

DOCUMENTACIÓN TÉCNICA

OAuth2 Identity Server

Manual de uso: puesta en marcha, panel de administración, flujos OAuth2 y API REST completa para construir cualquier cliente propio (Angular, Flutter, NestJS, etc.).

Laravel 12 · Passport 13 · Vue 3 + TypeScript · SCSS/BEM · spatie/laravel-permission · L5-Swagger
Generado el 27 de julio de 2026 (actualizado: webhooks salientes + protección anti-SSRF)

Índice

1. Qué es esta app
2. Puesta en marcha
3. Usuarios por defecto (seed)
4. Uso de la interfaz web (panel de administración)
5. Verificación en dos pasos (2FA)
6. Analíticas: sesiones, logins y actividad
7. Audit log: quién cambió qué
8. Roles, permisos y roles por app
9. Idioma (español / inglés / catalán)
10. Cómo conectar tu app (Angular, NestJS, Flutter, etc.)
11. API REST completa: construí tu propio panel admin
12. Webhooks: enterate sin hacer polling
13. Documentación interactiva (Swagger)
14. Referencia rápida de rutas

1. Qué es esta app

Es un **servidor de identidad / OAuth2** (Laravel Passport) pensado para ser el centro único de usuarios, autenticación y permisos de un conjunto de aplicaciones propias. No es una app de negocio con datos propios: su único "recurso" son los **usuarios** y sus **roles**.

Provee tres piezas:

- **Servidor de autorización OAuth2** — login, pantalla de consentimiento, y emisión de tokens con los cuatro flujos estándar: Authorization Code + PKCE, Client Credentials, Password Grant y Personal Access Tokens.
- **Panel de administración en Vue 3 + TypeScript** — la UI que trae la app "de fábrica", servida por el propio Laravel, autenticada por sesión. Estilos en SCSS con nomenclatura BEM ([resources/scss/](#)), sin frameworks de utility-classes.
- **API REST completa ([/api/*](#))** — el *mismo* conjunto de funciones del panel Vue (usuarios, roles por app, clientes OAuth, personal access tokens, apps autorizadas), pero autenticada 100% por token OAuth2 (sin sesión ni cookies). Con esto se puede construir **otro admin entero** en Angular, Flutter, o cualquier otro cliente — no hace falta usar la SPA Vue.

Idea central: cada aplicación externa que quiera loguear usuarios se registra acá como un *cliente OAuth* ("app"). Un mismo usuario puede tener un rol global (admin/user) y, además, un rol distinto *dentro de cada app* (por ejemplo: admin en la app de Angular, usuario normal en la de NestJS).

Dos frentes, misma API subyacente: la SPA Vue (sesión + cookies, pensada para uso interno en el navegador de este mismo servidor) y la API REST (Bearer token, pensada para cualquier cliente externo) exponen las *mismas* capacidades. Elegí el que corresponda: sesión para la UI que ya viene incluida, token para construir la tuya propia.

2. Puesta en marcha

Requisitos

- PHP 8.4+, Composer
- Node.js + npm
- SQLite (por defecto, cero configuración — se puede cambiar en `.env`)

Instalación (primera vez)

```
composer install
npm install
php artisan migrate --seed
npm run build
```

Frontend: TypeScript (`npm run type-check` corre `vue-tsc` sin emitir archivos) y SCSS compilado por Vite (paquete `sass`) desde `resources/scss/app.scss` — no hay ningún framework de CSS utility-first instalado.

Levantar todo en desarrollo

```
composer run dev
```

Este comando levanta en paralelo: el servidor PHP (`php artisan serve`), Vite en modo watch, la cola de trabajos y los logs. La app queda disponible en `http://localhost:8000` .

Reiniciar todo de cero: `php artisan migrate:fresh --seed` borra la base y vuelve a crear los usuarios/clientes de ejemplo descritos en la sección 3. Los `client_secret` de los clientes OAuth de ejemplo se imprimen en la consola en ese momento — solo se muestran una vez.

Email: por defecto `MAIL_MAILER=log` — los emails (hoy, solo el de recuperación de contraseña, ver sección 4) se escriben en `storage/logs/laravel.log` en vez de enviarse de verdad, para no necesitar SMTP en desarrollo. Para producción, configurará un mailer real en `.env` (`MAIL_MAILER` , `MAIL_HOST` , etc. — cualquiera soportado por Laravel). El envío va encolado (`ShouldQueue`), así que también necesitás un worker corriendo (`php artisan queue:work` , o `composer run dev` en desarrollo).

3. Usuarios por defecto (seed)

El seeder (`database/seeder/DatabaseSeeder.php`) crea una cuenta admin y algunos usuarios de ejemplo para poder probar la app de inmediato:



Administrador

Email: `admin@example.com`

Password: `password`

Rol global: `admin`

Acceso al panel de administración de usuarios.



Usuario normal

Email: `carla@example.com`

Password: `password`

Rol global: `user`

Sin acceso al panel admin. Es **admin** dentro de la app "Demo SPA" (rol por-app de ejemplo). Junto con `martin@example.com` y `bea@example.com`, solo están para que la sección de analíticas tenga datos de entrada.

El seeder también crea `nuevo@example.com` / `password` sin aprobar, para que la pestaña "**Pendientes**" del panel de usuarios (sección 4) no arranque vacía.

Instalación sin seed (producción): si nunca corriste `db:seed`, no hay ningún usuario en la base. En ese caso, la **primera** cuenta que se registre por `/register` queda automáticamente como **admin**, aprobada y con la sesión iniciada — no hace falta que nadie más la apruebe, porque todavía no existe ningún admin que pudiera hacerlo. La pantalla de registro avisa de esto con un mensaje cuando detecta que la base está vacía. A partir del segundo registro, vuelve a regir el flujo normal (pendiente de aprobación por un admin).

Cientes OAuth (apps) de ejemplo

El seeder también registra apps de prueba, una por cada flujo soportado. El `client_id` es fijo por ejecución (UUID nuevo cada `migrate:fresh`); el `client_secret` solo se muestra por consola al sembrar — si lo perdiste, hay que crear un cliente nuevo desde el panel (sección 4) o volver a sembrar.

Nombre	Grant type	Uso
Demo SPA (Authorization Code + PKCE)	<code>authorization_code</code>	Cliente público (sin secret), para apps tipo SPA/móvil.
Demo Service (Client Credentials)	<code>client_credentials</code>	Comunicación máquina-a-máquina, sin usuario.
Demo First-Party App (Password Grant)	<code>password</code>	Apps propias de confianza, login directo usuario/contraseña.
Swagger UI	<code>authorization_code</code> , <code>client_credentials</code>	Para probar el botón "Authorize" de la documentación interactiva.
Personal Access Client	<code>personal_access</code>	Interno — necesario para poder emitir Personal Access Tokens.

4. Uso de la interfaz web (panel de administración)

Todo se maneja desde `http://localhost:8000` (Vue Router, sin recargar página).

Modo oscuro: el ícono de sol/luna en la barra de navegación alterna entre claro y oscuro. La preferencia se guarda en `localStorage` (por navegador, no por cuenta) y, si nunca la tocaste, arranca siguiendo el modo del sistema operativo. El cambio se aplica antes del primer render — no hay parpadeo del tema equivocado al cargar la página.

Login / Registro

`/login` y `/register`. El registro crea la cuenta con rol global `user` automáticamente, pero **no loguea al usuario**: queda pendiente de aprobación. La pantalla muestra un mensaje de "Cuenta creada" y el login devuelve error hasta que un admin la aprueba desde el panel de usuarios (sección 4). **Excepción:** si la base de usuarios está vacía, esa primera cuenta se crea como admin y queda logueada de una (ver sección 3).

Rate limiting: `POST /web-api/login` admite 5 intentos por minuto por combinación de email + IP (frena fuerza bruta de contraseña sin afectar a otros usuarios que compartan IP), y `POST /web-api/register` 5 por minuto por IP. Al superar el límite, la respuesta es `429 Too Many Requests` con un header `Retry-After`.

Recuperación de contraseña

Desde `/login` → "¿Olvidaste tu contraseña?" (`/forgot-password`), cualquiera puede pedir un enlace de restablecimiento con solo su email. La respuesta es **siempre la misma**, exista o no una cuenta con ese email — lo contrario permitiría enumerar qué emails están registrados probando uno por uno. Si la cuenta existe, se manda un email (ver la nota sobre `MAIL_MAILER` en la sección 2) con un enlace a `/reset-password?token=...&email=...`, válido durante **60 minutos** y de un solo uso.

Al restablecer la contraseña se cierran todas las sesiones activas (los tokens OAuth vigentes de esa cuenta, igual que "Cerrar todas" en Sesiones activas — sección 4) y queda un `user.password_reset` en el audit log (sección 7). El motivo más común para resetear una contraseña es sospechar que se filtró, así que no tiene sentido dejar vigente lo que se emitió con la vieja.

Rate limiting: tanto pedir el enlace como confirmar el restablecimiento admiten 5 intentos por minuto por combinación de email + IP.

Dashboard

`/` — accesos rápidos a Clientes OAuth, Personal Access Tokens, Apps autorizadas y (si sos admin) Usuarios.

Clientes OAuth

`/clients` — cada usuario ve y gestiona *sus propios* clientes (apps que registró). Al crear uno se elige:

- **Authorization Code + PKCE:** hay que indicar redirect URIs; opcionalmente "confidencial" (con secret) o público (solo PKCE, sin secret).
- **Client Credentials:** siempre confidencial, sin redirect URI.

- **Password Grant:** para apps propias de confianza.

El `client_secret` se muestra **una sola vez**, justo después de crear el cliente. Copiarlo en ese momento.

Cada cliente tiene, además, un botón "**Webhooks**" para registrar endpoints que reciben avisos cuando pasa algo relevante para esa app (ver sección 12) — sin tener que hacer polling.

Rotar secret. Los clientes confidenciales (con `client_secret`) tienen un botón "**Rotar secret**". Genera un secret nuevo y lo muestra una sola vez, igual que al crear el cliente — pero el secret **anterior sigue siendo válido durante 48 horas** (grace period), así que podés actualizar la config de tu app sin downtime: mientras dure el grace period, ambos secrets (viejo y nuevo) funcionan contra `POST /oauth/token`. Pasadas las 48 horas, o si rotás de nuevo antes, el secret anterior deja de servir. No aplica a clientes públicos (Authorization Code + PKCE sin secret) — no hay nada que rotar.

Personal Access Tokens

`/tokens` — tokens tipo "API key" para el propio usuario, sin pasar por el flujo de consentimiento. Se elige nombre y scopes. El token completo también se muestra una única vez.

Apps autorizadas

`/authorized-apps` — lista de apps de terceros a las que el usuario le dio acceso a su cuenta vía

`/oauth/authorize`, con opción de revocar el acceso en cualquier momento.

Usuarios (solo admins)

`/admin/users` — alta, edición (nombre, email, contraseña, rol global, activo/desactivado) y baja de cualquier usuario del sistema. Protecciones incluidas: no podés borrar a vos mismo, no podés quitarte el rol admin a vos mismo, no se puede borrar al último administrador. Un usuario desactivado no puede iniciar sesión.

Borrado = papelera (soft delete). "Borrar" un usuario no lo elimina de la base: lo oculta (desaparece de logins, listados y de cualquier chequeo de permisos) pero queda recuperable. El botón "**Ver papelera**" arriba a la derecha del listado muestra los usuarios borrados, con dos acciones por fila: **Restaurar** (vuelve a aparecer tal cual estaba) o **Borrar definitivamente** (ahí sí, sin vuelta atrás).

Registros pendientes de aprobación. Arriba a la derecha del listado hay tres pestañas: *Ver activos*, **Pendientes** (con un contador si hay alguno) y *Ver papelera*. Todo el que se registra por `/register` aparece en "Pendientes" hasta que un admin lo **aprueba** (ahí ya puede iniciar sesión) o lo **rechaza** (va a la papelera, igual que un borrado normal — se puede restaurar). Un usuario dado de alta directamente por un admin (el formulario "Nuevo usuario" de esta misma pantalla) queda aprobado automáticamente.

Sesiones activas. Dentro de "Editar" (junto a "Roles por app") hay una sección "Sesiones activas" con los tokens OAuth vigentes de ese usuario — qué apps tienen acceso a su cuenta ahora mismo, con scope y fecha de emisión. Cada sesión se puede revocar individualmente, o todas de una con "Cerrar todas". Pensado para incident response: si un usuario reporta una cuenta comprometida (o algo raro aparece en Analíticas/Audit log), un admin puede cortar el acceso a todo sin esperar a que expire solo.

Seguridad: verificación en dos pasos

`/security` — cualquier usuario (no solo admins) puede activar 2FA acá. Ver sección 5.

Analíticas (solo admins)

`/admin/analytics` — ver sección 6.

Audit log (solo admins)

`/admin/audit-log` — ver sección 7.

5. Verificación en dos pasos (2FA)

Cualquier usuario (no hace falta ser admin) puede activar 2FA con TOTP desde [/security](#) — compatible con Google Authenticator, Authy, 1Password, o cualquier app que hable el estándar TOTP (RFC 6238). Es una capa extra sobre la contraseña: una vez activo, el login pide además un código de 6 dígitos.

Activación

1. El servidor genera un secreto (todavía no activo) y un código QR para escanear.
2. El usuario escanea el QR con su app autenticadora (o copia el secreto a mano si no puede escanear).
3. Confirma con un código real generado por la app — recién en ese momento queda activo de verdad. Un secreto generado pero nunca confirmado no habilita nada.
4. El servidor muestra **8 códigos de recuperación**, una sola vez. Cada uno sirve para un único login — son la salida de emergencia si se pierde el teléfono con la app autenticadora.

Login con 2FA activo

El login pasa a ser de dos pasos: primero email + contraseña (esto **no** crea sesión todavía); si el usuario tiene 2FA activo, el servidor responde pidiendo el código en vez de loguear directo. Recién con un código válido (TOTP o de recuperación) se crea la sesión. Un código de recuperación usado se invalida al instante — no sirve una segunda vez.

Rate limiting: el segundo paso del login ([/web-api/login/2fa](#)), confirmar la activación ([/2fa/confirm](#)), desactivar ([/2fa/disable](#)) y regenerar códigos de recuperación ([/2fa/recovery-codes](#)) admiten 6 intentos por minuto — el espacio de un código TOTP de 6 dígitos es chico como para no limitarlo.

Desactivar 2FA o regenerar códigos de recuperación: ambas acciones, desde [/security](#) , piden confirmar la contraseña actual. Desactivar borra el secreto y los códigos de recuperación por completo; para volver a activarlo hay que escanear un QR nuevo (no se reutiliza el secreto anterior). Regenerar invalida los 8 códigos viejos y muestra 8 nuevos.

Password Grant y 2FA: si el usuario tiene 2FA activo, [POST /oauth/token](#) con [grant_type=password](#) exige además un campo [otp](#) con un código TOTP o de recuperación válido, en el mismo request — sin él, o con uno inválido, el servidor responde [400 invalid_grant](#) igual que con una contraseña incorrecta (no delata si el problema fue la contraseña o el 2FA). Los usuarios sin 2FA no notan ningún cambio.

```
curl -X POST /oauth/token -d grant_type=password \
  -d client_id=... -d client_secret=... \
  -d username=ana@example.com -d password=... \
  -d otp=123456 -d scope="users:read"
```

Cada activación, desactivación y regeneración de códigos queda registrada en el audit log (sección 7).

6. Analíticas: sesiones, logins y actividad

Pantalla de monitoreo para el admin: cada vez que Passport emite un access token (cualquier grant type) o alguien inicia sesión en el panel, queda un registro en `login_events`. Con eso se arman:

- **KPIs:** usuarios totales, sesiones activas (tokens OAuth vigentes, no revocados ni expirados), logins de hoy, logins de los últimos 7 días, tasa de fallos de login y tasa de aprobación OAuth (últimos 7 días).
- **Logins por día** (últimos 7/30/90 días, elegible arriba a la derecha): barras apiladas — logins de apps (OAuth) vs. logins al panel admin (sesión) — con tooltip al pasar el mouse sobre cada día.
- **Logins por app:** qué apps se usan más.
- **Usuarios más activos:** ranking por cantidad de logins en el período.
- **Funnel de autorización OAuth:** por cada cliente, cuántas veces se mostró la pantalla de consentimiento (`/oauth/authorize`), cuántas se aprobaron y cuántas se denegaron, con la tasa de aprobación resultante.
- **Tokens activos por cliente:** snapshot en vivo de tokens OAuth vigentes (no revocados, no expirados), agrupados por app.
- **Uso de scopes:** qué scopes (`users:read`, `account:write`, etc.) aparecen más entre los tokens emitidos en el período.
- **Actividad reciente:** tabla con los últimos eventos (usuario, app, tipo, IP, cuándo).

Exportar PDF: el botón "Exportar PDF" arriba a la derecha abre el diálogo de impresión del navegador con una hoja de estilos dedicada — oculta la barra de navegación y los controles, fuerza tema claro y expande la tabla de actividad reciente completa. No genera el PDF en el servidor: es el navegador el que lo arma ("Guardar como PDF" en el diálogo de impresión), así que no depende de ninguna librería ni servicio adicional.

Tasa de fallos de login solo cuenta credenciales inválidas en el login de sesión (panel admin) — no cuenta intentos contra una cuenta desactivada o pendiente de aprobación, porque eso es un estado de cuenta, no una señal de ataque de fuerza bruta.

El seeder de demo genera 30 días de actividad sintética para que estos gráficos no arranquen vacíos. En un despliegue real, los datos son 100% eventos reales — no hay nada que "activar". El funnel de autorización y la tasa de fallos de login sí arrancan vacíos incluso con el seed (son eventos nuevos, no incluidos en la actividad sintética) hasta que ocurran de verdad.

Misma info disponible por API: `GET /api/analytics/summary?days=30` (scope `users:read`, requiere el permiso `users.view` — igual que el resto del panel admin).

7. Audit log: quién cambió qué

Pantalla de auditoría para el admin (</admin/audit-log>): registra automáticamente los cambios que importan desde el punto de vista de seguridad/gobernanza — no hace falta activar nada, queda un renglón cada vez que ocurre alguno de estos eventos:

Acción	Cuándo se registra	Qué guarda en "Detalles"
<code>user.created</code>	Un admin da de alta un usuario desde el panel.	Nombre, email y rol asignado.
<code>user.updated</code>	Un admin edita nombre, email, rol, estado activo/inactivo o contraseña de un usuario.	Solo los campos que efectivamente cambiaron (antes → después). La contraseña nunca se guarda en texto plano, solo si cambió o no.
<code>user.deleted</code>	Un admin borra (soft delete) un usuario.	—
<code>user.restored</code>	Un admin restaura un usuario desde la papelera.	—
<code>user.force_deleted</code>	Un admin borra definitivamente un usuario de la papelera.	—
<code>user.approved</code>	Un admin aprueba un registro pendiente.	—
<code>user.app_role_assigned</code>	Un admin asigna un rol por-app a un usuario.	Client y rol asignado.
<code>user.app_role_revoked</code>	Un admin quita un rol por-app a un usuario.	Client y rol quitado.
<code>client.created</code>	Se registra un cliente OAuth nuevo (panel o API).	Grant type y si es confidencial.
<code>client.revoked</code>	Se revoca un cliente OAuth (panel o API) — cascada a sus tokens.	Cuántos tokens quedaron revocados.
<code>user.2fa_enabled</code>	Un usuario activa 2FA.	—
<code>user.2fa_disabled</code>	Un usuario desactiva 2FA.	—
<code>user.2fa_recovery_regenerated</code>	Un usuario regenera sus códigos de recuperación.	—
<code>user.session_revoked</code>	Un admin revoca un token OAuth ("sesión") de un usuario.	—
<code>user.sessions_revoked_all</code>	Un admin cierra todas las sesiones activas de un usuario de una vez.	Cuántos tokens quedaron revocados.
<code>client.webhook_created</code>	Se registra un webhook nuevo en un cliente OAuth.	URL y eventos suscritos.
<code>client.webhook_deleted</code>	Se borra un webhook.	URL.
<code>client.secret_rotated</code>	El dueño de un cliente confidencial rota su <code>client_secret</code> .	Hasta cuándo sigue valiendo el secret anterior.

<code>user.password_reset</code>	Un usuario restablece su contraseña vía "¿Olvidaste tu contraseña?".	Cuántas sesiones quedaron revocadas.
----------------------------------	--	--------------------------------------

Cada evento guarda además quién lo hizo (o "(sistema)" si no hay actor), IP, y cuándo. Es filtrable por sujeto (Usuarios / Clientes OAuth) y por acción, y está paginado.

Qué NO cubre todavía: cambios de `name` / `redirect_uris` en un cliente OAuth existente (solo se audita creación y revocación), y acciones de auto-servicio de un usuario sobre sus propios clientes/tokens/apps autorizadas (ese flujo no pasa por rutas de admin).

Requiere el permiso `audit.view` (nuevo — separado de `users.*` porque también cubre clientes OAuth, no solo usuarios). Se lo asigna automáticamente al rol `admin`. Misma info por API: `GET /api/audit-log` (scope `users:read`).

8. Roles, permisos y roles por app

Roles globales

Gobiernan el panel de administración de esta app (`spatie/laravel-permission`):

Rol	Permisos
<code>admin</code>	<code>users.view</code> , <code>users.create</code> , <code>users.update</code> , <code>users.delete</code> , <code>audit.view</code>
<code>user</code>	Ninguno extra — gestiona su propio perfil, sus clientes OAuth y sus tokens.

Roles por app

Independientes del rol global: un admin le puede asignar a cualquier usuario un rol (`admin` o `user`) **específico de una app registrada**, desde el panel de edición de usuario ("Roles por app"). Sirve para que, por ejemplo, un usuario sea `admin` dentro de la app Angular pero solo `user` dentro de la app NestJS — sin tocar su rol global.

Esa información viaja en la respuesta de la API (`app_roles` , ver sección 11): cada app ve únicamente el rol que ese usuario tiene *en ella*, resuelto automáticamente según el `client_id` del token usado en la request.

9. Idioma (español / inglés / catalán)

El selector está arriba a la derecha en la barra de navegación (**ES / EN / CA**). Cambiarlo hace un POST a `/web-api/locale`, guarda el idioma en la sesión y recarga la página — el servidor decide el idioma con el que renderiza esa carga (mensajes de validación/authenticación vía `laravel-lang`, más los textos propios de la app en `lang/{es,en,ca}/messages.php`).

Idioma	Mensajes del servidor	Interfaz (SPA Vue)
ES (español, por defecto)	✓	✓
EN (inglés)	✓	✓ — traducción completa (nav, botones, formularios, tablas, gráficos)
CA (catalán)	✓	✓ — traducción completa

La interfaz usa `vue-i18n` con diccionarios en `resources/js/i18n/{es,en,ca}.js` (168 claves cada uno). El locale de arranque del frontend se inyecta desde el servidor (`window.__LOCALE__` en `app.blade.php`), así que la SPA nace ya en el idioma correcto sin parpadeo.

10. Cómo conectar tu app (Angular, NestJS, Flutter, etc.)

Paso 1 — Registrar la app

Desde `/clients`, crear un cliente OAuth eligiendo el grant type según el tipo de app:

Tipo de app	Grant recomendado
SPA (Angular, React) servida al navegador	Authorization Code + PKCE, cliente público
Backend con secret propio (NestJS, Express)	Authorization Code, cliente confidencial , o Client Credentials si no hay usuario final
Servicio interno máquina-a-máquina	Client Credentials

Paso 2 — Login del usuario (Authorization Code + PKCE, ej. Angular)

Redirigir el navegador a:

```
GET /oauth/authorize
  ?client_id={client_id}
  &redirect_uri={tu_callback}
  &response_type=code
  &scope=users:read users:write
  &state={valor_random}
  &code_challenge={challenge_pkce}
  &code_challenge_method=S256
```

El usuario loguea (o ya tiene sesión) y aprueba el acceso. Passport redirige a `redirect_uri` con `?code=...&state=...`.

Paso 3 — Canjear el código por un token

```
curl -X POST http://localhost:8000/oauth/token \
  -d grant_type=authorization_code \
  -d client_id={client_id} \
  -d code={code} \
  -d redirect_uri={tu_callback} \
  -d code_verifier={verifier_pkce}
```

Alternativa — Client Credentials (ej. NestJS backend-a-backend)

```
curl -X POST http://localhost:8000/oauth/token \
  -d grant_type=client_credentials \
  -d client_id={client_id} \
  -d client_secret={client_secret} \
  -d scope="users:read"
```

Un token de `client_credentials` no tiene usuario asociado: actúa como "servicio de confianza" y puede listar/crear/editar usuarios sin más chequeo que el scope del token.

11. API REST completa: construí tu propio panel admin

Todo lo que hace la SPA Vue por sesión también está disponible por token OAuth2. Esto significa que un cliente Angular, Flutter, o lo que sea, puede ser un **admin completo** — no solo consultar usuarios — sin depender de la sesión de Laravel. Todos los endpoints requieren `Authorization: Bearer {access_token}`.

Usuarios y roles (scopes `users:read` / `users:write`)

Método	Ruta	Descripción
GET	/api/user	Whoami: usuario del token actual, con roles y app_roles .
GET	/api/users	Listar usuarios (requiere users.view , salvo client_credentials).
POST	/api/users	Crear usuario (requiere users.create).
GET	/api/users/{id}	Ver un usuario. Siempre permitido sobre uno mismo.
PUT	/api/users/{id}	Editar. Uno mismo puede cambiar nombre/email/contraseña; rol/estado y otros usuarios requieren users.update .
DELETE	/api/users/{id}	Borrar — soft delete, va a la papelera (requiere users.delete ; no se puede autoeliminar).
GET	/api/users/trashed	Listar usuarios en la papelera (requiere users.view).
POST	/api/users/{id}/restore	Restaurar un usuario borrado (requiere users.update).
DELETE	/api/users/{id}/force	Borrado definitivo, sin vuelta atrás (requiere users.delete ; el usuario tiene que estar en la papelera).
GET	/api/users/pending	Listar registros pendientes de aprobación (requiere users.view).
POST	/api/users/{id}/approve	Aprobar un registro pendiente (requiere users.update).
GET	/api/users/{id}/app-roles	Roles del usuario por app (propio siempre permitido; otros requieren users.view).
POST	/api/users/{id}/app-roles	Asignar un rol en una app (requiere users.update , sin excepción para uno mismo).
DELETE	/api/users/{id}/app-roles/{clientId}/{role}	Quitar un rol en una app (requiere users.update).
GET	/api/users/{id}/sessions	Tokens OAuth activos de ese usuario ("sesiones") (requiere users.view).
DELETE	/api/users/{id}/sessions/{tokenId}	Revocar una sesión puntual (requiere users.update).
DELETE	/api/users/{id}/sessions	Revocar todas las sesiones activas de ese usuario (requiere users.update).
GET	/api/apps	Listar todas las apps registradas (requiere users.view) — para armar el selector de roles-por-app.
GET	/api/analytics/summary?days=30	KPIs, logins por día/app, usuarios más activos, funnel de autorización OAuth, tokens activos por cliente, uso de

		scopes y actividad reciente (requiere <code>users.view</code>).
GET	<code>/api/audit-log?action=&subject_type=&per_page=25</code>	Historial de auditoría paginado: altas/bajas de usuarios, cambios de rol, clientes OAuth creados/revocados (requiere <code>audit.view</code>).

Ejemplo de respuesta de usuario

```
{
  "id": 2,
  "name": "Carla Gómez",
  "email": "carla@example.com",
  "is_active": true,
  "roles": ["user"],
  "app_roles": ["admin"],
  "created_at": "2026-07-25T08:52:39.000000Z"
}
```

`roles` es el rol global; `app_roles` es el rol de ese usuario específicamente en la app dueña del token usado para la request (útil para que cada app decida qué mostrar/permitir sin tener su propio sistema de permisos). El campo `deleted_at` solo aparece (no-null) en las respuestas de `/api/users/trashed` .

Cuenta propia: clientes, tokens y apps autorizadas (scopes `account:read` / `account:write`)

Equivalente por token a `/web-api/clients` , `/personal-access-tokens` y `/authorized-apps` de la sección 4 — siempre sobre el usuario dueño del token (no aplica a `client_credentials`, que no tiene usuario asociado):

Método	Ruta	Descripción
GET	<code>/api/clients</code>	Listar tus clientes OAuth (apps que registraste).
POST	<code>/api/clients</code>	Registrar un cliente nuevo (devuelve <code>plain_secret</code> una sola vez).
PUT	<code>/api/clients/{id}</code>	Editar nombre/redirect URIs.
DELETE	<code>/api/clients/{id}</code>	Revocar el cliente y sus tokens.
POST	<code>/api/clients/{id}/rotate-secret</code>	Rotar el <code>client_secret</code> (solo clientes confidenciales). Devuelve <code>plain_secret</code> una sola vez; el secret anterior sigue valiendo 48h.
GET	<code>/api/personal-access-tokens</code>	Listar tus Personal Access Tokens.
POST	<code>/api/personal-access-tokens</code>	Crear uno (el token se muestra una sola vez).
DELETE	<code>/api/personal-access-tokens/{id}</code>	Revocarlo.
GET	<code>/api/authorized-apps</code>	Apps de terceros a las que le diste acceso.
DELETE	<code>/api/authorized-apps/{tokenId}</code>	Revocarle el acceso a una.
GET	<code>/api/clients/{id}/webhooks</code>	Listar los webhooks de uno de tus clientes (ver sección 12).
POST	<code>/api/clients/{id}/webhooks</code>	Registrar un webhook nuevo (devuelve

		<code>secret</code> una sola vez).
PUT	<code>/api/clients/{id}/webhooks/{webhookId}</code>	Editar URL, eventos o pausar/reanudar.
DELETE	<code>/api/clients/{id}/webhooks/{webhookId}</code>	Borrar un webhook.
GET	<code>/api/clients/{id}/webhooks/{webhookId}/deliveries</code>	Historial paginado de intentos de entrega.
POST	<code>/api/clients/{id}/webhooks/{webhookId}/deliveries/{deliveryId}/retry</code>	Reintentar manualmente una entrega puntual.

Scopes disponibles

Scope	Permite
<code>users:read</code>	Leer usuarios y roles por app
<code>users:write</code>	Crear, editar y borrar usuarios; asignar roles por app
<code>account:read</code>	Leer tus clientes OAuth, tokens y apps autorizadas
<code>account:write</code>	Gestionar tus clientes OAuth, tokens y apps autorizadas

No hay scope por defecto: cada app debe pedir explícitamente los que necesita al iniciar el flujo OAuth (ej. `scope=users:read users:write account:read account:write` para un admin completo).

Client credentials y recursos self-service: un token `client_credentials` no tiene usuario asociado, así que `/api/clients`, `/api/personal-access-tokens` y `/api/authorized-apps` le devuelven `422` (no hay "cuenta propia" que gestionar). Sí puede usar `/api/users` y `/api/apps` como servicio de confianza.

12. Webhooks: enterate sin hacer polling

Cada cliente OAuth puede registrar uno o más endpoints webhook desde `/clients` → botón "Webhooks" (o por API, sección 11). Cuando pasa algo relevante para ese cliente, este servidor le manda un `POST` con el evento — no hace falta consultar la API cada tanto para enterarse.

Eventos disponibles

Evento	Cuándo se dispara	A quién le llega
<code>token.revoked</code>	Se revoca un token de ese cliente — por el usuario dueño (self-service), por un admin (Sesiones activas, sección 4), o en cascada al revocar el cliente entero.	Los webhooks de ese cliente.
<code>user.deactivated</code>	Un admin desactiva un usuario.	Los webhooks de todo cliente que tenga al menos un token vigente para ese usuario en ese momento.

Con `token.revoked` tu app se entera al instante de que ya no tiene acceso a la cuenta de ese usuario (por ejemplo, para cerrarle la sesión local), en vez de esperar a que el token expire solo.

URLs permitidas

Protección anti-SSRF: la URL de un webhook tiene que resolver a una dirección pública. Se rechazan `localhost`, IPs loopback (`127.0.0.0/8`), privadas (`10.0.0.0/8` , `172.16.0.0/12` , `192.168.0.0/16`) y de link-local (`169.254.0.0/16` — incluye el endpoint de metadata de instancia en AWS/GCP/Azure, `169.254.169.254`). También esquemas que no sean `http` / `https` . Sin esto, cualquiera que registre un client podría hacer que este servidor le pegue a servicios internos (Redis, la propia API, metadata de la nube) y leer la respuesta en el panel — el clásico ataque SSRF. La validación se repite justo antes de cada envío, no solo al crear el webhook: un dominio público hoy podría resolver a una IP interna mañana (DNS rebinding).

Verificar la firma

Cada entrega incluye el header `X-Webhook-Signature: sha256={firma}` , un HMAC-SHA256 del body crudo (JSON) usando el `secret` que se te mostró al crear el webhook (una sola vez, igual que un `client_secret` — si lo perdiste, borra el webhook y crea uno nuevo). Verificalo antes de confiar en el payload:

```
// Node.js
import crypto from 'crypto';

function isValid(rawBody: string, signatureHeader: string, secret: string): boolean {
  const expected = 'sha256=' + crypto.createHmac('sha256', secret).update(rawBody).digest('hex')
  return crypto.timingSafeEqual(Buffer.from(signatureHeader), Buffer.from(expected));
}
```

El body tiene forma `{"event": "token.revoked", "data": {...}}`. El header `X-Webhook-Event` repite el nombre del evento, por si preferís ramificar por header sin parsear el body.

Reintentos

Si tu endpoint no responde `2xx` (o no responde en 5 segundos), se reintentará hasta 2 veces más con backoff creciente (10s, 60s, 300s). Después de 3 intentos fallidos, se abandona esa entrega — no hay cola de reintento infinita. Cada intento (éxito o fracaso) queda registrado; el panel muestra el estado del último envío junto a cada webhook (código de respuesta o "pendiente").

Pausar en vez de borrar: si tu endpoint va a estar caído un rato (deploy, mantenimiento), pausalo desde el panel en vez de dejar que fallen los reintentos — un webhook pausado no genera intentos de entrega hasta que lo reactivés.

Historial de entregas y reintento manual

El botón "**Ver historial**" de cada webhook (junto a Pausar/Borrar) abre el listado paginado de todas las entregas — no solo la última. Cada fila muestra el código de respuesta (o "pendiente" si nunca respondió), el evento, cuándo se creó y cuántos intentos acumula. Las entregas que no llegaron a un `2xx` tienen un botón "**Reintentar**": vuelve a encolar el mismo registro (no crea uno nuevo — el contador de intentos sigue acumulando sobre esa entrega) para forzar un envío inmediato sin esperar al backoff automático. Útil después de arreglar algo en tu endpoint y no querer esperar los 5 minutos del backoff.

Cada alta y baja de un webhook queda en el audit log (sección 7): `client.webhook_created` / `client.webhook_deleted`.

13. Documentación interactiva (Swagger)

Disponible en <http://localhost:8000/api/documentation> . Incluye el botón **Authorize** para probar los tres flujos (Authorization Code, Client Credentials, Password) directamente desde el navegador, usando el cliente de ejemplo "Swagger UI" sembrado (sección 3).

El JSON crudo de la especificación OpenAPI está en </docs> .

14. Referencia rápida de rutas

Interfaz web (sesión, uso interno de esta app)

Ruta	Qué hace
POST /web-api/register	Registro — crea la cuenta pendiente, no loguea.
POST /web-api/login / /logout	Autenticación por sesión. Si el usuario tiene 2FA, /login devuelve {"two_factor":true} en vez de loguear.
POST /web-api/login/2fa	Segundo paso del login con 2FA activo (código TOTP o de recuperación).
POST /web-api/forgot-password	Pedir el enlace de restablecimiento (respuesta genérica, exista o no la cuenta).
POST /web-api/reset-password	Confirmar el restablecimiento con el token del email — revoca todas las sesiones activas.
GET /web-api/me	Usuario logueado (con roles/permisos y two_factor_enabled).
POST /web-api/2fa/setup	Genera secreto pendiente + QR para activar 2FA.
POST /web-api/2fa/confirm	Confirma la activación con un código real — devuelve los códigos de recuperación una vez.
POST /web-api/2fa/disable	Desactiva 2FA (requiere contraseña actual).
POST /web-api/2fa/recovery-codes	Regenera los códigos de recuperación (requiere contraseña actual).
GET/POST/PUT/DELETE /web-api/clients	Clientes OAuth propios.
POST /web-api/clients/{id}/rotate-secret	Rotar el client_secret de un cliente confidencial propio (secret anterior válido 48h más).
GET/POST/DELETE /web-api/personal-access-tokens	Personal Access Tokens propios.

GET/DELETE /web-api/authorized-apps	Apps de terceros autorizadas.
GET/POST/PUT/DELETE /web-api/clients/{id}/webhooks	Webhooks de un cliente propio.
GET /web-api/clients/{id}/webhooks/{webhookId}/deliveries	Historial paginado de entregas de un webhook propio.
POST /web-api/clients/{id}/webhooks/{webhookId}/deliveries/{deliveryId}/retry	Reintentar manualmente una entrega.
GET/POST/PUT/DELETE /web-api/admin/users	Administración de usuarios (admin).
GET /web-api/admin/users/trashed	Usuarios en la papelera (admin).
POST /web-api/admin/users/{id}/restore	Restaurar un usuario borrado (admin).
DELETE /web-api/admin/users/{id}/force	Borrado definitivo (admin).
GET /web-api/admin/users/pending	Registros pendientes de aprobación (admin).
POST /web-api/admin/users/{id}/approve	Aprobar un registro pendiente (admin).
GET /web-api/admin/apps	Listado de todas las apps registradas (admin).
GET/POST/DELETE /web-api/admin/users/{id}/app-roles	Roles por app de un usuario (admin).
GET /web-api/admin/users/{id}/sessions	Tokens OAuth activos de un usuario (admin).
DELETE /web-api/admin/users/{id}/sessions/{tokenId}	Revocar una sesión puntual (admin).
DELETE /web-api/admin/users/{id}/sessions	Revocar todas las sesiones activas de un usuario (admin).
GET /web-api/admin/analytics/summary	Sesiones, logins, funnel de autorización, tokens activos, uso de scopes y actividad (admin).
GET /web-api/admin/audit-log	Historial de auditoría paginado y filtrable (admin).
POST /web-api/oauth/authorize-events	Registro interno del funnel de autorización (llamado por la

	pantalla de consentimiento, no pensado para uso directo).
POST /web-api/locale	Cambiar idioma (es/en/ca).

OAuth2 (Passport)

Ruta	Qué hace
GET /oauth/authorize	Inicio del flujo Authorization Code (pantalla de consentimiento).
POST /oauth/token	Emisión/renovación de tokens (todos los grant types).

Rate limiting: todas las rutas `/oauth/*` admiten 20 requests por minuto por combinación de IP + `client_id` — protege especialmente al Password Grant (usuario/contraseña directos contra `/oauth/token`) de fuerza bruta, sin penalizar a otros clientes si uno solo está siendo atacado. Va además del `throttle` genérico (60/min por IP) que Passport ya trae por defecto en `/oauth/token`.

API REST (Bearer token, para apps externas — Angular, Flutter, lo que sea)

Ruta	Qué hace
GET /api/user	Whoami.
GET/POST/PUT/DELETE /api/users	Gestión de usuarios (DELETE = papelera).
GET /api/users/trashed	Usuarios en la papelera.
POST /api/users/{id}/restore	Restaurar un usuario borrado.
DELETE /api/users/{id}/force	Borrado definitivo.
GET /api/users/pending	Registros pendientes de aprobación.
POST /api/users/{id}/approve	Aprobar un registro pendiente.
GET/POST/DELETE /api/users/{id}/app-roles	Roles por app de un usuario.
GET/DELETE /api/users/{id}/sessions	Tokens OAuth activos de un usuario; revocar uno o todos.
GET /api/apps	Listado de apps registradas.
GET /api/analytics/summary	Sesiones, logins, funnel de autorización, tokens activos, uso de scopes y actividad.
GET /api/audit-log	Historial de auditoría paginado y filtrable.
GET/POST/PUT/DELETE /api/clients	Tus clientes OAuth.
POST /api/clients/{id}/rotate-secret	Rotar el <code>client_secret</code> de un cliente confidencial (secret anterior válido 48h más).
GET/POST/DELETE /api/personal-access-tokens	Tus Personal Access Tokens.
GET/DELETE /api/authorized-apps	Apps de terceros que autorizaste.
GET/POST/PUT/DELETE /api/clients/{id}/webhooks	Webhooks de uno de tus clientes.
GET /api/clients/{id}/webhooks/{webhookId}/deliveries	Historial paginado de entregas de un webhook.
POST /api/clients/{id}/webhooks/{webhookId}/deliveries/{deliveryId}/retry	Reintentar manualmente una entrega.

