitudes desde los handlers de la capa de infraestructura.
- Valida los datos de entrada y aplica reglas de negocio.
- Transforma los datos entre el formato de la API y las entidades de dominio.
- Delega las operaciones de almacenamiento al repositorio definido en la capa de dominio.

##### Dependencias:

- **domain/documento**: Utiliza las entidades y el repositorio del dominio.
- **shared/logger**: Para el registro de eventos y errores.

## 1.2. CAPA DE DOMINIO (domain/)

### Responsabilidad:

Es el núcleo de la aplicación, donde se definen las entidades de negocio y las reglas fundamentales, independientes de cualquier infraestructura o tecnología específica.

### Componentes:

- **domain/auth/servicio\_auth.go:** Define la lógica abstracta de autenticación, incluyendo generación y validación de tickets, gestión de roles y permisos.
- **domain/documento/entidad.go:** Define la estructura de datos principal (Documento) y sus reglas de validación (por ejemplo, validación de campos).
- **domain/documento/repositorio.go:** Define la interfaz AlmacenamientoDocumentos que abstrae las operaciones CRUD para documentos, permitiendo desacoplar la implementación concreta.

### Interacción:

- Las entidades encapsulan el estado y comportamiento del modelo de negocio.
- El repositorio define los contratos que deben implementar los servicios de almacenamiento.

## 1.3. CAPA DE INFRAESTRUCTURA (infrastructure/)

### Responsabilidad:

Implementa los detalles técnicos que permiten a la aplicación interactuar con el exterior: APIs REST, bases de datos, servicios externos, etc.

### 1.3.1. API (infrastructure/api/)

#### Responsabilidad:

Maneja las solicitudes HTTP entrantes, valida los datos de entrada y delega el procesamiento a la capa de aplicación.

#### Componentes:

- **infrastructure/api/handlers/auth\_handler.go:** Gestiona los endpoints de autenticación (login).
- **infrastructure/api/handlers/documento\_handler.go:** Maneja las operaciones básicas con documentos (subir, listar, descargar).
- **infrastructure/api/handlers/busqueda\_descarga\_handler.go:** Implementa funcionalidades específicas para buscar y descargar archivos del servicio externo.

- **infrastructure/api/handlers/lote\_documentos\_handler.go**: Gestiona la carga de múltiples documentos mediante arrays personalizados.

#### **Interacción:**

- Recibe solicitudes HTTP del cliente.
- Valida el formato y la estructura de los datos de entrada.
- Llama a los servicios de la capa de aplicación.
- Formatea y devuelve las respuestas al cliente.

#### **Dependencias:**

- **application/documento**: Utiliza los servicios de aplicación para procesar las solicitudes.
- **shared/middleware**: Para la autenticación y el registro de solicitudes.
- **shared/logger**: Para el registro de eventos y errores.

### **1.3.2. PERSISTENCIA (infrastructure/persistence/)**

#### **Responsabilidad:**

Implementa la interfaz de repositorio definida en el dominio, proporcionando acceso a sistemas de almacenamiento externos (actualmente aplicado a Alfresco y MongoDB).

#### **Componentes:**

- **infrastructure/persistence/servicio/cliente.go**: Implementa el cliente HTTP para comunicarse con servicio externo.
- **infrastructure/persistence/servicio/mock\_cliente.go**: Proporciona implementaciones simuladas para pruebas unitarias.
- **infrastructure/persistence/servicio/repositorio.go**: Implementa la interfaz AlmacenamientoDocumentos para servicio externo.
- **infrastructure/persistence/servicio/auth\_cliente.go**: Gestiona la autenticación específica con servicio externo.
- **infrastructure/persistence/db/bd\_mongo.go**: Implementa la persistencia en MongoDB.
- **infrastructure/persistence/db/mongo\_dto.go**: Define los DTOs para el almacenamiento en MongoDB.

#### **Interacción:**

- Recibe solicitudes de los servicios de aplicación.

- Realiza operaciones HTTP o de base de datos.
- Transforma los datos entre el formato de dominio y el formato requerido por los sistemas externos.

**Dependencias:**

- **domain/documento:** Implementa la interfaz de repositorio definida en el dominio.
- **shared/config:** Para obtener configuraciones como URLs y credenciales.
- **shared/logger:** Para el registro de eventos y errores.

### 1.3.3. ROUTER (infrastructure/router/)

**Responsabilidad:**

Configura las rutas HTTP de la API, aplica middlewares globales y enlaza los handlers.

**Componentes:**

- **infrastructure/router/API\_router.go:** Define las rutas HTTP, registra los middlewares y enlaza los handlers correspondientes.

**Interacción:**

- Es utilizado por el punto de entrada principal (main.go) para configurar el servidor HTTP.
- Registra los middlewares globales como autenticación y logging.
- Enlaza las rutas con los handlers correspondientes.

**Dependencias:**

- **infrastructure/api/handlers:** Utiliza los handlers para procesar las solicitudes.
- **shared/middleware:** Aplica los middlewares de autenticación y logging.

## 1.4. UTILIDADES COMPARTIDAS (shared/)

**Responsabilidad:**

Proporciona utilidades transversales utilizadas por múltiples capas de la aplicación.

### 1.4.1. MIDDLEWARE (shared/middleware/)

**Responsabilidad:**

Implementa funciones que interceptan y procesan las solicitudes HTTP antes de que lleguen a los handlers.

**Componentes:**

- **shared/middleware/logging.go:** Registra información sobre las solicitudes HTTP, como duración, estado y dirección IP del cliente.

**Interacción:**

- Es aplicado por el router a las rutas de la API.
- Procesa las solicitudes HTTP antes y después de que sean manejadas por los handlers.

**Dependencias:**

- **shared/logger:** Para registrar información sobre las solicitudes.
- **shared/config:** Para obtener configuraciones relacionadas con la autenticación.

#### 1.4.2. CONFIGURACIÓN (shared/config/)

**Responsabilidad:**

Centraliza la carga y gestión de la configuración de la aplicación desde variables de entorno o archivos de configuración.

**Componentes:**

- **shared/config/config.go:** Carga y proporciona acceso a la configuración de la aplicación.

**Interacción:**

- Es utilizado por diversos componentes para obtener configuraciones como URLs, credenciales, límites de tamaño, etc.

#### 1.4.3. LOGGER (shared/logger/)

**Responsabilidad:**

Proporciona un sistema de logging estructurado para registrar eventos y errores en la aplicación.

**Componentes:**

- **shared/logger/logger.go:** Implementa funcionalidades de logging con diferentes niveles de severidad y formatos.

**Interacción:**

- Es utilizado por prácticamente todos los componentes de la aplicación para registrar eventos y errores.

**Dependencias:**

- **shared/config:** Para obtener configuraciones relacionadas con el logging.

## 1.5. ARCHIVOS DE CONFIGURACIÓN Y ENTRADA

### Componentes:

- **.env.example:** Plantilla que muestra las variables de entorno necesarias para configurar la aplicación.
- **go.mod y go.sum:** Definen las dependencias del proyecto y sus versiones.
- **main.go:** Punto de entrada de la aplicación, inicializa los componentes principales y arranca el servidor HTTP.

## 2. FLUJO DE EVENTOS

A continuación, se describe el flujo de eventos típico en la API, desde que llega una solicitud HTTP hasta que se devuelve una respuesta:

### 2.1. Solicitud HTTP

- El cliente realiza una solicitud HTTP (por ejemplo, subir un documento) a uno de los endpoints definidos en el router.

### 2.2. Middleware

- El middleware de logging registra información sobre la solicitud entrante (método, ruta, IP del cliente, etc.).
- El middleware de autenticación valida las credenciales proporcionadas (token, ticket de Servicio, API Key) y las añade al contexto de la solicitud.

### 2.3. Handler

- El handler correspondiente (por ejemplo, en documento\_handler.go) recibe la solicitud.
- Valida los datos de entrada (por ejemplo, verifica la presencia del archivo y los metadatos).
- Transforma los datos de la solicitud en DTOs de aplicación.
- Llama al servicio correspondiente en la capa de aplicación.

### 2.4. Servicio de Aplicación

- El servicio (por ejemplo, en servicio\_documento.go) valida las reglas de negocio.
- Transforma los DTOs en entidades de dominio.
- Llama al repositorio de dominio para realizar operaciones con las entidades.

## 2.5. Repositorio (Implementación)

- La implementación del repositorio (por ejemplo, en `infrastructure/persistence/servicio/repositorio.go`) transforma las entidades de dominio en DTOs específicos para las persistencias (actualmente aplicado a Alfresco y MongoDB).
- Utiliza el cliente HTTP para realizar solicitudes a Alfresco o interactúa con la base de datos MongoDB.
- Procesa las respuestas y las transforma en entidades de dominio.

## 2.6. Respuesta

- El repositorio devuelve las entidades de dominio al servicio de aplicación.
- El servicio de aplicación devuelve los DTOs al handler.
- El handler construye la respuesta HTTP (JSON, archivo para descarga, etc.) y la devuelve al cliente.

## 2.7. Logging

- El middleware de logging registra información sobre la respuesta (duración, estado, etc.).

# 3. CASOS DE USO PRINCIPALES

## 3.1. Autenticación

La API proporciona endpoints para autenticarse con el servicio de gestión documental Alfresco, que luego devuelve un ticket .

### Flujo:

1. El cliente envía sus credenciales al endpoint de autenticación.
2. El `auth_handler.go` valida las credenciales y llama al servicio de autenticación.
3. El servicio de autenticación utiliza el `auth_cliente.go` para obtener un ticket de Alfresco.
4. El ticket se devuelve al cliente.

## 3.2. Subir un Documento

La API permite subir documentos a Alfresco junto con sus metadatos.

### Flujo:

1. El cliente envía un archivo y sus metadatos al endpoint de subida.

2. El documento\_handler.go valida la solicitud y extrae el archivo y los metadatos.
3. Llama al servicio de aplicación servicio\_documento.go.
4. El servicio valida los metadatos y llama al repositorio implementado en repositorio.go.
5. El repositorio utiliza el cliente HTTP para subir el documento a Alfresco.
6. La respuesta de Alfresco se procesa y se devuelve al cliente.

### **3.3. Listar Documentos**

La API permite buscar documentos en Alfresco según criterios específicos.

**Flujo:**

1. El cliente envía criterios de búsqueda al endpoint de listado.
2. El documento\_handler.go valida la solicitud y extrae los criterios.
3. Llama al servicio de aplicación servicio\_documento.go.
4. El servicio valida los criterios y llama al repositorio.
5. El repositorio utiliza el cliente HTTP para buscar documentos en servicio.
6. La lista de documentos se procesa y se devuelve al cliente.

### **3.4. Descargar un Documento**

La API permite descargar documentos almacenados en servicio externo.

**Flujo:**

1. El cliente solicita la descarga de un documento específico.
2. El busqueda\_descarga\_handler.go valida la solicitud y extrae el ID del documento.
3. Llama al servicio de aplicación.
4. El servicio llama al repositorio.
5. El repositorio utiliza el cliente HTTP para descargar el documento de Alfresco.
6. El documento se devuelve al cliente como un archivo descargable.

### **3.5. Cargar Lote de Documentos**

La API permite cargar múltiples documentos en una sola operación.



## Flujo:

1. El cliente envía un array de documentos y metadatos al endpoint de carga por lotes.
2. El lote\_documentos\_handler.go valida la solicitud y extrae los documentos y metadatos.
3. Itera sobre el array y procesa cada documento individualmente.
4. Para cada documento, sigue un flujo similar al de subir un solo documento.
5. Recopila los resultados y los devuelve al cliente.

## 4. EJEMPLOS DE USO

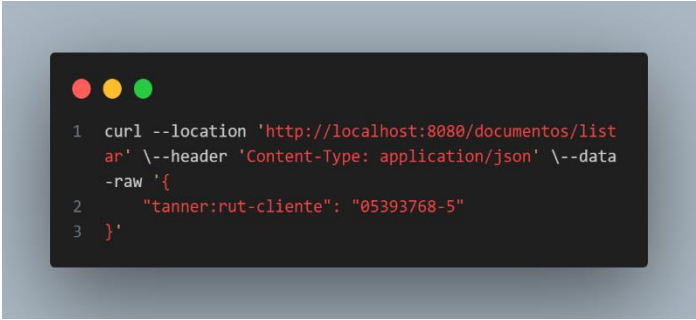
### 4.1. Autenticación

```
1 curl --location 'http://localhost:8080/auth/login' \
  --header 'ADFTannerServices: {{API_KEY}}' --header \
  'Content-Type: application/json' --data-raw '{
2   "userId": "admin",
3   "password": "admin"
4 }'
```

### 4.2. Subir un Documento

```
1 curl --location 'http://localhost:8080/documentos/subir' --form \
  'documento=@/ruta/archivo.pdf' --form 'propiedades=
2   {
3     "tanner:tipo-documento": "pruebaotramas",
4     "cm:title": "Articulo 85",
5     "cm:versionType": "MAJOR",
6     "cm:versionLabel": "2.0",
7     "tanner:razon-social-cliente": "Samuel",
8     "tanner:estado-visado": "Sin información",
9     "tanner:estado-vigencia": "Vigente",
10    "tanner:fecha-carga": "2024-11-19T18:42:03.730Z",
11    "tanner:fecha-modificacion": "2024-11-19T18:42:03.730Z",
12    "tanner:observaciones": "Demos",
13    "tanner:rut-cliente": "05393768-5",
14    "tanner:nombre-doc": "Articulo 85",
15    "cm:lastThumbnailModification": [
16      "doclib:1733284022799"
17    ],
18    "cm:description": "Articulo 85",
19    "tanner:categorias": "Empresas",
20    "tanner:sub-categorias": "Comercial",
21    "tanner:origen": "sistema-adf",
22    "tanner:relacion": "Cliente",
23    "tanner:fecha-termino-vigencia": "2024-11-22T05:00:00.000
24  }"
```

### 4.3. Listar Documentos



```
1 curl --location 'http://localhost:8080/documentos/listar' \
  --header 'Content-Type: application/json' \
  --data -raw '{
2   "tanner:rut-cliente": "05393768-5"
3 }'
```

### 4.4. Descargar un Documento



```
1 curl --location --request POST 'http://localhost:8080/documentos/descargar?idFile=e72c4f24-ae2f-4ad9-b654-5fee2654e315'
```

## 4.5. Cargar lote de Documentos

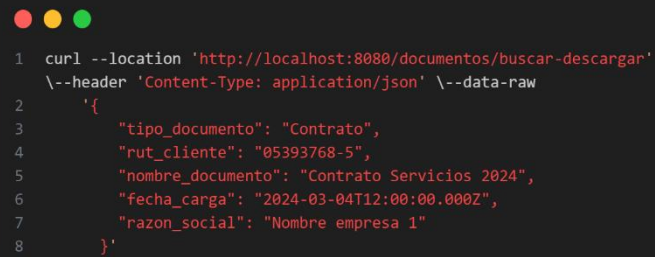
```
1 curl --location 'http://localhost:8080/documentos/subir-lote' \--f
orm 'documento=@"/ruta/archivo.pdf"' \--form 'propiedades=
2 [{"
3     "tanner:tipo-documento": "Prueba LoteBD 1",
4     "cm:title": "prueba loteBD1",
5     "cm:description": "Descripción del documento 1",
6     "tanner:nombre-doc": "prueba loteBD 1",
7     "cm:versionType": "MAJOR",
8     "cm:versionLabel": "2.0",
9     "tanner:rut-cliente": "14171166-0",
10    "tanner:razon-social-cliente": "Nombre Empresa 1",
11    "tanner:categorias": "Empresas",
12    "tanner:sub-categorias": "Comercial",
13    "tanner:estado-vigencia": "Vigente",
14    "tanner:fecha-termino-vigencia": "2025-12-31T05:00:00.000
Z",
15    "tanner:fecha-carga": "2025-03-04T12:00:00.000Z",
16    "tanner:fecha-modificacion": "2024-03-04T12:00:00.000Z",
17    "tanner:observaciones": "Demos",
18    "tanner:estado-visado": "Sin información",
19    "tanner:origen": "sistema-adf",
20    "tanner:relacion": "Ambos"
21  },
22  {
23    "tanner:tipo-documento": "Prueba LoteBD 2",
24    "cm:title": "prueba loteBD1",
25    "cm:description": "Descripción del documento 2",
26    "tanner:nombre-doc": "prueba loteBD 2",
27    "cm:versionType": "MAJOR",
28    "cm:versionLabel": "2.0",
29    "tanner:rut-cliente": "14171166-0",
30    "tanner:razon-social-cliente": "Nombre Empresa 2",
31    "tanner:categorias": "Empresas",
32    "tanner:sub-categorias": "Comercial",
33    "tanner:estado-vigencia": "Vigente",
34    "tanner:fecha-termino-vigencia": "2025-12-31T05:00:00.000
Z",
35    "tanner:fecha-carga": "2025-03-04T12:00:00.000Z",
36    "tanner:fecha-modificacion": "2024-03-04T12:00:00.000Z",
37    "tanner:observaciones": "Demos",
38    "tanner:estado-visado": "Sin información",
39    "tanner:origen": "sistema-adf",
40    "tanner:relacion": "Ambos"
41  }]"
```

## 4.6. Buscar Documento Por Metadata/Propiedades

### Opción 1 (Búsqueda propiedades Tanner)

```
1 curl --location 'http://localhost:8080/documentos/buscar-descargar'
\--header 'Content-Type: application/json' \--data-raw '{
2   {
3     "tanner:nombre-doc": "Contrato Servicios 2024",
4     "tanner:tipo-documento": "Contrato",
5     "tanner:razon-social-cliente": "Nombre empresa 1",
6     "tanner:rut-cliente": "05393768-5",
7     "tanner:estado-vigencia": "Vigente",
8     "tanner:fecha-carga": "2024-03-04T12:00:00.000Z"
9   }'}
```

## Opción 2 (Búsqueda propiedades Mongo DB)



```
1 curl --location 'http://localhost:8080/documentos/buscar-descargar'
  --header 'Content-Type: application/json' --data-raw
2  '{
3    "tipo_documento": "Contrato",
4    "rut_cliente": "05393768-5",
5    "nombre_documento": "Contrato Servicios 2024",
6    "fecha_carga": "2024-03-04T12:00:00.000Z",
7    "razon_social": "Nombre empresa 1"
8  }'
```

## 5. INTEGRACIÓN CON MONGO DB

La API también integra MongoDB para almacenar información complementaria o realizar operaciones que no requieren acceso al servicio externo.

### Componentes:

- `infrastructure/persistence/db/bd_mongo.go`: Implementa las operaciones CRUD en MongoDB.

- **infrastructure/persistence/db/mongo\_dto.go**: Define los DTOs para el almacenamiento en MongoDB.

**Uso:**

- Registro de operaciones y auditoría.
- Almacenamiento de metadatos adicionales no soportados por servicio.
- Caché de datos frecuentemente accedidos para mejorar el rendimiento.

## 6. GESTIÓN DE ERRORES

La API implementa un manejo uniforme de errores a través de todas las capas:

1. **Errores de dominio**: Definidos en la capa de dominio, representan violaciones de reglas de negocio.
2. **Errores de aplicación**: Ocurren en la capa de aplicación, durante la validación o procesamiento.
3. **Errores de infraestructura**: Surgen en la comunicación con sistemas externos o problemas técnicos.
4. **Errores HTTP**: Devueltos al cliente con códigos de estado apropiados y mensajes descriptivos.

El logger registra los errores con niveles de severidad adecuados y contexto relevante para facilitar el diagnóstico.

## 7. SEGURIDAD

La API implementa varias capas de seguridad:

1. **Autenticación**: Valida la identidad del usuario mediante tickets de Alfresco.
2. **Autorización**: Verifica que el usuario tenga permisos para realizar operaciones específicas.
3. **API Key**: Requiere una clave de API válida para acceder a los endpoints.
4. **Validación de entrada**: Previene inyecciones y otros ataques mediante validación estricta.
5. **Límites de tamaño**: Restringe el tamaño máximo de archivos y cargas para prevenir ataques de denegación de servicio.

## 8. BUENAS PRACTICAS DE DESARROLLO

El proyecto sigue varias buenas prácticas de desarrollo en Go:

1. **Manejo de errores explícito:** Utiliza el patrón de retorno de error de Go para manejar fallos de manera explícita.
2. **Interfaces pequeñas:** Define interfaces cohesivas con pocos métodos para facilitar la implementación y las pruebas.
3. **Principio de responsabilidad única:** Cada componente tiene una única razón para cambiar.
4. **Inyección de dependencias:** Las dependencias se pasan explícitamente a los componentes que las necesitan.
5. **Programación idiomática:** Sigue las convenciones y patrones estándar de Go.
6. **Pruebas unitarias:** Utiliza mocks y stubs para probar componentes de manera aislada.

## G. CONSIDERACIONES FINALES

- **Escalabilidad:** La arquitectura en capas y la separación de responsabilidades facilitan la escalabilidad horizontal.
- **Mantenibilidad:** El uso de interfaces y la inversión de dependencias facilitan los cambios y la evolución del sistema.
- **Extensibilidad:** Nuevas funcionalidades pueden integrarse fácilmente implementando nuevos handlers, servicios o repositorios.
- **Robustez:** La gestión adecuada de errores y la validación de entrada garantizan un sistema resistente a fallos.
- **Compatibilidad:** La API está diseñada para integrarse con otros sistemas y clientes mediante interfaces RESTful estándar.