# Manual de integración (API)

Explica cómo acceder y consumir el API del Catálogo, incluyendo autenticación, consulta de datos, estructura de los endpoints, y documentación interactiva para pruebas en vivo con credenciales válidas.

# Quick Start (Inicio rápido)

# ¿Qué es el API del Catálogo?

El **API del Catálogo de Servicios del Estado Dominicano** es una interfaz que permite a instituciones y desarrolladores **consultar, integrar y utilizar la información de los servicios públicos** registrados en el Catálogo.

Su objetivo es facilitar la interoperabilidad entre plataformas del Estado y sistemas externos, permitiendo que los datos del catálogo estén disponibles de forma **segura, estructurada y actualizada**.

---

# ¿Para qué sirve?

- **Consultar servicios públicos en tiempo real**
  Accede a información de los servicios que ofrecen las instituciones del Estado Dominicano.
- **Integrar con portales o aplicaciones**
  Conecta datos del catálogo con portales ciudadanos, sistemas internos o aplicaciones móviles.
- **Optimizar procesos institucionales**
  Permite a las instituciones reutilizar datos y automatizar tareas relacionadas con servicios y trámites.

---

# ¿Qué ofrece el API?

- **Acceso a datos de servicios y trámites** (información general, estados, instituciones responsables).
- **Parámetros de búsqueda y filtrado avanzados** (por institución, tipo de servicio, estado, entre otros).
- **Formato estándar (JSON)** para fácil integración.
- **Documentación y herramientas interactivas** para desarrolladores.
- **Autenticación y control de acceso** para garantizar el uso seguro de la información.

---

# ¿Quiénes lo pueden usar?

- **Instituciones del Estado** que necesiten integrar información del catálogo en sus portales o sistemas internos.
- **Desarrolladores** que trabajen en soluciones digitales para el sector público.
- **Sistemas ciudadanos** o portales que consulten datos oficiales del catálogo.

---

# ¿Qué necesitas para empezar?

1. **Credenciales de acceso** (API Keys u otro método, según el caso).
2. **Revisar la Guía de Inicio Rápido (Quick Start)** para probar tu primera consulta.
3. **Acceder a Rapidocus** para explorar y probar los endpoints de forma interactiva.
4. **Consultar la Referencia Técnica (OpenAPI)** para ver todos los endpoints disponibles.

---

# Notas importantes

> - El API es **RESTful** y devuelve datos en **formato JSON**.
> - Algunos endpoints son **públicos** y otros requieren autenticación.
> - El uso indebido o abusivo (consultas masivas sin autorización) puede generar restricciones según los **SLA (Acuerdos de Nivel de Servicio)**.

# Cómo obtener acceso (credenciales, entorno)

Para poder utilizar el **API del Catálogo de Servicios del Estado Dominicano**, es necesario **solicitar credenciales y configurar tu entorno de acceso**.

De forma predeterminada, todos los datos del catálogo están protegidos y solo algunos endpoints básicos son públicos.

---

## 1. Solicitar un usuario y credenciales

Antes de poder autenticarte en el API, debes solicitar un usuario institucional:

1. **Enviar una solicitud a la División de Arquitectura de OGTIC**
   - Especifica el proyecto o sistema que consumirá el API.
   - Indica el responsable técnico de la integración.
   - Proporciona un correo de contacto institucional.
2. **Recibirás credenciales de acceso** que incluyen:
   - **Usuario y contraseña inicial**.
   - **Instrucciones para generar tu primer token de acceso**.
   - **Enlace al entorno de pruebas** y, posteriormente, al **entorno de producción**.

---

## 2. Configurar tu entorno de desarrollo

Una vez recibidas tus credenciales:

1. **Accede al entorno de pruebas (sandbox)** para validar tus integraciones.
   La dirección base (base URL) será indicada en el correo de la OGTIC.

   Ejemplo:
   `https://catalogo-staging-1062351960128.us-east1.run.app/`

2. **Genera un token de acceso** (siguiendo la guía de Autenticación).
   Este token será necesario para consumir cualquier endpoint protegido.
3. **Usa herramientas como Postman o cURL** para probar tus primeras solicitudes antes de integrar en tu sistema.

---

# 3. Pasar a producción

Cuando tus pruebas en sandbox estén aprobadas:

1. Solicita a OGTIC la habilitación de tu usuario para **producción**.
2. Recibirás:
   - **Nueva URL base** para producción.
   - **Límites y parámetros de consumo (según los SLA vigentes)**.
   - **Credenciales actualizadas si es necesario**.

# Notas importantes

> - **El entorno de pruebas y producción son separados**; asegúrate de usar las URLs correctas en cada fase.
> - **Los tokens expiran**; debes renovarlos periódicamente (ver Actualizar token).
> - **El uso del API está regulado por los SLA**: evita consultas masivas sin autorización.
> - La información sobre autenticación avanzada (renovación de tokens, roles y permisos) está en el capítulo de **Autenticación**.

# Tu primera consulta (ejemplos prácticos)

En este paso realizarás tu **primera consulta exitosa** al API del Catálogo para obtener la lista de servicios. Aquí tienes ejemplos.

## □□ Requisitos previos

1. Ya tienes un **token de acceso válido** (consulta Obtener acceso)
2. Estás usando correctamente el **entorno sandbox** o **producción**, según corresponda

# REST API y los Datos relacionales

De forma predeterminada, Catalogo solo recupera el valor de referencia de un campo relacional en los elementos. Para recuperar también datos anidados de un campo relacional, se puede utilizar el parámetro `fields` en REST. Esto le permite recuperar datos referenciados de un servicio.

Para definir el ámbito de los campos que se devuelven por tipo de colección, puede usar la sintaxis `<field>:<scope>` del parámetro fields de la siguiente manera:

```
GET /items/services
□?fields[]=name
□&fields[]=description
□&fields[]=legal_framework_ids.description
□&fields[]=services_electronic_gov_type_ids.electronic_gov_type_id.type
```

Otra alternativa seria:

```
GET /items/services
□?fields[]=name,description,legal_framework_ids.description,services_electronic_gov_type_ids.electronic_gov_type
```

## Método HTTP SEARCH

Al usar la API de REST para leer varios elementos mediante filtros (muy) avanzados, es posible que se encuentre con el problema de que la dirección URL simplemente no puede contener suficientes datos para incluir la estructura de consulta completa. En esos casos, puede usar el método HTTP SEARCH como reemplazo directo de GET, donde se le permite colocar la consulta en el cuerpo de la solicitud de la siguiente manera:

**Antes**:

```
GET /items/services?filter[name][_eq]=Hello World
```

**Después**:

```
SEARCH /items/services

{
  "query": {
    "filter": {
      "name": {
        "_eq": "Hello World"
      }
    }
  }
}
```

Hay mucha discusión sobre si se debe o no poner un cuerpo en una solicitud GET, usar POST para crear consultas de búsqueda o confiar en un método completamente diferente. A partir de ahora, hemos optado por alinearnos con la especificación del método HTTP SEARCH de IETF.

**Lectura útil**:

- Método de búsqueda HTTP (IETF, 2021)
- Definición de un nuevo método HTTP: HTTP SEARCH (Tim Perry, 2021)
- HTTP GET con cuerpo de solicitud (StackOverflow, 2009 y en curso)
- Uso del cuerpo de Elastic Search GET (elástico, s.f.)
- Dropbox comienza a usar POST y por qué se trata de un diseño deficiente de la API. (Evert Pot, 2015)

# Códigos de error

A continuación, se muestran los códigos de error globales utilizados en Catalogo y lo que significan.

| Código de error | Estado | Descripción |
|---|---|---|
| FAILED VALIDATION | 400 | Error en la validación de este elemento en particular |
| FORBIDDEN | 403 | No se le permite realizar la acción actual |
| INVALID_TOKEN | 403 | El token proporcionado no es válido |
| TOKEN_EXPIRED | 401 | El token proporcionado es válido, pero ha caducado |

| Código de error | Estado | Descripción |
| --- | --- | --- |
| INVALID_CREDENTIALS | 401 | El nombre de usuario/contraseña o el token de acceso son incorrectos |
| INVALID IP | 401 | Su dirección IP no está en la lista de permitidos para ser utilizada con este usuario |
| INVALID_OTP | 401 | Se proporcionó una OTP incorrecta |
| INVALID_PAYLOAD | 400 | La carga útil proporcionada no es válida |
| INVALID_QUERY | 400 | No se pueden utilizar los parámetros de consulta solicitados |
| UNSUPPORTED_MEDIA_TYPE | 415 | El formato de carga útil o el encabezado proporcionados no son compatibles `Content-Type` |
| REQUESTS_EXCEEDED | 429 | Alcanza el límite de velocidad |
| ROUTE_NOT_FOUND | 404 | El punto de conexión no existe |
| SERVICE_UNAVAILABLE | 503 | No se pudo usar el servicio externo |
| UNPROCESSABLE_CONTENT | 422 | Intentaste hacer algo ilegal |

> **Seguridad**
>
> Para evitar que se filtren los elementos existentes, todas las acciones de los elementos no existentes devolverán un error FORBIDDEN.

Para conocer más sobre los items, consultas y filtrado en catalogo visita los enlaces correspondientes.

# Guía paso a paso (Step by Step)

# Autenticación

Todos los datos dentro de la plataforma son privados de forma predeterminada. El rol público se puede configurar para exponer datos sin autenticación, o puede pasar un token de acceso a la API para acceder a datos privados.

# Tokens de acceso

Hay tres tipos de tokens que se pueden usar para autenticarse dentro de Catalogo.

1. **El token temporal (JWT)** es devuelto por el punto de conexión/mutación de inicio de sesión. Estos tokens tienen un tiempo de caducidad relativamente corto y, por lo tanto, son la opción más segura de usar. Los tokens se devuelven con un token de actualización que se puede usar para recuperar un nuevo token de acceso a través del punto de conexión o mutación de actualización.
2. **El token de sesión (JWT)** también puede ser devuelto por el punto de conexión/mutación de inicio de sesión. Los tokens de sesión combinan un token de actualización y un token de acceso en una sola cookie. Estos tokens no deben tener un tiempo de caducidad corto como los tokens temporales, ya que no se pueden actualizar después de que hayan caducado.
3. **El token estático** se puede configurar para cada usuario de la plataforma y nunca caducan. Son menos seguros, pero bastante útiles para la comunicación de servidor a servidor.

Una vez que tenga su token de acceso, hay tres formas de pasarlo a la API: en el encabezado o header `Authorization` de la solicitud, como **cookie de sesión** o a través del parámetro `access_token` de consulta.

## Encabezado de autorización

```
Authorization: Bearer <token>
```

## Cookie de sesión

```
Cookie: directus_session_token=<token>
```

## Parámetro de consulta

```
?access_token=<token>
```

> ADVERTENCIA
>
> La opción de parámetro de consulta no se recomienda en las configuraciones de producción, ya que varios sistemas pueden registrar los parámetros.

# Iniciar sesión

Autenticarse como usuario.

### Request

```
POST /auth/login
{
"email": user_email,
"password": user_password
}
```

### Cuerpo de la solicitud

- `email`: Dirección de correo electrónico requerida del usuario.
- `password`: Contraseña requerida del usuario.
- `otp`: La contraseña de un solo uso del usuario (si MFA está habilitada).
- `mode`: Si se va a recuperar el token de actualización en la respuesta JSON o en una cookie `httpOnly`. Uno de `json`, `cookie` o `session`. El valor predeterminado es `json`.

### Respuesta

- `access_token` **string**: Token de acceso temporal que se usará en las solicitudes de seguimiento. Nota: si lo usaste como modo en la solicitud, el token de acceso no se devolverá en el JSON.session
- `expires` **integer**: Cuánto tiempo pasará antes de que caduque el token de acceso. El valor se expresa en milisegundos.
- `refresh_token` **string**: El token que se puede usar para recuperar un nuevo token de acceso a través de /auth/refresh. Nota: si usaste `cookie` o `session` como el modo en la solicitud, el token de actualización no se devolverá en el JSON.

> Tiempo de caducidad
>
> El tiempo de caducidad del token se puede configurar a través de la variable de entorno ACCESS_TOKEN_TTL.

**Ejemplo**

```
POST /auth/login
{
"email": "admin@example.com",
"password": "c4t4l0g0"
}
```

# Actualizar token

Recupere un nuevo token de acceso mediante un token de actualización.

**Request**

```
POST /auth/refresh
{
"refresh_token": refresh_token_string,
"mode": refresh_mode
}
```

**Cuerpo de la solicitud**

- `refresh_token`: El token de actualización que se va a usar. Si tiene el token de actualización en una cookie a través de `/auth/login`, no es necesario que lo envíe aquí.
- `mode`: Si se debe enviar y recuperar el token de actualización en la respuesta JSON o en una cookie `httpOnly`. Uno de `json`, `cookie` o `session`.

**Respuesta**

- `access_token` **string**: Token de acceso temporal que se usará en las solicitudes de seguimiento. Nota: si lo usaste como modo en la solicitud, el token de acceso no se devolverá en el JSON.session
- `expires` **integer**: Cuánto tiempo pasará antes de que caduque el token de acceso. El valor se expresa en milisegundos.
- `refresh_token` **string**: El token que se puede usar para recuperar un nuevo token de acceso a través de `/auth/refresh`. Nota: si usaste `cookie` o `session` como el modo en la solicitud, el

token de actualización no se devolverá en el JSON.

**Ejemplo**

```
POST /auth/refresh
{
"refresh_token": "gmPd...8wuB",
"mode": "json"
}
```

# Cerrar sesión

Invalide el token de actualización, destruyendo así la sesión del usuario.

**Request**

```
POST /auth/logout
{
"refresh_token": refresh_token
}
```

**Cuerpo de la solicitud**

- `refresh_token`: El token de actualización que se va a invalidar. Si tiene el token de actualización en una cookie a través de /auth/login, no es necesario que lo envíe aquí.
- `mode`: Si el token de actualización se envía en la respuesta JSON o en una cookie `httpOnly`. Uno de `json`, `cookie` o `session`.

**Ejemplo**

```
POST /auth/logout
{
"refresh_token": "gmPd...8wuB",
"mode": "json"
}
```

# Solicitar restablecimiento de contraseña

Solicite que se envíe un correo electrónico de restablecimiento de contraseña al usuario determinado.

**Request**

```
POST /auth/password/request
{
"email": user_email
}
```

**Cuerpo de la solicitud**

- `email` **Requerido**: Dirección de correo electrónico del usuario para el que solicitas el restablecimiento de contraseña.
- `reset_url`: Proporcione una URL de restablecimiento personalizada a la que le llevará el enlace del correo electrónico. El token de restablecimiento se pasará como parámetro.

**Ejemplo**

```
POST /auth/password/request
{
"email": "admin@example.com"
}
```

# Restablecer una contraseña

La solicitud de un punto de conexión de restablecimiento de contraseña envía un correo electrónico con un vínculo a la aplicación de administración (o a una ruta personalizada) que, a su vez, usa este punto de conexión para permitir que el usuario restablezca su contraseña.

**Request**

```
POST /auth/password/reset
{
"token": password_reset_token,
"password": password
}
```

**Cuerpo de la solicitud**

- `token` **Requerido**: Token de restablecimiento de contraseña, tal y como se proporciona en el correo electrónico enviado por el punto de conexión de la solicitud.

- `password` **Requerido**: Nueva contraseña para el usuario.

### Ejemplo

```
POST /auth/password/reset
{
"token": "eyJh...KmUk",
"password": "c4t4l0g0"
}
```

Guía paso a paso (Step by Step)

# Parámetros de consulta global

La mayoría de las operaciones de punto de conexión de la API de Catalogo se pueden manipular con los siguientes parámetros. Es importante entenderlos para sacar el máximo partido a la plataforma.

## Campos

Elija los campos que se devuelven en el conjunto de datos actual. Este parámetro admite la notación de puntos para solicitar campos relacionales anidados. También puede utilizar un comodín (*) para incluir todos los campos a una profundidad específica.

```
GET /items/services
  ?fields=name,description,status_id.name
```

### Ejemplos

| Campo | Descripción |
|---|---|
| `name,description` | Devuelve solo los campos `name` y `description`. |
| `name,status_id.name` | Devuelve `name` y el elemento `name` relacionado con `status_id` |
| `*` | Devuelve todos los campos. |
| `*.*` | Devuelve todos los campos de primer y segundo nivel relacionado. |
| `*,institution_id.*` | Devuelve todos los campos y todos los campos de `institution_id`. |

> **Rendimiento y tamaño**
>
> Aunque el comodín `fields` es muy útil para fines de depuración, se recomienda solicitar solo campos específicos para su uso en producción. Al solicitar solo los campos que realmente necesita, puede acelerar la solicitud y reducir el tamaño total de la salida.

# Relaciones

En el **Catálogo de Servicios** algunos campos de la colección `services` almacenan datos relacionados con múltiples colecciones mediante campos relacionales o tablas intermedias.

Para solicitar **campos específicos** de cada relación, utiliza la sintaxis:

```
?fields=<relacion>.<subrelacion>.<campo>
```

Esto permite que, al consultar servicios, podamos incluir campos de colecciones relacionadas sin traer datos innecesarios.

---

**Escenario**

Sabiendo que tenemos la colección `services`, que tiene relaciones con otras colecciones:

- **status_id** → colección `status`
- **service_type_id** → colección `service_type`
- **addressed_to_id** → colección `addressed_to`

Queremos traer:

- Campos básicos del servicio
- Estado y tipo
- Público objetivo

---

Quedaría de esta manera:

```
GET /items/services
?fields[]=name
  &fields[]=status_id.name
  &fields[]=service_type_id.type
  &fields[]=addressed_to_id.other_entity
```

# Filtro

Se utiliza para buscar elementos en una colección que coincida con las condiciones del filtro. El parámetro filter sigue la especificación Reglas de filtro, que incluye información adicional sobre operadores lógicos (AND/OR), filtrado relacional anidado y variables dinámicas.

Hay dos sintaxis disponibles:

```
GET /items/services
    ?filter[name][_eq]=Renovación pasaporte

// or

GET /items/services
    ?filter={ "name": { "_eq": "Renovación pasaporte" }}
```

**Ejemplos**

Recupera todos los elementos donde es igual a "Renovación pasaporte" **name**

```
{
  "name": {
    "_eq": "Renovación pasaporte"
  }
}
```

Recupere todos los elementos de uno de los siguientes estados: "Borrador", "Publicado"

```
{
  "status_id": {
    "_in": [1, 2]
  }
}
```

Recuperar todos los elementos que se publican entre dos fechas

```
{
  "date_created": {
    "_between": ["2021-01-24", "2021-02-23"]
  }
}
```

Recupere todos los elementos en los que la marca "**default_variation**" de una variación es verdadera

```
{
  "variation_ids": {
```

```
  "default_variation": {
    "_eq": true
  }
}
}
```

> **Filtros anidados**
>
> En el ejemplo anterior se filtrarán los elementos de nivel superior en función de una condición del elemento relacionado. Si quieres filtrar los elementos relacionados, ¡echa un vistazo al parámetro deep!

# Buscar

El parámetro de búsqueda permite realizar una búsqueda en todos los campos de tipo cadena y texto de una colección. Es una manera fácil de buscar un artículo sin crear filtros de campo complejos, aunque está mucho menos optimizado. Solo busca en los campos del elemento raíz, no se incluyen los campos de elementos relacionados.

### Ejemplo

Buscar todos los servicios que mencionan `Pasaporte`

```
GET /items/services
  ?search=Pasaporte
```

# Ordenar

La ordenación predeterminada es ascendente, pero se puede usar un signo menos `-` para revertir esto a un orden descendente. Los campos se priorizan según el orden del parámetro. La notación de puntos debe usarse cuando se ordena con valores de campos anidados.

### Ejemplos

| Campo | Descripción |
| --- | --- |
| **`date_created`** | Ordenar por fecha de creación descendente. |

| | |
|---|---|
| **sort,-date_updated** | Ordene por un campo de "ordenación", seguido de la fecha de actualización descendente. |
| **sort,-variation_ids.name** | Ordenar por un campo de "ordenación", seguido del nombre de una variación anidada. |

```
GET /items/services
    ?sort=sort,-date_updated,author.name

// or

GET /items/services
    ?sort[]=sort
    &sort[]=-date_updated
    &sort[]=-variation_ids.name
```

# Límite

Establezca el número máximo de artículos que se devolverán. El límite predeterminado se establece en `100`.

**Ejemplos**

- Consigue los primeros 200 artículos: `200`
- Obtener el número máximo permitido de artículos: `-1`

```
GET /items/services
    ?limit=200
```

> **Límites elevados y rendimiento**
>
> Dependiendo del tamaño de la colección, obtener la cantidad máxima de elementos puede provocar una disminución del rendimiento o tiempos de espera más largos. Úselo con precaución.

# Omitir

Omita los primeros `n` elementos de la respuesta. Este parámetro se puede utilizar para la paginación.

**Ejemplos**

- Consigue los objetos 101—200

```
GET /items/services
  ?offset=100
```

# Página

Una alternativa a `offset`. La página es una forma de establecer `offset` bajo el capó mediante el cálculo de **`limit * page`**. La primera página que está indexada es `1`.

**Ejemplos**

- Obtener artículos 101-200

```
GET /items/services
  ?page=2
```

# Agregación y agrupación

## Agregación

Las funciones de agregado o `aggregate` le permiten realizar cálculos en un conjunto de valores, devolviendo un único resultado.

Las siguientes funciones de agregación están disponibles en Catalogo:

| Nombre | Descripción |
|---|---|
| **`count`** | Cuenta cuántos elementos hay |
| **`countDistinct`** | Cuenta cuántos objetos únicos hay |
| **`sum`** | Suma los valores del campo dado |
| **`sumDistinct`** | Suma los valores únicos en el campo dado |
| **`avg`** | Obtener el valor promedio del campo dado |
| **`avgDistinct`** | Obtener el valor promedio de los valores únicos en el campo dado |

| Nombre | Descripción |
| --- | --- |
| `min` | Devuelve el valor más bajo del campo |
| `max` | Devuelve el valor más alto del campo |

```
GET /items/services
    ?aggregate[count]=*
```

# Agrupación

De forma predeterminada, las funciones de agregación anteriores se ejecutan en todo el conjunto de datos. Para permitir informes más flexibles, puede combinar la agregación anterior con la agrupación. La agrupación permite ejecutar las funciones de agregación en función de un valor compartido. Esto permite cosas como "Calificación promedio por mes" o "Ventas totales de artículos en la categoría de jeans".

La consulta permite agrupar varios campos simultáneamente. En combinación con las funciones, esto permite la generación de informes agregados por año-mes-fecha.groupBy

```
GET /items/services
    ?aggregate[count]=name,description
    &groupBy[]=institution_id
    &groupBy[]=date_updated
```

# Profundo

`deep` le permite establecer cualquiera de los otros parámetros de consulta (excepto `fields` y `deep` en sí) en un conjunto de datos relacional anidado.

**Ejemplos**

- Limite las variaciones relacionadas anidados a 3

```
{
  "variation_ids": {
    "_limit": 3
  }
}
```

- Solo obtenga 3 artículos relacionados, con solo el comentario mejor calificado anidado

```
{
  "variation_ids": {
    "_limit": 3,
    "result_ids": {
      "_sort": "name",
      "_limit": 1
    }
  }
}
```

Hay dos sintaxis disponibles:

```
GET /items/services
  ?deep[institution_id][_filter][acronym][_eq]=OGTIC

// or

GET /items/services
  ?deep={ "institution_id": { "_filter": { "acronym": { "_eq": "OGTIC" }}}}
```

# Alias

Los `alias` le permiten cambiar el nombre de los campos sobre la marcha y solicitar el mismo conjunto de datos anidados varias veces utilizando diferentes filtros.

> **Campos anidados**
>
> Solo es posible asignar alias a los campos del mismo nivel. El alias de los campos anidados, por ejemplo, `field.nested`, no funcionará.

```
GET /items/services
  ?alias[name]=nombre
  &alias[description]=descripcion
  &deep[institution_id][_filter][acronym][_eq]=OGTIC
```

# Exportar

Guarde la respuesta del API actual en un archivo. Acepta un tipo de los siguientes: `csv`, `json`, `xml`, `yaml`.

```
GET /items/services
  ?export=type
```

**Ejemplos:**

```
?export=csv
?export=json
?export=xml
?export=yaml
```

# Funciones

Las funciones permiten la modificación "en vivo" de los valores almacenados en un campo. Las funciones se pueden usar en cualquier parámetro de consulta que normalmente proporcionaría una clave de campo, incluidos los campos, la agregación y el filtro.

Las funciones se pueden usar envolviendo la clave de campo en una sintaxis similar a JavaScript, por ejemplo:

- `timestamp` -> `year(timestamp)`

## Funciones de fecha y hora

| Filtro | Descripción |
|---|---|
| **`year`** | Extraer el año de un campo de fecha y hora/fecha/marca de tiempo |
| **`month`** | Extraer el mes de un campo datetime/date/timestamp |
| **`week`** | Extraer la semana de un campo datetime/date/timestamp |
| **`day`** | Extraer el día de un campo de fecha y hora/fecha/marca de tiempo |
| **`weekday`** | Extraer el día de la semana de un campo datetime/date/timestamp |

| Filtro | Descripción |
| --- | --- |
| **hour** | Extraer la hora de un campo datetime/date/timestamp |
| **minute** | Extraer el minuto de un campo de fecha y hora/fecha/marca de tiempo |
| **second** | Extraiga el segundo de un campo de fecha y hora/fecha/marca de tiempo |

# Funciones de matriz

| Filtro | Descripción |
| --- | --- |
| **count** | Extraer el número de elementos de una matriz JSON o un campo relacional |

```
GET /items/services
    ?fields=id,name,weekday(date_created)
    &filter[year(date_created)][_eq]=2024
```

Guía paso a paso (Step by Step)

# Reglas de filtrado

Los permisos, la validación y el parámetro filter de la API se basan en una estructura JSON específica para definir sus reglas. En esta página se describe la sintaxis para crear reglas de filtro planas, relacionales o complejas.

# Sintaxis

- Campo: cualquier campo raíz, campo relacional u operador lógico válido
- Operador: cualquier operador de filtro válido
- Valor: cualquier valor estático válido o variable dinámica

```
{
  <field>: {
    <operator>: <value>
  }
}
```

**Ejemplos**

```
{
  "title": {
    "_contains": "Catalogo"
  }
}
```

```
{
  "owner": {
    "_eq": "$CURRENT_USER"
  }
}
```

```
{
  "datetime": {
    "_lte": "$NOW"
  }
}
```

```
{
  "category": {
    "_null": true
  }
}
```

# Operadores de filtro

| Nombre del operador (en la aplicación) | Operador | Descripción |
|---|---|---|
| Igual a | `_eq` | Igual a |
| No es igual a | `_neq` | No es igual a |
| Menor que | `_lt` | Menor que |
| Menor o igual que | `_lte` | Menor o igual que |
| Mayor que | `_gt` | Mayor que |
| Mayor o igual que | `_gte` | Mayor o igual que |
| Es uno de los | `_in` | Coincide con cualquiera de los valores |
| No es uno de los | `_nin` | No coincide con ninguno de los valores |
| Es nulo | `_null` | Es nulo `null` |
| No es nulo | `_nnull` | No es nulo `null` |
| Contiene | `_contains` | Contiene la subcadena |
| Contiene (no distingue entre mayúsculas y minúsculas) | `_icontains` | Contiene la subcadena que no distingue entre mayúsculas y minúsculas |
| No contiene | `_ncontains` | No contiene la subcadena |
| Comienza con | `_starts_with` | Comienza con |
| Comienza con (sin distinción entre mayúsculas y minúsculas) | `_istarts_with` | Comienza con, sin distinción entre mayúsculas y minúsculas |
| No comienza con | `_nstarts_with` | No comienza con |
| No comienza con (sin distinción entre mayúsculas y minúsculas) | `_nistants with` | No comienza con, sin distinción entre mayúsculas y minúsculas |
| Termina con | `_ends_with` | Termina con |
| Termina con (sin distinción entre mayúsculas y minúsculas) | `_iends_with` | Termina con, sin distinción entre mayúsculas y minúsculas |
| No termina con | `_nends_with` | No termina con |

| Nombre del operador (en la aplicación) | Operador | Descripción |
|---|---|---|
| No termina con (sin distinción entre mayúsculas y minúsculas) | `_niends_with` | No termina con, sin distinción entre mayúsculas y minúsculas |
| Está entre | `_between` | El valor interseca un punto dado |
| No está entre | `_nbetween` | El valor no interseca un punto dado |
| Está vacío | `_empty` | Está vacío ( `null` o falso) |
| No está vacío | `_nempty` | No está vacío ( `null` o falso) |

# Relacional

Puede apuntar a valores relacionados anidando nombres de campo. Por ejemplo, si tiene un campo relacional Many-to-One, puede establecer una regla para el campo mediante la siguiente sintaxis.authorauthor.name

```
{
□"author": {
□□"name": {
□□□"_eq": "Rijk van Zanten"
□□}
□}
}
```

Cuando se utilizan relaciones M2M, se creará una tabla de uniones y el filtro se aplicará a la propia tabla de uniones. Por ejemplo, si tiene una colección, con una relación M2M con los autores de cada libro, es probable que la colección de cruces tenga un nombre y 3 campos: y . Para filtrar libros específicos en función de sus autores, debe pasar por la tabla de uniones y el campo :books books_authorsid books_id authors_id authors_id

```
{
□"authors": {
□□"authors_id": {
□□□"name": {
□□□□"_eq": "Rijk van Zanten"
□□□}
□□}
□}
}
```

# Operadores lógicos

Puede anidar o agrupar varias reglas mediante los operadores lógicos o. Cada operador lógico contiene una matriz de reglas de filtro, lo que permite un filtrado más complejo. Tenga en cuenta también en el ejemplo que los operadores lógicos se pueden subanidar en operadores lógicos. Sin embargo, no se pueden subanidar en reglas de filtro._and_or

```
{
  "_or": [
    {
      "_and": [
        {
          "user_created": {
            "_eq": "$CURRENT_USER"
          }
        },
        {
          "status": {
            "_in": ["published", "draft"]
          }
        }
      ]
    },
    {
      "_and": [
        {
          "user_created": {
            "_neq": "$CURRENT_USER"
          }
        },
        {
          "status": {
            "_in": ["published"]
          }
        }
      ]
    }
  ]
}
```

# Algunos vs Ninguno en uno a muchos

Al aplicar filtros a un campo de uno a varios, la plataforma usará de forma predeterminada una búsqueda de "algunos", por ejemplo, en:

```
{
  "categories": {
    "name": {
      "_eq": "Recipe"
    }
  }
```

```
}
```

El elemento primario de nivel superior se devolverá si una de las categorías tiene el nombre . Este comportamiento se puede invalidar mediante el uso de los operadores y explícitos, por ejemplo: Recipe_some_none

```
{
  "categories": {
    "_none": {
      "name": {
        "_eq": "Recipe"
      }
    }
  }
}
```

obtendrá todos los elementos principales que no tengan la categoría "Receta".

# Acceso a los elementos

Los elementos son datos individuales de la base de datos. Pueden ser cualquier cosa, desde artículos hasta comprobaciones de estado de IoT.

# El objeto Item

Los elementos tienen un esquema predefinido. El formato depende completamente de los datos y estructuras definidas.

> Datos relacionales
>
> De forma predeterminada, el objeto item no contiene datos relacionales anidados. Para recuperar datos anidados, consulte Datos relacionales y parámetros de campo.

# Obtener items

Enumera todos los elementos que existen en la colección definida.

**Request**

```
GET /items/:collection

SEARCH /items/:collection
```

Si utiliza SEARCH, puede proporcionar un objeto de consulta como cuerpo de la solicitud. Más información sobre SEARCH.

**Parámetros de consulta**

Admite todos los parámetros de consulta globales.

> Datos relacionales
>
> El parámetro Field es necesario para devolver datos relacionales anidados.

**Respuesta**

Matriz de objetos de elemento hasta límite. Si no hay elementos disponibles, los datos serán una matriz vacía.

**Ejemplo**

```
SEARCH /items/services
```

# Obtener item por ID

Obtén un elemento que exista en la colección definida.

**Request**

```
GET /items/:collection/:id
```

**Parámetros de consulta**

Admite todos los parámetros de consulta globales.

**Respuesta**

Devuelve un objeto de elemento si se proporcionó una clave principal válida.

**Ejemplo**

```
GET /items/services/15
```

# Referencia técnica

Referencia técnica

# Open API Specification

> Catalogo ofrece una API REST para consumir los datos en la base de datos. La API tiene direcciones URL predecibles orientadas a los recursos, se basa en códigos de estado HTTP estándar y utiliza JSON para la entrada y la salida de datos.

**Open API Specification**

Catalogo tiene un endpoint para mostrar las especificaciones Open API para describir y/o detallar el API REST. La idea es que los desarrolladores puedan hacer pruebas de manera más intuitivas.

Para obtener las especificaciones pueden usar el siguiente endpoint:

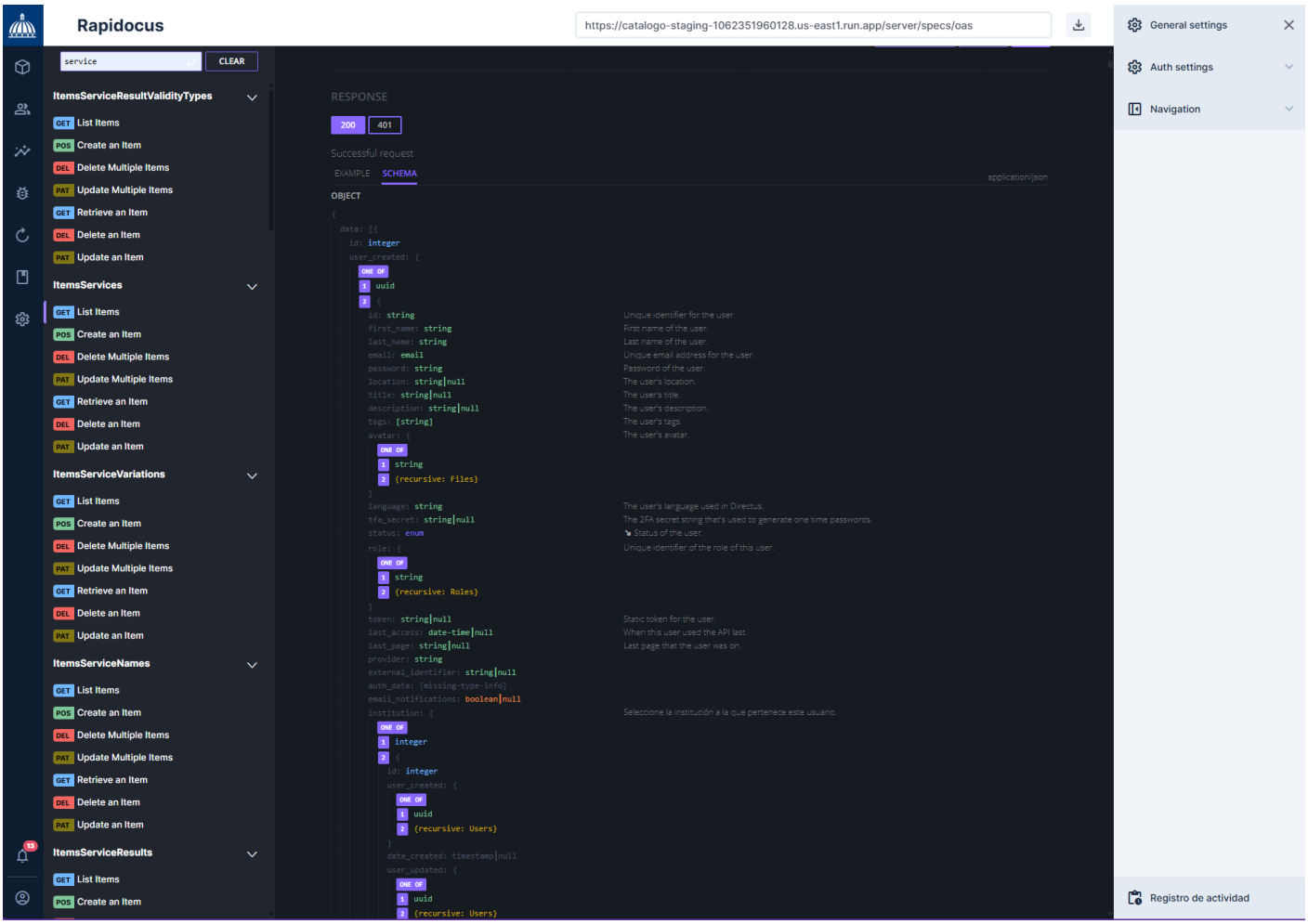
```
GET /server/specs/oas
```

Una vez tengan las especificaciones del API, pueden usar el Editor de Swagger u otro, pegar las especificaciones en el editor para generar dinámicamente todos los endpoints existentes a los que su usuario tiene acceso.

Referencia técnica

# Documentación Interactiva (Rapidocus)

La documentación interactiva es una herramienta integrada en la plataforma del Catálogo de Servicios que permite a los desarrolladores **explorar, probar y entender los endpoints del API** de manera visual e intuitiva, sin necesidad de instalar aplicaciones externas.




Esta sección proporciona un espacio donde cualquier usuario técnico puede aprender rápidamente cómo funciona el API, ver cómo se estructuran las llamadas y comprender de forma más profunda las respuestas y parámetros disponibles, reduciendo el tiempo de aprendizaje.

## ¿Qué es la documentación interactiva?

Es una interfaz dinámica disponible desde el panel del Catálogo que permite:

- Navegar por todos los endpoints del API organizados por categorías.
- Consultar descripciones, parámetros y respuestas de cada endpoint.
- Probar los endpoints directamente desde la interfaz (con un botón "Try"), utilizando tus credenciales y tokens válidos.
- Ver ejemplos de solicitudes y respuestas en tiempo real.
- Ajustar cómo se visualiza la información (filtros, formato de código, organización de secciones).

# ¿Cómo acceder?

1. Ingresa al panel del Catálogo con tus credenciales.
2. Dirígete a la URL **`/admin/rapidocus`**
3. Selecciona la categoría de endpoints que quieras explorar (por ejemplo, **`Services`**).

# Funciones principales

- **Exploración por categorías**: Los endpoints están organizados según los recursos del API, para facilitar la búsqueda.
- **Pruebas en vivo (Try)**: Permite ejecutar solicitudes GET, POST, PUT o DELETE directamente, utilizando tus tokens y parámetros, y ver la respuesta al instante.
- **Visualización personalizable**: Puedes cambiar la forma en que se presentan los resultados (código resaltado, JSON plano o estructurado, etc.).
- **Filtros avanzados**: Permite buscar endpoints o parámetros específicos con un cuadro de búsqueda.
- **Copiar ejemplos**: Incluye botones para copiar al portapapeles ejemplos de requests y respuestas.

# ¿Por qué usarla?

- Evita configurar herramientas externas como Postman para pruebas iniciales.
- Acelera el aprendizaje de cómo funciona cada endpoint y qué datos retorna.
- Facilita la depuración durante el desarrollo, ya que puedes confirmar rápidamente si el API está respondiendo como se espera.

# Recomendaciones de uso

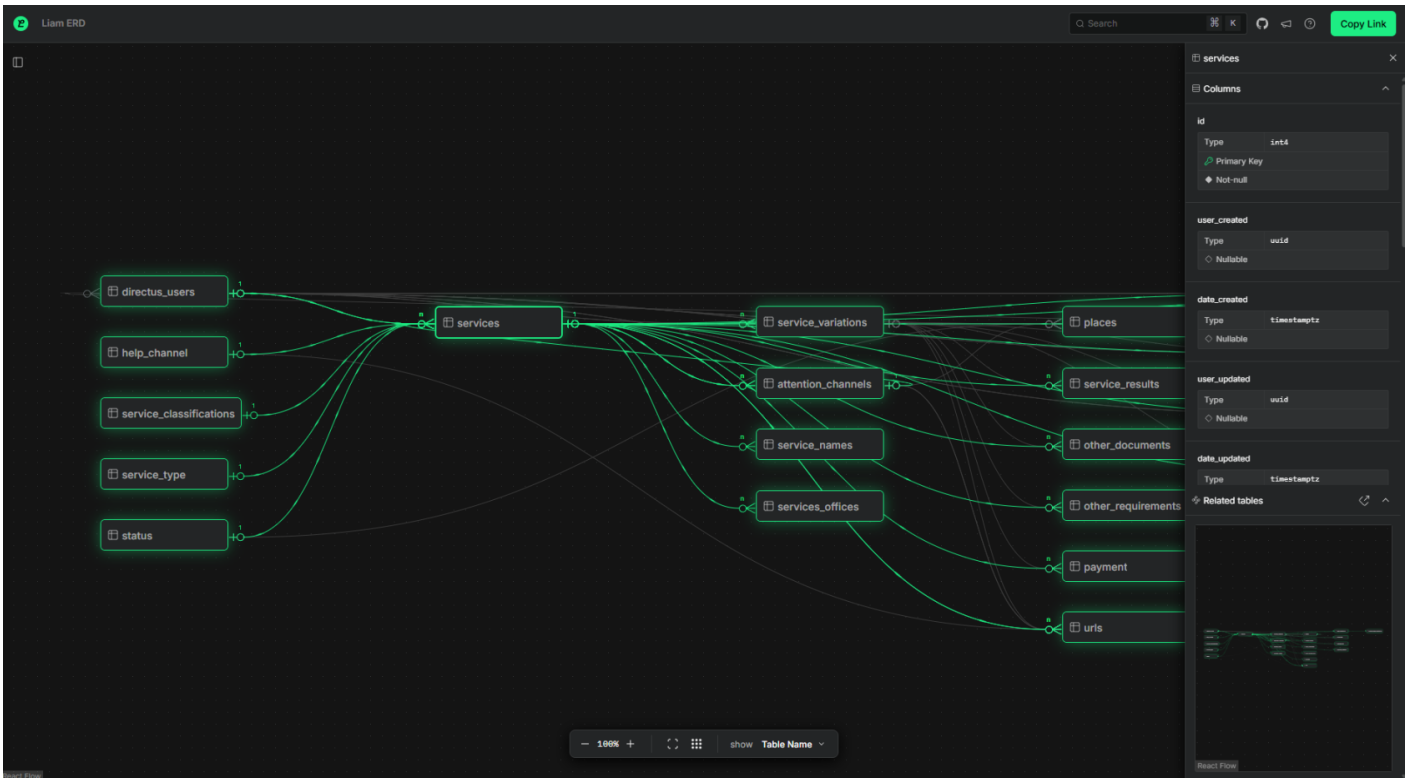
- Siempre inicia sesión con un usuario válido antes de probar endpoints protegidos.
- Utiliza primero el entorno de **sandbox** para realizar pruebas, antes de interactuar con producción.

- Respeta los límites de consumo (SLA) al ejecutar múltiples pruebas.
- Para integraciones reales, utiliza la guía "Quick Start" y la referencia completa del API (OpenAPI) en combinación con esta herramienta.

---

Referencia técnica

# Diagrama entidad-relación (ERD)




Esta sección proporciona una vista estructurada de las relaciones entre las entidades clave del sistema de Catálogo de Servicios. El diagrama resulta útil para entender cómo se conectan los diferentes componentes del sistema, facilitando tareas de integración, depuración o generación de reportes.

## □□ ¿Qué muestra este diagrama?

- Relación entre servicios y sus variaciones
- Asociación con canales de atención, oficinas, requisitos, documentos, resultados y URLs
- Vínculos con clasificaciones, tipos, estados y usuarios responsables

## □□ Acceso al diagrama interactivo

Puedes acceder al diagrama en tiempo real desde el siguiente enlace:

□□ Ver ERD interactivo

Este diagrama es interactivo y permite navegar visualmente por las relaciones entre las tablas, haciendo clic sobre cada entidad para ver sus campos y claves foráneas.

## Casos de uso comunes

- Comprender la estructura de datos para integraciones externas (ej. Power BI, ETL, etc.)
- Diagnosticar errores o relaciones faltantes en servicios
- Crear consultas SQL más precisas
- Diseñar APIs o apps externas basadas en el modelo del Catálogo