VERSION 0.4.0 · PHASE 4 — AUTO-ENQUEUE + 20 CHANNELS

EASYSOCIALAUTOPILOT

– BEDIENHANDBUCH –

20 Channels verbinden, Posts automatisch verteilen, Queue verstehen

Stand: 30.05.2026

VERSION 0.4.0 · PHASE 4 — AUTO-ENQUEUE + 20 CHANNELS

EASYSOCIALAUTOPILOT · BEDIENHANDBUCH

20 Channels verbinden, Posts automatisch verteilen, Queue verstehen

UWE HILTMANN

Internet-Unternehmensberater & KI-Consultant

ORION Consulting Solutions Administration Ltd.
Cardiff · Wiesbaden
Version 1.0 · 30.05.2026

IMPRESSUM

HERAUSGEBER & VERLAG

ORION Consulting Solutions Administration Ltd.
International House, 10 Churchill Way
CF10 2HE Cardiff
Wales, United Kingdom

KONTAKTBÜRO

Büro Wiesbaden
E-Mail: buero-wiesbaden@regioads24.de

VERTRETUNGSBERECHTIGT

Uwe Hiltmann · Internet-Unternehmensberater & KI-Consultant

HANDELSREGISTER

Handelsregister-Nr.: 11998547 · Registergericht: Cardiff

UMSATZSTEUER-IDENTIFIKATIONSNUMMER

USt-IdNr.: GB326020253

KONTAKT

Telefon: +49-160-98946921
Telefax: +49-6172-495008
E-Mail: uwe.hiltmann@uhdigital.net
Web: www.orion-csa.de · regioads24.com

VERANTWORTLICH FÜR DEN INHALT NACH § 18 ABS. 2 MSTV

ORION Consulting Solutions Administration Ltd.
International House, 10 Churchill Way
CF10 2HE Cardiff, Wales, United Kingdom

HAFTUNGSHINWEIS

Trotz sorgfältiger inhaltlicher Kontrolle wird keine Haftung für externe Links übernommen. Für den Inhalt verlinkter Seiten sind ausschließlich deren Betreiber verantwortlich. Rechtswidrige Inhalte werden nach Kenntnisnahme umgehend entfernt.

URHEBERRECHT

Die Inhalte dieses Handbuchs unterliegen dem deutschen Urheberrecht. Vervielfältigung, Bearbeitung, Verbreitung und jede Art der Verwertung außerhalb der Grenzen des Urheberrechts bedürfen der schriftlichen Zustimmung des Herausgebers.

VERBRAUCHERSTREITBEILEGUNG

Der Herausgeber ist weder bereit noch verpflichtet, an einem Streitbeilegungsverfahren vor einer Verbraucherschlichtungsstelle teilzunehmen.

MARKEN

„ContentForge Studio" ist eine Marke der ORION Consulting Solutions Administration Ltd. Genannte Drittmarken (z. B. YouTube, LinkedIn, Anthropic Claude, Resend) sind Eigentum ihrer jeweiligen Inhaber.

INHALTSVERZEICHNIS

EasySocialAutopilot · Bedienhandbuch

- Worum geht's?
- Voraussetzungen
- Installation
- Erster Login & Tour der UI
- Channels verbinden — der gemeinsame Flow
- Channels im Detail
- App-Registrierung & API-Referenz pro Channel
- Einen Post veröffentlichen (Phase-4-Flow)
- Queue verstehen
- Mehrere Channels gleichzeitig
- History
- Logs & Debugging
- Sicherheit
- Roadmap (Phase 5+)
- Technische Referenz
- Glossar
- Über dieses Plugin

EasySocialAutopilot · Bedienhandbuch

Dieses Handbuch begleitet dich durch Installation, Channel-Anbindung und Betrieb des Plugins **EasySocialAutopilot v0.4.0** in der Ausbaustufe **Phase 4 — Auto-Enqueue + 20 Channels**. Es beschreibt den realen Funktionsumfang dieser Version — ohne Versprechungen, die der Code noch nicht einlöst.

Worum geht's?

Was EasySocialAutopilot heute tut, warum es existiert und für wen sich diese Phase-4-Version lohnt.

EasySocialAutopilot ist ein WordPress-Plugin, das deine Blog-Beiträge automatisiert an Social-Media-Plattformen verteilt. Die Architektur ist von Anfang an als **Multi-Channel-Pipeline** gebaut: jeder Kanal (Mastodon, Bluesky, Telegram, Discord, LinkedIn, Reddit, Pinterest, …) ist ein eigener Driver hinter einem stabilen `ChannelDriver`-Interface. Du verbindest beliebig viele Accounts pro Kanal, deine WordPress-Posts wandern in eine Queue, ein Cron-Worker greift sich pending Jobs alle fünf Minuten und ruft den jeweiligen Driver auf.

In der vorliegenden Version 0.4.0 sind 20 Channel-Driver registriert. Elf davon sind voll funktional und posten in Produktion (Mastodon, Bluesky, Telegram, Discord, Webhook, LinkedIn, Reddit, Pinterest, Tumblr, WordPress.com, Google Business Profile, Blogger). Fünf weitere haben einen funktionierenden OAuth-Connect, aber der Posting-Pfad ist bis zu einer externen Freigabe (App-Review oder Paid-Tier) blockiert — Facebook Pages, Instagram, Threads, TikTok, X (Twitter). YouTube läuft im `description_update`-Modus produktiv, `video_upload` und Community Posts geben eine ehrliche Failure zurück. Medium läuft per Integration-Token (kein OAuth-Onboarding mehr möglich). XING ist ein dokumentierter Platzhalter — die öffentliche API wurde 2022 abgeschaltet.

Phase 4 ist live: Der Hook auf `transition_post_status` legt beim Publish eines Posts automatisch Queue-Jobs für alle aktiven Channel-Accounts an. Per Post-Meta `_esa_selected_accounts` kannst du die Zielauswahl pro Post überschreiben, mit `_esa_skip_distribution=1` ein einzelner Post komplett aus der Verteilung herausgenommen werden. Idempotenz ist eingebaut: `QueueRepository::hasActiveJob` verhindert Duplikate, wenn ein Post mehrfach gespeichert wird.

Die zentrale Designentscheidung bleibt: **kein Phone-Home, kein Lizenz-Server, keine SaaS-Abhängigkeit.** Tokens liegen AES-256-GCM-verschlüsselt in deiner WordPress-Datenbank. Wenn du das Plugin morgen ausschaltest, ist alles auf deinem Server — nichts auf einem fremden.

Zielgruppe der aktuellen Version sind **Plugin-Entwickler, Self-Hoster, Agenturen und Power-User**, die mit einer ehrlich beschriebenen Phase-4-Reife produktiv testen oder produktiv betreiben wollen. End-User mit Komfortanspruch sollten auf Phase 5 (Gutenberg-Sidebar-UI für die Per-Post-Selection) warten.

Voraussetzungen

Welche Plattform, welche WordPress-Konfiguration und welche externen Accounts du brauchst, bevor du startest.

ANFORDERUNG	MINDESTVERSION	HINWEIS
WordPress	6.0	Niedrigere Versionen werden weder geprüft noch supportet.
PHP	8.0	Nutzt `declare(strict_types=1)`, readonly-Properties, Named Arguments.
OpenSSL	beliebig moderne Version	Erforderlich für `aes-256-gcm` (Token-Verschlüsselung).
MySQL/MariaDB	dbDelta-kompatibel	Standard-WordPress-Schema, keine Sonderwünsche.
Ausgehendes HTTPS	–	Server muss die gewünschten Channel-Endpoints (Port 443) erreichen. In 99 % aller WordPress-Hosts gegeben.
WP-Cron oder System-Cron	aktiv	Ohne Cron läuft die Queue nicht. Bei deaktiviertem WP-Cron (`DISABLE_WP_CRON`) musst du `wp-cron.php` selbst anstoßen.
WP-Salts gesetzt	–	`AUTH_KEY`, `SECURE_AUTH_SALT`, `NONCE_KEY` aus `wp-config.php` werden zur Key-Derivation für die Token-Verschlüsselung genutzt.

Channel-spezifische Voraussetzungen

Für die meisten Channels brauchst du auf der Plattform-Seite mindestens ein normales Nutzerkonto. Manche Channels verlangen zusätzlich, dass du als **Plugin-Betreiber** eine eigene Developer-App registrierst (OAuth-Channels) — Channel-spezifische Setup-Schritte stehen in **Kapitel 5b**. Eine grobe Vorab-Übersicht:

ANFORDERUNG	CHANNELS
Nur Account auf der Plattform	Mastodon, Bluesky
Bot-/Webhook-/App-Token (kein Developer-Account)	Telegram, Discord, Webhook, Medium
Eigene OAuth-App registrieren	LinkedIn, Reddit, Pinterest, Tumblr, WordPress.com
Google-Cloud-Projekt + OAuth-Consent-Screen	Google Business Profile, YouTube, Blogger
Meta-Developer-App + App Review	Facebook Pages, Instagram, Threads
Plattform-Developer-Subscription	TikTok, X (Twitter)
Nicht verfügbar	XING (API 2022 abgeschaltet)

Installation

Drei Wege zur Installation — wähle den, der zu deiner Umgebung passt.

Variante A: ZIP-Upload (Endnutzer)

Empfohlen, wenn du ein klassisches Managed-WordPress-Hosting nutzt.

1. Lade das Release-ZIP von GitHub (Releases-Tab) oder von einer Build-Pipeline.
2. WordPress-Admin → **Plugins → Hochladen** → ZIP wählen → **Jetzt installieren**.
3. Nach dem Upload **Plugin aktivieren** klicken.

Variante B: Git-Clone (Entwickler)

```
cd wp-content/plugins
git clone https://github.com/uh8888/easysocialautopilot.git
cd easysocialautopilot
composer install --no-dev --optimize-autoloader
```

Anschließend im WP-Admin aktivieren.

Variante C: Bind-Mount im Docker-Setup (Self-Hoster)

Wenn dein WordPress in einem Docker-Container läuft, mountest du das Plugin-Verzeichnis direkt ins `wp-content/plugins`-Volume. Beispiel-Snippet für `docker-compose.yml`:

```
services:
  wordpress:
    image: wordpress:latest
    volumes:
      - ./easysocialautopilot:/var/www/html/wp-content/plugins/easysocialautopil
```

Vorteil: Code-Updates sind ein `git pull` weit weg, ohne ZIP-Roundtrip.

Was bei der Aktivierung passiert

Der Activator (`EasySocialAutopilot\Core\Activator::activate`) führt folgende Schritte aus:

SCHRITT	AKTION
1	Registriert einen Custom-Cron-Schedule `esa_five_minutes` (Intervall 300 s).
2	Führt `Migrations::run()` aus — legt fünf DB-Tabellen via `dbDelta` an.
3	Plant den Cron-Hook `esa_process_queue` auf 60 Sekunden in der Zukunft.
4	Schreibt `esa_db_version` in die `wp_options`-Tabelle.

Die fünf Tabellen (Prefix `wp_esa_` bei Standard-WP-Prefix):

TABELLE	INHALT
`wp_esa_channels`	Per-Channel-Flags (`is_enabled`) und globale Settings.
`wp_esa_channel_accounts`	Verbundene Accounts inkl. AES-verschlüsseltem Credential-Blob.
`wp_esa_queue`	Pending/Processing-Jobs mit Backoff-State.
`wp_esa_jobs`	Job-History (success + terminal failure).
`wp_esa_logs`	Application-Log-Stream der UI.

Erster Login & Tour der UI

Was du nach der Aktivierung im Admin siehst — und welche Sidebar-Items in Phase 4 wirklich gefüllt sind.

Nach der Aktivierung erscheint im linken WordPress-Sidebar-Menü der Eintrag **Social Autopilot**. Klick darauf — du landest auf einer einseitigen React-SPA mit Hash-Router (`#/dashboard` , `#/channels` , …) im klassischen ContentForge-Schwarz mit Lime-Akzenten.

Die SPA hat sechs Pages, die du oben über die Sub-Navigation erreichst:

PAGE	HASH-ROUTE	STATUS IN PHASE 4	WAS DU DORT SIEHST
Dashboard	`#/`	Funktionsfähig (`/dashboard/stats` -Endpoint neu)	Vier StatTiles (Channels Connected, Pending Jobs, Sent Today, Failed Today) plus eine Tabelle der letzten 8 Jobs.
Channels	`#/channels`	Voll funktionsfähig	Karten-Grid aller registrierten Channel-Driver — in Phase 4: 20 Karten. Connect-Button, verbundene Accounts mit Avatar + Status-Pill, Disconnect-Button.
Queue	`#/queue`	Funktionsfähig (lesend + Retry/Cancel + **Cron-Trigger**)	Tabelle aller pending/processing-Jobs. Spalten: `#` , Channel, Post, Status-Pill, Scheduled, Attempts, Aktionen Retry/Cancel. Plus Button **Cron jetzt ausführen**.
History	`#/history`	Funktionsfähig	Tabelle aller terminierten Jobs (done + failed) mit Link auf den Remote-Post (`remote_post_url`).
Logs	`#/logs`	Funktionsfähig	Application-Log-Stream aus `wp_esa_logs` mit Level-Filter (All / Debug / Info / Warn / Error).
Settings	`#/settings`	**Placeholder**	Zeigt einen EmptyState. Per-Channel-Settings werden via Constants in `wp-config.php` gesetzt — siehe Kapitel 5b.

Die SPA lädt ihre Daten ausschließlich über den REST-Namespace `esa/v1`. Jeder Request trägt einen `X-WP-Nonce`-Header, der von der `RestApi::authorize`-Permission-Callback gegen die `wp_rest`-Action verifiziert wird. Capability-Check: `manage_options` — also nur Administratoren.

Bundle-Loading-Hinweis

Das React-Bundle wird als ESM-Modul geladen, um die WordPress-`wp`-Global-Kollision zu umgehen. Vite ist mit `base: './'` gebaut — so funktionieren auch relative Font-Pfade in Subdirectory-Installationen. Wenn du dein eigenes Bundle baust: `npm run build` im Plugin-Root.

Channels verbinden — der gemeinsame Flow

Das Hauptkapitel: Schritt-für-Schritt durch den Connect-Flow, den die meisten Channels gemeinsam haben. Channel-spezifische Setup-Schritte stehen direkt danach in Kapitel 5b.

Was technisch passiert (Kurzfassung)

Der Connect-Flow folgt einem stabilen Muster, egal ob OAuth, App-Password oder Token:

PHASE	WAS PASSIERT	WO
1	UI sendet `POST /wp-json/esa/v1/channels/<id>/connect/start` mit dem passenden `context` (z. B. Instance-URL, Webhook-URL, Bot-Token, App-Password).	WP REST
2	Driver bereitet alles vor: registriert ggf. dynamisch eine App (Mastodon), baut die Authorize-URL (OAuth), validiert das Token sofort (App-Password / Webhook).	Driver-spezifisch
3	Bei OAuth: Browser geht zur Plattform, du klickst **Authorize**. Bei Token-Flows: Driver verifiziert das Credential synchron im selben Request.	Plattform-seitig
4	Bei OAuth-Callback: `admin-post.php?action=esa_oauth_callback&channel=<id>&code=…&state=…` tauscht den Code gegen den Access-Token.	`OAuthHandler`
5	Credentials werden mit AES-256-GCM verschlüsselt und in `wp_esa_channel_accounts` abgelegt. UI bekommt einen Toast und die neue Account-Karte erscheint.	DB

Schritt für Schritt im Browser

1. Öffne die Channels-Page. Sidebar → **Social Autopilot** → Tab **Channels**. Du siehst 20 Karten, jeweils mit Channel-Logo, Status (verbunden / nicht verbunden), Account-Counter und einem **Connect**-Button.

2. Klick auf Connect. Es öffnet sich ein modaler Dialog (`ConnectModal`) mit den Feldern, die der jeweilige Driver verlangt — von einer einzigen Instance-URL (Mastodon) bis zu drei Feldern (Bluesky: Handle, App-Password, optional PDS).

3. Fülle aus & sende ab. Bei OAuth-Channels wirst du auf die Plattform redirected. Bei Token-Channels macht der Driver eine Live-Verifizierung — wenn das Token nicht stimmt, siehst du sofort einen Fehler im Dialog.

4. Erfolg. Du landest auf `#/channels?connected=1&account_id=<neue-id>`, ein Success-Toast erscheint, die Account-Karte rendert sich mit Avatar, Anzeigename und **Aktiv**-Pill.

Mehrere Accounts pro Channel

Du kannst pro Channel beliebig viele Accounts verbinden — auch über mehrere Instanzen / Pages / Subreddits / Bot-Identitäten hinweg. Der Connect-Button heißt nach dem ersten verbundenen Account **Weiteren Account verbinden**. Jeder Account hat eine eigene `account_id`, einen eigenen Credential-Blob und kann einzeln getrennt werden.

Was passiert beim Trennen?

Trennen heißt: Row in `wp_esa_channel_accounts` wird gelöscht. **Externe App-Autorisierungen bleiben** in der Regel bestehen — sauber widerrufen kannst du sie auf der jeweiligen Plattform (Mastodon: *Authorized Apps*; Google: *Drittanbieter-Apps mit Zugriff*; Meta: *Business-Einstellungen → Verbundene Apps*; etc.). Pending Queue-Jobs, die auf den gelöschten Account verweisen, schlagen beim nächsten Cron-Run mit *"account not found"* terminal fehl.

Channels im Detail

Pro Channel: kurze Beschreibung, Setup-Schritte als Tabelle und Verweis auf die ausführliche README im Repo unter `src/Channels/<Name>/README.md`. Wo eine Plattform-seitige Freigabe fehlt, ist das ehrlich als "blockiert pending external approval" markiert.

5b.1 Simple Auth — kein Developer-Account nötig

Diese fünf Channels sind die schnellsten zum Verbinden: keine eigene App-Registrierung, keine Developer-Subscription, kein Verifizierungsprozess.

Mastodon

Voll funktional. Die OAuth-App wird beim ersten Connect dynamisch via `POST /api/v1/apps` auf der gewählten Mastodon-Instance registriert — du musst weder bei Mastodon noch beim Plugin-Hersteller etwas einrichten. Der Driver normalisiert Instance-Eingaben (Schema-Strip, Lowercase, Pfad weg), unterstützt Featured-Image-Upload und sendet `Idempotency-Key`-Header zur Doppel-Post-Vermeidung.

SCHRITT	AKTION
1	Mastodon-Account auf einer Instance deiner Wahl haben (`mastodon.social`, `chaos.social`, `troet.cafe`, eigener Server, ...).
2	In ESA: Channels → **Connect Mastodon** → Instance eintragen (z. B. `mastodon.social`).
3	Auf der Authorize-Seite **Authorize** klicken. Fertig.

Per-Post-Optionen: `visibility` (`public` / `unlisted` / `private` / `direct`), `language`, `sensitive`, `spoiler_text`.

Bluesky

Voll funktional. Authentifiziert mit einem **App-Password** (nicht dem Account-Passwort). Der Driver speichert `accessJwt` (~1 h) und `refreshJwt` (~2 Monate), rotiert den Access-Token automatisch, und fällt bei Doppel-Expire auf erneutes `createSession` zurück. Link-Facets werden automatisch erkannt, Featured Image als `app.bsky.embed.images` mit max. 1 MB eingebettet.

SCHRITT	AKTION
1	https://bsky.app → *Settings* → *Privacy and security* → *App passwords* → neues App-Password anlegen.
2	In ESA Connect-Dialog: `identifier` (Handle wie `alice.bsky.social`), `app_password`, optional `pds_url` (Default `https://bsky.social`).

Per-Post-Optionen: `langs` (Default `en`). Hard-Limit: 300 Zeichen. Details: `src/Channels/Bluesky/README.md`.

Telegram

Voll funktional. Nutzt einen Bot-Token statt OAuth. Mit Featured Image wird `sendPhoto` mit Caption gesendet (1024-Zeichen-Limit), ohne wird `sendMessage` mit HTML-formatiertem Body gesendet (4096-Zeichen-Limit).

SCHRITT	AKTION
1	In Telegram `@BotFather` anschreiben, `/newbot` senden, Token im Format `1234567890:ABCdef...` kopieren.
2	Ziel-Chat-ID besorgen — bei Personal Chat: `@userinfobot` nach numerischer ID fragen. Bei Channel: Bot als Admin mit "Post Messages"-Recht hinzufügen, `@channel_username` nutzen. Bei Group: Bot adden, `getUpdates`-Call gegen die API für die Chat-ID.
3	In ESA Connect-Dialog: `bot_token` und `default_chat_id` eintragen.

Per-Post-Optionen: `chat_id` (Override), `parse_mode` (`HTML` / `MarkdownV2`), `disable_link_preview`. Details: `src/Channels/Telegram/README.md`.

Discord

Voll funktional. Postet via Channel-Webhook — kein Bot-Account, kein OAuth-Scope. Sendet pro Post einen Rich-Embed mit Title, Description, URL, Featured-Image und Discord-Blurple als Farbe.

SCHRITT	AKTION
1	Discord → *Server Settings* → *Integrations* → *Webhooks* → *New Webhook* → Channel wählen, Name/Avatar setzen, **Copy Webhook URL** klicken.
2	In ESA Connect-Dialog: `url` einfügen. Optional: `username` / `avatar_url` Override.

Per-Post-Optionen: `content` (Message über dem Embed, max. 2000 Zeichen). Details: `src/Channels/Discord/README.md`.

Webhook

Voll funktional. Generischer JSON-POST an einen beliebigen HTTPS-Endpoint — perfekt für Zapier, n8n, Make, Custom-CRMs, Log-Aggregatoren. Optional mit HMAC-SHA256-Signatur per Shared-Secret.

SCHRITT	AKTION
1	Empfangs-URL im Ziel-System einrichten.
2	In ESA Connect-Dialog: `url`, optional `secret` (HMAC), optional `headers` (z. B. `X-API-Key`).

Payload-Schema enthält `id`, `title`, `content`, `excerpt`, `link`, `featured_image`, `published_at`, `author`. Bei gesetztem Secret kommt ein `X-ESA-Signature: sha256=<hex>` Header mit. Details: `src/Channels/Webhook/README.md`.

5b.2 OAuth 2.0 Standard — User registriert eigene App

Sechs Channels mit klassischem OAuth-2-Authorization-Code-Flow. Du brauchst pro Channel **eine eigene Developer-App** beim Provider und musst Client-ID/Secret als WP-Konstanten setzen.

LinkedIn

Voll funktional. Postet auf Member-Feeds und optional auf Company Pages via REST Posts API (`/rest/posts`, Version `202405`).

SCHRITT	AKTION
1	App anlegen auf https://www.linkedin.com/developers/apps . Redirect-URL: `https://YOUR-SITE/wp-admin/admin.php?page=esa-channels&esa_oauth=linkedin`
2	Products anfordern: **Sign In with LinkedIn using OpenID Connect**, **Share on LinkedIn**, optional **Community Management API** (für Pages).
3	In `wp-config.php` : `ESA_LINKEDIN_APP_ID` + `ESA_LINKEDIN_APP_SECRET` .
4	In ESA Channels → **Connect LinkedIn** → OAuth-Consent zustimmen.
5	Für Company Pages: erneut verbinden mit Toggle **Also connect Company Pages** — Pages mit ADMINISTRATOR/CONTENT_ADMIN-Rolle werden gelistet.

Scopes: `openid profile w_member_social` (Member) bzw. zusätzlich `w_organization_social r_organization_social rw_organization_admin` (Pages). Per-Post: `visibility` (`PUBLIC` / `CONNECTIONS`), `reshare_disabled` . Details: `src/Channels/LinkedIn/README.md` .

Reddit

Voll funktional. Submitted Link- oder Self-Posts via `POST /api/submit` auf `oauth.reddit.com` mit `duration=permanent` (Refresh-Token).

SCHRITT	AKTION
1	https://www.reddit.com/prefs/apps → **create another app…** → Type **web app**. Redirect-URI wie bei LinkedIn-Muster.
2	Client-ID (kurzer String unter App-Name) + Secret kopieren.
3	In `wp-config.php` : `ESA_REDDIT_APP_ID` + `ESA_REDDIT_APP_SECRET` .
4	In ESA Connect → Reddit-Account-Consent.

Scopes: `identity submit flair mysubreddits read` . Per-Post: `subreddit` (Pflicht, ohne `r/` -Prefix), `kind` (`link` / `self`), `text` , `flair_id` , `nsfw` , `spoiler` . Reddit verlangt einen unique User-Agent — der Driver sendet `EasySocialAutopilot/<version> by <reddit-username>` . Details: `src/Channels/Reddit/README.md` .

Pinterest

Voll funktional. Erzeugt Pins via Pinterest API v5 (`POST /v5/pins`). Pinterest **erfordert ein Bild** pro Pin — Posts ohne Featured Image failen mit klarer Fehlermeldung.

SCHRITT	AKTION
1	https://developers.pinterest.com/ → **My apps** → **Create app**. App-Type: Standard-Tier (Production).
2	Redirect-URI eintragen, App-ID + App-Secret kopieren.
3	In `wp-config.php` : `ESA_PINTEREST_APP_ID` + `ESA_PINTEREST_APP_SECRET` .
4	In ESA Connect → Pinterest-Consent. Beim ersten Connect wird das erste Board als Default gecached; in Per-Account-Settings änderbar.

Scopes: `boards:read,pins:read,pins:write,user_accounts:read` . Per-Post: `board_id` , `section_id` , `alt_text` . Access-Token ~30 Tage, Refresh ~60 Tage (Driver refreshed automatisch). Details: `src/Channels/Pinterest/README.md` .

Tumblr

Voll funktional. Nutzt **OAuth 1.0a HMAC-SHA1** (historisch bedingt). Postet als Tumblr-Text-Post auf den Primary-Blog.

SCHRITT	AKTION
1	App registrieren unter https://www.tumblr.com/oauth/apps . Callback-URL: ESA-OAuth-Callback.
2	In `wp-config.php` : `ESA_TUMBLR_CONSUMER_KEY` + `ESA_TUMBLR_CONSUMER_SECRET` .
3	In ESA Connect → Tumblr-Consent → Primary-Blog wird per `/v2/user/info` ermittelt.

Per-Post: `state` (`published` / `draft` / `queue` / `private`), `format` (`html` / `markdown`), `tags` . Driver nutzt die stabile Legacy-form-encoded-Variante, nicht NPF. Details: `src/Channels/Tumblr/README.md` .

WordPress.com

Voll funktional. Postet auf WordPress.com-Blogs oder Jetpack-verbundene Self-Hosted-Sites via REST API v1.1.

SCHRITT	AKTION
1	App registrieren unter https://developer.wordpress.com/apps/ .
2	In `wp-config.php` : `ESA_WPCOM_CLIENT_ID` + `ESA_WPCOM_CLIENT_SECRET` .
3	In ESA Connect → WordPress.com-Consent. Multi-Site: Tokens resolven via `/me/sites` .

Per-Post: `status` , `tags` , `categories` , `sticky` . **Achtung:** Bearer-Tokens haben ~2 Wochen TTL, kein Refresh-Token — bei Expire `status='expired'` , User muss neu autorisieren. Details: `src/Channels/WordPressCom/README.md` .

Medium

Funktional via **Integration-Token-Pfad** (OAuth ist seit März 2023 für neue Partner geschlossen und das API-Repo archiviert). Wenn `POST /v1/users/{id}/posts` mal 404/410 zurückgibt, zeigt der Driver einen ehrlichen "Medium API is no longer publicly accessible"-Fehler.

SCHRITT	AKTION
1	https://medium.com/me/settings → **Security & apps** → **Integration tokens** → Token-Description (z. B. `EasySocialAutopilot`) → **Get integration token**.
2	Hex-Token kopieren. Wenn der Settings-Bereich fehlt: Account nicht eligible, Channel nicht nutzbar.
3	In ESA: **Connect Medium** → Token einfügen. Driver verifiziert per `GET /v1/me` .

Per-Post: `publish_status` (`public` / `draft` / `unlisted`), `tags` (max. 5), `canonical_url` (Default `true` , gut für SEO-Credit), `license` , `notify_followers` . Details: `src/Channels/Medium/README.md` .

5b.3 Google Ecosystem — ein Cloud-Projekt, drei Channels

Google Business Profile, YouTube und Blogger teilen sich **einen** Google-Cloud-Projekt und **einen** OAuth-Client. Du registrierst einmal, aktivierst dann pro Channel die passende API. Konstanten in `wp-config.php`:

```
define( 'ESA_GOOGLE_APP_ID',     '...apps.googleusercontent.com' );
define( 'ESA_GOOGLE_APP_SECRET', '...' );
```

Gemeinsames Google-Setup

SCHRITT	AKTION
1	https://console.cloud.google.com → Projekt anlegen oder bestehendes wählen.
2	**APIs & Services → Enabled APIs** → die gewünschten APIs aktivieren (Business Profile API + My Business Account Management + My Business Business Information für GBP; YouTube Data API v3 für YouTube; Blogger API v3 für Blogger).
3	**OAuth consent screen** → User-Type **External** → die nötigen Scopes hinzufügen (siehe Channel-Sektionen). Während der Verifizierung sich selbst als Test-User adden — Production-Posting erfordert Google-App-Verification (Wochen Wartezeit).
4	**Credentials → Create OAuth client ID** → Application type **Web application** → Authorized redirect URI: `https://YOUR-SITE/wp-admin/admin-post.php?action=esa_oauth_callback&channel=<channel-id>` (eine URI pro Channel-ID).
5	Client-ID + Secret in `wp-config.php` setzen.

Google Business Profile

Voll funktional. Postet in den Location-Feed (eine `ChannelAccount` pro Location).

Scope: `https://www.googleapis.com/auth/business.manage`. Per-Post: `language_code`, `cta_action_type` (`LEARN_MORE` / `BOOK` / `ORDER` / `SHOP` / `SIGN_UP` / `CALL`), `cta_url`. Limits: Summary max. 1500 Zeichen, Featured Image als `media[0].sourceUrl` (muss öffentlich erreichbar sein), Posts laufen nach 7 Tagen automatisch ab (GBP-Policy). Details: `src/Channels/GoogleBusinessProfile/README.md`.

YouTube

`description_update`-Modus voll funktional — perfekt um z. B. Show-Notes oder Affiliate-Links automatisiert aus einem WP-Post-Update an ein bestehendes Video anzuhängen. `video_upload` und `community_post` returnen ehrlich `SendResult::failure`.

Scope: `https://www.googleapis.com/auth/youtube`. Per-Post: `mode` (`description_update` Default), `video_id` (Pflicht), `prepend` (Bool — Default ist Append). Warum die anderen Modi blockiert sind: `video_upload` braucht den sensitiven `youtube.upload`-Scope + Google-Branding-Verification + Resumable-Multipart-Upload; `community_post` hat keine öffentliche API, nur YouTube Studio. Details: `src/Channels/YouTube/README.md`.

Blogger

Voll funktional. Postet einen WordPress-Post als neuen Blogger-v3-Post. Eine `ChannelAccount` pro Blogger-Blog — eine Google-Identität kann mehrere Blogs haben, jeden separat verbinden.

Scope: `https://www.googleapis.com/auth/blogger`. Per-Post: `labels` (Tags), `draft` (Bool), `title` (Override). `post_content` läuft durch `the_content`-Filter, Blogger nimmt HTML direkt an. Details: `src/Channels/Blogger/README.md`.

5b.4 Meta + Paid Tier — OAuth klappt, Posten blockiert

Fünf Channels mit funktionierendem OAuth-Connect, aber blockiertem Posting-Pfad. Du kannst **jetzt schon** die Accounts verbinden und in der UI sehen — sobald die externe Freigabe da ist, wird der `POSTING_BLOCKED`-Stub in `send()` entfernt und das Posten geht live.

Drei davon (Facebook Pages, Instagram, Threads) teilen sich **eine** Meta-Developer-App. Konstanten:

```
define( 'ESA_META_APP_ID',     '...' );
define( 'ESA_META_APP_SECRET', '...' );
```

Facebook Pages

Connect funktional. Posten BLOCKIERT pending Facebook App Review für `pages_manage_posts`.

Setup: Meta Developer Portal → App → Scopes `pages_show_list, pages_read_engagement, pages_manage_posts, business_management` anfordern → in Live-Mode + App Review schicken. OAuth: Short-Lived-User-Token → Long-Lived-User-Token (~60 d) → `GET /me/accounts` liefert pro Page einen **perpetual** Page-Access-Token. Pro Page ein `ChannelAccount`. Posting-Flow (vorbereitet, aber blockiert): `POST graph.facebook.com/v21.0/{page-id}/feed`. Details: `src/Channels/Facebook/README.md`.

Instagram

Connect funktional. Posten BLOCKIERT pending Instagram Business Account + Facebook App Review für `instagram_content_publish`.

Voraussetzung: User verwaltet mindestens eine FB-Page, die mit einem **Instagram-Business-Konto** verknüpft ist. Scopes: `pages_show_list, pages_read_engagement, business_management, instagram_basic, instagram_content_publish`. Discovery: `GET /me/accounts?fields=instagram_business_account`. Posting-Flow (geplant): 2-Step Create-Media → Publish-Media. Details: `src/Channels/Instagram/README.md`.

Threads

Connect funktional. Posten BLOCKIERT pending Threads-API-Access (derzeit Limited Rollout).

OAuth läuft über `threads.net/oauth/authorize` (nicht facebook.com), Token-Exchange auf `graph.threads.net`. Long-lived-Exchange `?grant_type=th_exchange_token` ergibt ~60-Tage-Token. Scopes: `threads_basic, threads_content_publish`. Posting-Flow analog Instagram (2-Step). Details: `src/Channels/Threads/README.md`.

TikTok

Connect funktional. Posten BLOCKIERT pending TikTok Developer Sandbox + Production Review für `video.publish`.

TikTok hat **keine Text-only Posts** — jeder Publish braucht ein Video. Der Driver fail-fast auf fehlendes `options['video_url']` (auch vor dem Posting-Block). Konstanten: `ESA_TIKTOK_CLIENT_KEY` + `ESA_TIKTOK_CLIENT_SECRET`. Scopes: `user.info.basic, video.publish`. Refresh-Token-Flow implementiert. Posting-Flow (geplant): Async 3-Step Init → PUT-Upload → Status-Poll. Details: `src/Channels/TikTok/README.md`.

X (Twitter)

Connect funktional. Posten BLOCKIERT pending X API v2 Basic-Tier ($100+/Monat).

OAuth 2.0 mit PKCE. Konstanten: `ESA_X_CLIENT_ID` (optional `ESA_X_CLIENT_SECRET` für Confidential-Client). Scopes: `tweet.read, tweet.write, users.read, offline.access`. PKCE: `code_verifier` (64 random Bytes, base64url), `code_challenge = SHA256(verifier)` (S256). Refresh-Token-Flow implementiert. Posting-Flow (geplant): `POST /2/tweets` mit max. 280 Zeichen Free / 25.000 Premium. Details: `src/Channels/X/README.md`.

5b.5 Discontinued — XING

XING

Platzhalter — nicht funktional. XING hat seine öffentliche Developer-API (`dev.xing.com`) 2022 abgeschaltet. Der OAuth 1.0a Flow und der `/v1/users/me/status_message`-Endpoint sind nicht mehr erreichbar, kein Ersatz wurde veröffentlicht.

Was der Driver tut: Er registriert den Channel in der Connector-Liste, alle anderen Methoden werfen einen klaren `RuntimeException` mit *"XING public API was discontinued in 2022; this channel is a placeholder pending an alternative integration path."* `refreshIfNeeded()` markiert Accounts als `status='expired'`.

Manueller Workaround: XING Business Manager für Company-Page-Scheduling nutzen, persönliche Status-Updates aus WordPress per Copy-Paste übertragen. Wenn die API zurückkommt: der Stub-Code in `src/Channels/Xing/` ist so gebaut, dass man die Real-Implementierung 1:1 austauschen kann (OAuth 1.0a kann von `\EasySocialAutopilot\Channels\Tumblr\TumblrOAuth1` recycelt werden). Details: `src/Channels/Xing/README.md`.

App-Registrierung & API-Referenz pro Channel

Dieses Kapitel ist die technische Tiefen-Sektion zu Kapitel 5b. Pro Channel bekommst du:

1. **Stammdaten** (Auth-Typ, Developer-Portal-URL, ob App-Review nötig, Kosten).
2. **Tabelle der API-Endpoints**, die der Driver tatsächlich aufruft (extrahiert aus den `*Driver.php`-Source-Files in `src/Channels/`).
3. **Schritt-für-Schritt-Anleitung** für die App-Registrierung beim Provider — mit den `wp-config.php`-Konstanten, die der Driver liest.
4. **Stolpersteine** — plattform-spezifische Quirks, die dich beim Connect oder Posting kalt erwischen können.

Callback-URL-Muster für alle OAuth-Channels:

```
https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel
```

Die Channel-ID ist der Wert, den `$driver→getId()` zurückgibt — alle 20 sind in der Tabelle:

CHANNEL	CHANNEL-ID	CHANNEL	CHANNEL-ID
Mastodon	`mastodon`	Medium	`medium`
Bluesky	`bluesky`	Google Business Profile	`google_business_profile`
Telegram	`telegram`	YouTube	`youtube`
Discord	`discord`	Blogger	`blogger`
Webhook	`webhook`	Facebook Pages	`facebook`
LinkedIn	`linkedin`	Instagram	`instagram`
Reddit	`reddit`	Threads	`threads`
Pinterest	`pinterest`	TikTok	`tiktok`
Tumblr	`tumblr`	X (Twitter)	`x`
WordPress.com	`wordpress_com`	XING	`xing`

Für LinkedIn weicht die Redirect-URL ab (`/wp-admin/admin.php?page=esa-channels&esa_oauth=linkedin`); alle anderen folgen dem Standard-Muster oben.

5c.1 Mastodon

Auth-Typ: OAuth 2.0 (Dynamic App Registration — du registrierst keine App) **Developer-Portal:** keiner — der Driver registriert pro Instance dynamisch via `POST /api/v1/apps` **App-Review nötig:** Nein **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
POST	`https://<instance>/api/v1/apps`	Client-Credentials pro Instance dynamisch erzeugen
GET	`https://<instance>/oauth/authorize`	Browser-Redirect zum User-Consent
POST	`https://<instance>/oauth/token`	Code → Access-Token
GET	`https://<instance>/api/v1/accounts/verify_credentials`	Identity-Check nach Consent
POST	`https://<instance>/api/v1/media`	Featured-Image-Upload
POST	`https://<instance>/api/v1/statuses`	Toot publishen (mit `Idempotency-Key` -Header)

App-Registrierung (Schritt für Schritt)

1. **Keine** manuelle App-Registrierung nötig. Du brauchst nur einen Mastodon-Account auf einer Instance deiner Wahl (`mastodon.social` , `chaos.social` , `troet.cafe` , eigener Server, …).
2. In ESA: Channels → **Connect Mastodon** → Instance eintragen (z. B. `mastodon.social`). Der Driver normalisiert die Eingabe (Schema-Strip, Lowercase, Pfad weg).
3. Der Driver feuert `POST /api/v1/apps` mit App-Name `EasySocialAutopilot` , scopes `read write` , redirect-URI = ESA-Callback. Die zurückgegebenen `client_id` / `client_secret` werden in einem 10-Minuten-Transient gespeichert (`esa_mastodon_oauth_<state>`).

4. Browser-Redirect auf `https://<instance>/oauth/authorize?...&scope=read+write`. Du klickst **Authorize**.
5. ESA tauscht den Code gegen einen Access-Token (kein Refresh-Token bei Mastodon — Tokens sind langlebig bis User-Revoke).

Keine `wp-config.php`-Konstanten nötig.

Stolpersteine

- **Pro Instance eine eigene App-Registrierung:** Wenn dieselbe ESA-Installation gegen 5 Mastodon-Instances connected, liegen 5 Client-Credential-Paare in der Mastodon-Datenbank. Bei `Account trennen` werden die App-Credentials in ESA gelöscht — die App bleibt aber Instance-seitig liegen (Mastodon hat keine `DELETE /api/v1/apps`).
- **Status-Längenlimit:** 500 Zeichen (Mastodon-Standard). Der Driver kürzt nicht — Posts >500 Zeichen failen mit Mastodon-422.
- **Idempotency:** ESA sendet `Idempotency-Key: <sha256(post_id + account_id)>`. Bei Mastodon-Retries innerhalb von 24 h kommt derselbe Status zurück, nicht ein Duplikat.

5c.2 Bluesky

Auth-Typ: App-Password (kein OAuth — AT Protocol nutzt `createSession`) **Developer-Portal:** keiner — Setup über User-Settings unter https://bsky.app/settings/app-passwords **App-Review nötig:** Nein **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
POST	`https://bsky.social/xrpc/com.atproto.server.createSession`	Login mit Handle + App-Password → `accessJwt` + `refreshJwt`
POST	`https://bsky.social/xrpc/com.atproto.server.refreshSession`	Access-JWT (~1 h) per Refresh-JWT (~2 Monate) erneuern
POST	`https://bsky.social/xrpc/com.atproto.repo.uploadBlob`	Featured Image hochladen (max. 1 MB)
POST	`https://bsky.social/xrpc/com.atproto.repo.createRecord`	Post als `app.bsky.feed.post`-Record erzeugen

(Wenn ein eigener PDS gesetzt ist, ersetzt dessen URL `https://bsky.social`.)

App-Registrierung (Schritt für Schritt)

1. Bei https://bsky.app einloggen.
2. **Settings → Privacy and security → App passwords → Add app password** → Name vergeben (z. B. `EasySocialAutopilot`) → **Create**.
3. Bluesky zeigt dir das App-Password genau einmal — sofort kopieren.
4. In ESA: **Connect Bluesky** → `identifier` (Handle wie `alice.bsky.social`), `app_password`, optional `pds_url` (Default `https://bsky.social`).
5. Driver feuert `createSession`, speichert `accessJwt` + `refreshJwt` AES-256-GCM-verschlüsselt.

Keine `wp-config.php`-Konstanten.

Stolpersteine

- **Hard-Limit 300 Zeichen** pro Post (AT-Protocol-Spec). Der Driver kürzt nicht.
- **Image-Limit 1 MB:** Größere Featured Images werden vom Driver nicht automatisch verkleinert — Post failt mit `BlobTooLarge`.
- **Token-Doppel-Expire:** Wenn auch der Refresh-JWT abläuft (~2 Monate Inaktivität), fällt der Driver auf ein neues `createSession` mit dem gespeicherten App-Password zurück. Wenn du das App-Password Bluesky-seitig revoked hast, schlägt das fehl und der Account geht auf `status='expired'`.
- **Link-Facets:** ESA detektiert URLs im Body und sendet sie als `facets` mit — sonst sind Links auf Bluesky nicht klickbar.

5c.3 Telegram

Auth-Typ: Bot-Token (kein OAuth) **Developer-Portal:** https://t.me/BotFather (im Telegram-Client) **App-Review nötig:** Nein **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://api.telegram.org/bot<TOKEN>/getMe`	Bot-Identity beim Connect verifizieren
POST	`https://api.telegram.org/bot<TOKEN>/sendMessage`	Text-Post (max. 4096 Zeichen, HTML-Mode)
POST	`https://api.telegram.org/bot<TOKEN>/sendPhoto`	Post mit Featured Image (max. 1024 Zeichen Caption)
GET	`https://api.telegram.org/bot<TOKEN>/getUpdates`	Optional zur Chat-ID-Ermittlung in Groups

App-Registrierung (Schritt für Schritt)

1. In Telegram (App oder Web-Client) **@BotFather** anschreiben.
2. `/newbot` senden → Display-Name vergeben → Username vergeben (muss auf `bot` enden, z. B. `meine_blog_bot`).
3. BotFather antwortet mit dem Token im Format `1234567890:ABCdef-...` (45-46 Zeichen). **Sofort kopieren.**
4. **Ziel-Chat-ID besorgen:**
5. **Personal Chat:** @userinfobot anschreiben, der gibt deine numerische User-ID zurück.
6. **Channel:** Bot als **Admin** mit *"Post Messages"*-Recht zum Channel hinzufügen. Public-Channel: `@channelusername` reicht. Private-Channel: Bot-Update abrufen (`getUpdates`) für die `-100…`-ID.
7. **Group:** Bot in die Group adden, eine Nachricht in der Group senden, dann `getUpdates` abrufen.
8. In ESA: **Connect Telegram** → `bot_token` + `default_chat_id` eintragen. Driver ruft `getMe` zur Verifikation.

Keine `wp-config.php`-Konstanten.

Stolpersteine

- **Bot-Privacy-Mode:** Standard-Bots in Groups sehen keine Messages → für `getUpdates` zur Chat-ID-Ermittlung muss in BotFather (`/setprivacy` → `Disable`) Privacy off geschaltet werden (oder den Bot Admin machen).

- **HTML-Parse-Mode:** Der Driver sendet `parse_mode=HTML`. Telegram akzeptiert nur einen kleinen Whitelist-Subset (`<b>`, `<i>`, `<a>`, `<code>`, …). Ungültige Tags → 400-Error. Per-Post override auf `MarkdownV2` möglich, hat aber strenge Escape-Anforderungen.
- **Channel-Bot-Permissions:** Wenn der Bot später aus dem Channel rausgeworfen wird, returnt die API `chat not found` — der Driver markiert den Account NICHT automatisch als expired (Bot-Token bleibt gültig für andere Chats).

5c.4 Discord

Auth-Typ: Webhook-URL (kein Developer-Account, kein OAuth, kein Bot) **Developer-Portal:** keiner — Setup direkt in Discord (Server-Settings) **App-Review nötig:** Nein **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://discord.com/api/webhooks/<id>/<token>`	Webhook-Meta abfragen beim Connect
POST	`https://discord.com/api/webhooks/<id>/<token>`	Post als Embed publishen

App-Registrierung (Schritt für Schritt)

1. In Discord den Ziel-Server öffnen → **Server Settings → Integrations → Webhooks → New Webhook**.
2. Channel wählen, Name + Avatar setzen.
3. **Copy Webhook URL** klicken (Format: `https://discord.com/api/webhooks/<numeric-id>/<long-token>`).
4. In ESA: **Connect Discord** → `url` einfügen. Optional: `username` / `avatar_url` Override pro Account.

Du brauchst **Manage Webhooks**-Recht im Discord-Server. Keine `wp-config.php`-Konstanten.

Stolpersteine

- **Webhook ist Channel-spezifisch:** Ein Webhook postet immer nur in genau einen Channel. Für mehrere Channels mehrere Webhooks anlegen, jeweils als eigenen ESA-Account hinzufügen.
- **Embed-Limits:** Title 256 Zeichen, Description 4096 Zeichen, optional `content` (Message über dem Embed) 2000 Zeichen.
- **Rate-Limit:** ~5 Posts/Sekunde pro Webhook. ESA's Batch-Größe von 25 Jobs/Cron-Run liegt weit drunter.
- **Webhook löschen = Account tot:** Wer in Discord den Webhook löscht, lässt den ESA-Account auf `404 Unknown Webhook` laufen. Der Driver markiert NICHT automatisch als expired — du musst in ESA selbst neu verbinden.

5c.5 Webhook

Auth-Typ: Shared-Secret (HMAC-SHA256, optional) — kein OAuth **Developer-Portal:** keiner — Empfangs-URL kommt aus deinem eigenen System **App-Review nötig:** Nein **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`<deine-empfangs-URL>`	Verifikations-Probe beim Connect (HEAD/GET, akzeptiert jeden 2xx)
POST	`<deine-empfangs-URL>`	JSON-Payload pro Post

App-Registrierung (Schritt für Schritt)

1. Im Ziel-System (Zapier, n8n, Make, eigenes CRM, Log-Aggregator) einen HTTPS-Endpoint anlegen, der POST mit `Content-Type: application/json` akzeptiert.
2. Optional: ein langes Random-Secret generieren (z. B. `openssl rand -hex 32`) und im Ziel-System hinterlegen — der Empfänger muss den `X-ESA-Signature: sha256=<hex>`-Header gegen `HMAC-SHA256(secret, body)` prüfen.
3. In ESA: **Connect Webhook** → `url`, optional `secret`, optional `headers` (z. B. `{"X-API-Key": "..."}`).

Keine `wp-config.php`-Konstanten.

Stolpersteine

- **Payload-Schema ist fix:** `{id, title, content, excerpt, link, featured_image, published_at, author}`. Wenn dein Ziel-System ein anderes Feld-Mapping erwartet, brauchst du einen Transformer dazwischen (n8n-Node, Zapier-Formatter).
- **HTTPS-Pflicht:** Der Driver weigert sich, gegen `http://` zu senden.
- **Timeout:** ESA wartet 30 Sekunden auf Response. Lange laufende Empfänger nicht synchron verarbeiten — sofort `200 OK` zurückgeben und async weiterverarbeiten.

5c.6 LinkedIn

Auth-Typ: OAuth 2.0 (3-legged authorization code) **Developer-Portal:** https://www.linkedin.com/developers/apps **App-Review nötig:** Nein für Personal Posts (Standard-Products); Ja für Company-Pages-Posts (Community Management API) **Kosten:** kostenlos für Standard-Tier

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://www.linkedin.com/oauth/v2/authorization`	Browser-Redirect zum User-Consent
POST	`https://www.linkedin.com/oauth/v2/accessToken`	Code → Access-Token (~60 d)
GET	`https://api.linkedin.com/v2/userinfo`	OIDC-Profil-Lookup für Identity
GET	`https://api.linkedin.com/rest/organizationAcls?...&role=ADMINISTRATOR`	Company-Pages des Users listen
POST	`https://api.linkedin.com/rest/posts`	Post publishen (API-Version-Header `LinkedIn-Version: 202405`)

App-Registrierung (Schritt für Schritt)

1. Geh zu https://www.linkedin.com/developers/apps → **Create app**.
2. Trage ein:

3. **App Name:** `EasySocialAutopilot` (oder dein eigener)
4. **LinkedIn Page:** dein Company-Page-Account (LinkedIn verlangt eine Page als Owner — Personal-Profile geht nicht)
5. **Privacy policy URL:** Pflicht
6. **App logo:** Pflicht
7. Tab **Auth** → **Authorized redirect URLs for your app** ergänzen: `https://<deine-domain>/wp-admin/admin.php?page=esa-channels&esa_oauth=linkedin` (LinkedIn-spezifisches Muster, weicht vom Standard ab).
8. Tab **Products** → folgende Products anfordern:
9. **Sign In with LinkedIn using OpenID Connect** (sofort verfügbar)
10. **Share on LinkedIn** (sofort verfügbar) — gibt scope `w_member_social`
11. Optional: **Community Management API** (App-Review nötig) — gibt scope `w_organization_social`
12. Notiere: **Client ID** und **Primary Client Secret** (Tab **Auth**).
13. In `wp-config.php` ergänzen:

```
php define( 'ESA_LINKEDIN_APP_ID', 'deine-client-id' ); define( 'ESA_LINKEDIN_APP_SECRET', 'dein-client-secret' );
```

1. In ESA: **Connect LinkedIn**. Für Company Pages: Toggle **Also connect Company Pages** aktivieren → Pages mit ADMINISTRATOR-Rolle werden gelistet.

Scopes (Member-Only): `openid profile w_member_social` · (mit Pages): zusätzlich `w_organization_social r_organization_social rw_organization_admin`.

Stolpersteine

- **Token-TTL:** Access-Tokens ~60 Tage, **kein Refresh-Token** im Standard-Tier — Driver markiert nach Expire `status='expired'`, User muss neu authorizen.
- **Versions-Header zwingend:** Jeder `/rest/posts`-Request braucht `LinkedIn-Version: 202405` und `X-Restli-Protocol-Version: 2.0.0` Header.
- **Post-Length:** Commentary 3000 Zeichen.
- **Company-Page-Posts** brauchen den `urn:li:organization:<id>`-Owner und das Community-Management-API-Product (App-Review-Dauer: 1-4 Wochen).

5c.7 Reddit

Auth-Typ: OAuth 2.0 (authorization code, `duration=permanent` für Refresh-Token) **Developer-Portal:** https://www.reddit.com/prefs/apps **App-Review nötig:** Nein **Kosten:** kostenlos für Personal-Use (großvolumige Commercial-Use seit 2023 paid — für Posting irrelevant)

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://www.reddit.com/api/v1/authorize`	Browser-Redirect zum User-Consent
POST	`https://www.reddit.com/api/v1/access_token`	Code → Access + Refresh (HTTP-Basic-Auth mit Client-ID/Secret)
GET	`https://oauth.reddit.com/api/v1/me`	Identity-Lookup (Username)
POST	`https://oauth.reddit.com/api/submit`	Link- oder Self-Post submitten

App-Registrierung (Schritt für Schritt)

1. Geh zu https://www.reddit.com/prefs/apps → ganz unten **are you a developer? create an app...**.
2. Trage ein:
3. **Name:** `EasySocialAutopilot`
4. **Type: web app** (NICHT installed/script — du brauchst Confidential-Client + Refresh-Tokens)
5. **Description:** "WordPress-Plugin zur automatischen Verteilung von Blog-Posts"
6. **about url:** Link auf deine WordPress-Site
7. **redirect uri:** `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=reddit`
8. **create app** klicken. Du siehst:
9. **Client-ID:** kurzer String direkt UNTER dem App-Namen (nicht das "personal use script"-Label, das ist nur Hinweistext).
10. **secret:** in der Detail-Ansicht aufgelistet.

11. In `wp-config.php`:

```
php    define(    'ESA_REDDIT_APP_ID',    'kurzer-client-id-string'    );    define( 'ESA_REDDIT_APP_SECRET', 'das-secret' );
```

1. In ESA: **Connect Reddit** → Consent zustimmen.

Scopes: `identity submit flair mysubreddits read`.

Stolpersteine

- **User-Agent ist Pflicht:** Reddit blockt Standard-User-Agents (curl, python-requests). ESA sendet `EasySocialAutopilot/<version> by /u/<username>` — das ist Reddit-Konvention.
- **Subreddit-Rules ≠ API-Rules:** Submit-Endpoint akzeptiert deinen Post technisch, die Subreddit-Moderation kann ihn aber sofort entfernen (auto-mod regex, flair-Pflicht, account-age-Limit). Errors kommen NICHT von der API zurück — Posts erscheinen in deinem History mit Reddit-URL, sind im Sub aber unsichtbar.
- **`subreddit`-Option ist Pflicht** pro Post (ohne `r/`-Prefix). Ohne → SendResult-Failure noch vor dem API-Call.
- **NSFW/Spoiler/Flair:** Pro Sub unterschiedliche Anforderungen — die Driver-Options `nsfw`, `spoiler`, `flair_id` decken die häufigsten ab.

5c.8 Pinterest

Auth-Typ: OAuth 2.0 (authorization code + refresh) **Developer-Portal:** https://developers.pinterest.com/apps/ **App-Review nötig:** Nein für Standard-Tier (Production aktivierbar ohne Review für die Basis-Scopes) **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://www.pinterest.com/oauth/`	Browser-Redirect zum User-Consent
POST	`https://api.pinterest.com/v5/oauth/token`	Code → Access (~30 d) + Refresh (~60 d)
GET	`https://api.pinterest.com/v5/user_account`	Identity-Lookup
GET	`https://api.pinterest.com/v5/boards?page_size=25`	Boards listen (für Default-Board-Cache)
POST	`https://api.pinterest.com/v5/pins`	Pin erstellen (Image zwingend)

App-Registrierung (Schritt für Schritt)

1. Geh zu https://developers.pinterest.com/apps/ → **Connect app** / **Create app**.
2. Trage ein:
3. **App Name:** `EasySocialAutopilot`
4. **Description:** "WordPress-Plugin für automatisierte Pin-Veröffentlichung aus Blog-Posts"
5. **Redirect URIs:** `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=pinterest`
6. Speichern → App-ID + App-Secret auf der Detail-Seite kopieren.
7. In `wp-config.php`:

```
php define( 'ESA_PINTEREST_APP_ID', 'deine-app-id' ); define( 'ESA_PINTEREST_APP_SECRET', 'dein-app-secret' );
```

1. In ESA: **Connect Pinterest** → Consent. Beim ersten Connect cached der Driver das erste Board als Default — in den Per-Account-Settings änderbar.

Scopes: `boards:read,pins:read,pins:write,user_accounts:read`.

Stolpersteine

- **Bild ist Pflicht:** Pinterest erlaubt keine Text-only-Pins. Posts ohne Featured Image failen mit klarer Fehlermeldung VOR dem API-Call.
- **Trusted-Domain für Image-URLs:** Pinterest lehnt Pins mit Image-URLs ab, die auf einer noch nie gepinnten Domain liegen. Beim ersten Run von deiner Domain dauert es manchmal Stunden bis Pinterest sie als "trusted" einstuft.
- **Token-Rotation:** Access-Token 30 d, Refresh-Token 60 d. Driver refreshed automatisch wenn `expires_at - 5min < now`. Wer 60 Tage nichts postet, muss neu connecten.
- **Bord-Sections:** Nur, wenn der Board Sections hat. `section_id`-Option ohne Sections → 422.

5c.9 Tumblr

Auth-Typ: OAuth 1.0a HMAC-SHA1 (historisch bedingt — Tumblr hat nie auf OAuth 2 migriert) **Developer-Portal:** https://www.tumblr.com/oauth/apps **App-Review nötig:** Nein **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
POST	`https://www.tumblr.com/oauth/request_token`	Step 1 — Request-Token holen
GET	`https://www.tumblr.com/oauth/authorize`	Browser-Redirect zum User-Consent
POST	`https://www.tumblr.com/oauth/access_token`	Step 3 — Verifier → Access-Token + Access-Secret
GET	`https://api.tumblr.com/v2/user/info`	Primary-Blog + Username ermitteln
POST	`https://api.tumblr.com/v2/blog/<blog-id>/post`	Text-Post als Legacy form-encoded publishen

App-Registrierung (Schritt für Schritt)

1. Geh zu https://www.tumblr.com/oauth/apps → **Register application**.

2. Trage ein:
3. **Application Name:** `EasySocialAutopilot`
4. **Application Website:** deine WordPress-URL
5. **Application Description:** "WordPress-Plugin zur automatischen Tumblr-Veröffentlichung"
6. **Administrative contact email:** deine
7. **Default callback URL:** `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=tumblr`
8. **Register** klicken → die App-Detail-Seite zeigt **OAuth Consumer Key** und **Secret Key**.
9. In `wp-config.php`:

```
php define( 'ESA_TUMBLR_CONSUMER_KEY', 'consumer-key' ); define( 'ESA_TUMBLR_CONSUMER_SECRET', 'consumer-secret' );
```

1. In ESA: **Connect Tumblr** → 3-Step OAuth 1.0a Flow läuft, Primary-Blog wird via `/v2/user/info` ermittelt.

Stolpersteine

- **OAuth 1.0a:** Jeder API-Call wird mit Consumer-Secret + Token-Secret HMAC-SHA1-signiert. Driver bringt eigene `TumblrOAuth1`-Implementierung mit (potenziell auch für Xing recyclebar).
- **Keine Refresh-Tokens:** OAuth-1.0a-Access-Tokens sind permanent gültig bis User-Revoke. Kein Token-Lifetime-Issue, aber kein Force-Reauth-Hebel.
- **Legacy-Format statt NPF:** Der Driver nutzt den `/v2/blog/<blog-id>/post`-Endpoint mit `type=text` und HTML-Body (Legacy form-encoded), NICHT den neueren NPF-Endpoint (`/posts`). Stabiler, aber keine Story-Blocks oder Reblogs möglich.
- **Primary-Blog only:** Der Driver postet auf den Primary-Blog des authentifizierten Users. Für Secondary-Blogs müsstest du den Driver erweitern oder pro Sekundärblog einen separaten Tumblr-User connecten.

5c.10 WordPress.com

Auth-Typ: OAuth 2.0 (authorization code) **Developer-Portal:** https://developer.wordpress.com/apps/ **App-Review nötig:** Nein **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://public-api.wordpress.com/oauth2/authorize`	Browser-Redirect zum User-Consent
POST	`https://public-api.wordpress.com/oauth2/token`	Code → Bearer-Token (~2 Wochen, kein Refresh)
GET	`https://public-api.wordpress.com/rest/v1.1/me`	Identity-Lookup
GET	`https://public-api.wordpress.com/rest/v1.1/me/sites`	Multi-Site-Discovery
POST	`https://public-api.wordpress.com/rest/v1.1/sites/<site>/posts/new`	Post auf Ziel-Site publishen

App-Registrierung (Schritt für Schritt)

1. Geh zu https://developer.wordpress.com/apps/ → **Create New Application**.
2. Trage ein:
3. **Name:** `EasySocialAutopilot`
4. **Description:** "WordPress-Plugin zur Cross-Posting-Verteilung"
5. **Website URL:** deine Site
6. **Redirect URLs:** `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=wordpress_com`
7. **Type: Web**
8. **Create** klicken → Client-ID und Client-Secret auf der Detail-Seite.
9. In `wp-config.php`:

```
php define( 'ESA_WPCOM_CLIENT_ID', 'deine-client-id' ); define( 'ESA_WPCOM_CLIENT_SECRET', 'dein-client-secret' );
```

1. In ESA: **Connect WordPress.com** → Consent → User wählt die Ziel-Site aus `/me/sites`.

Scope: `posts` (Default). Für Multi-Site-Workflows mit Stats-Lookups optional `posts global`.

Stolpersteine

- **Bearer-Token ~2 Wochen TTL, kein Refresh-Token:** Bei Expire setzt der Driver `status='expired'`, User muss komplett neu authorizen. Plane den Re-Auth-Flow in deine Operations ein.
- **Jetpack-Connected Self-Hosted Sites** zählen ebenfalls als WordPress.com-Sites — du kannst über diesen Channel auch auf Jetpack-verbundene Self-Hosted-WordPress-Installationen posten.
- **Mehrere Sites pro Token:** Jede Site bekommt einen eigenen ESA-`ChannelAccount`. Beim Connect listet ESA dir alle Sites zur Auswahl.

5c.11 Medium

Auth-Typ: Integration-Token (kein OAuth — der Onboarding-Pfad ist seit März 2023 geschlossen) **Developer-Portal:** keiner mehr — Token-Generierung in User-Settings unter https://medium.com/me/settings/security **App-Review nötig:** Nein (auch nicht möglich) **Kosten:** kostenlos solange dein Account eligible ist

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://api.medium.com/v1/me`	Identity-Verifikation beim Connect (User-ID holen)
POST	`https://api.medium.com/v1/users/<id>/posts`	Post publishen

App-Registrierung (Schritt für Schritt)

1. Bei Medium einloggen → **Settings → Security & apps → Integration tokens**.
2. Token-Description eintragen (z. B. `EasySocialAutopilot`) → **Get integration token**.
3. Hex-Token wird einmal angezeigt — sofort kopieren.
4. In ESA: **Connect Medium** → Token in Connect-Dialog einfügen. Driver ruft `GET /v1/me` zur Verifikation.

Wenn der Settings-Bereich **Integration tokens** bei dir nicht erscheint: dein Account ist nicht eligible (Medium hat die Funktion für viele Accounts deaktiviert) — Channel nicht nutzbar.

Keine `wp-config.php`-Konstanten.

Stolpersteine

- **OAuth-Onboarding geschlossen:** Seit März 2023 nimmt Medium keine neuen OAuth-Partner mehr auf. Das `mediumxapi`-GitHub-Repo ist archiviert. Integration-Token-Pfad ist der einzige Weg.
- **API kann jederzeit weggehen:** Wenn `POST /v1/users/<id>/posts` mal 404/410 zurückgibt, zeigt der Driver einen ehrlichen "Medium API is no longer publicly accessible"-Fehler. Hab einen Plan B.
- **`canonical_url`-Default true:** Sehr empfohlen — Google sieht deinen WordPress-Original-Post als Quelle, Medium-Republish bekommt keinen Duplicate-Content-Malus.
- **Max. 5 Tags** pro Post.

5c.12 Google Business Profile

Auth-Typ: OAuth 2.0 (authorization code + refresh) via Google Cloud **Developer-Portal:** https://console.cloud.google.com **App-Review nötig:** Ja für Production-Posting (Google-App-Verification) — Test-User-Modus reicht für Self-Use **Kosten:** kostenlos (Google Cloud Free-Tier reicht)

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://accounts.google.com/o/oauth2/v2/auth`	Browser-Redirect zum User-Consent
POST	`https://oauth2.googleapis.com/token`	Code → Access + Refresh
GET	`https://mybusinessaccountmanagement.googleapis.com/v1/accounts?pageSize=20`	GBP-Accounts des Users listen
GET	`https://mybusinessbusinessinformation.googleapis.com/v1/<account>/locations?...`	Locations pro Account listen
POST	`https://mybusiness.googleapis.com/v4/<location>/localPosts`	LocalPost publishen

App-Registrierung (Schritt für Schritt)

1. Geh zu https://console.cloud.google.com → Projekt anlegen oder bestehendes wählen.
2. **APIs & Services → Library** → aktivieren:
3. **Business Profile API**
4. **My Business Account Management API**
5. **My Business Business Information API**
6. **APIs & Services → OAuth consent screen** → User-Type **External** → App-Info ausfüllen (App-Name, Support-Email, Developer-Email) → Scope `https://www.googleapis.com/auth/business.manage` adden → **Test users** dich selbst eintragen.
7. **APIs & Services → Credentials → Create credentials → OAuth client ID**:
8. Application type: **Web application**
9. Name: `EasySocialAutopilot – GBP`
10. Authorized redirect URI: `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=google_business_profile`
11. Client-ID + Secret auf der Detail-Seite.
12. In `wp-config.php` (geteilt mit YouTube + Blogger):

```
php define( 'ESA_GOOGLE_APP_ID', 'xyz.apps.googleusercontent.com' ); define( 'ESA_GOOGLE_APP_SECRET', 'GOCSPX-...' );
```

1. In ESA: **Connect Google Business Profile** → Consent → eine ESA-`ChannelAccount` pro Location.

Scope: `https://www.googleapis.com/auth/business.manage`.

Stolpersteine

- **API-Quotas:** Business Profile API hat Default-Quota 1 Request/Minute pro Project — du musst per Google-Cloud-Console eine Quota-Erhöhung beantragen (Formular ausfüllen, ~1 Woche Wartezeit). Sonst limitierst du dich auf 1 Post pro Minute.
- **App-Verification:** Production-Posting ohne Test-User-Modus erfordert Google-App-Verification (Privacy-Policy, Sicherheits-Review, Domain-Verifikation) — Wochen Wartezeit. Für Self-Use bleib im Test-User-Modus.
- **Post-Lifetime 7 Tage:** GBP-Posts laufen automatisch nach 7 Tagen ab (Google-Policy). Der Post bleibt im History, ist auf dem GBP-Profil aber nicht mehr sichtbar.
- **Image-URL muss öffentlich erreichbar sein:** Lokale `wp-content/uploads/`-URLs auf staging-Domains failen.

5c.13 YouTube

Auth-Typ: OAuth 2.0 (authorization code + refresh) via Google Cloud **Developer-Portal:** https://console.cloud.google.com (gleiches Projekt wie GBP + Blogger) **App-Review nötig:** Nein für `description_update`-Mode. `video_upload` würde `youtube.upload`-Scope + Branding-Verification erfordern (Wochen). **Kosten:** kostenlos (Quota: 10.000 Units/Tag)

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://accounts.google.com/o/oauth2/v2/auth`	Browser-Redirect (shared Google-OAuth)
POST	`https://oauth2.googleapis.com/token`	Code → Access + Refresh
GET	`https://www.googleapis.com/youtube/v3/channels?part=snippet&mine=true`	Channel-Identity beim Connect
GET	`https://www.googleapis.com/youtube/v3/videos?part=snippet&id=<video-id>`	Bestehende Video-Description holen
PUT	`https://www.googleapis.com/youtube/v3/videos?part=snippet`	Description updaten (append/prepend)

App-Registrierung (Schritt für Schritt)

1. Im gleichen Google-Cloud-Projekt wie GBP: **APIs & Services → Library → YouTube Data API v3** aktivieren.
2. **OAuth consent screen** → Scope `https://www.googleapis.com/auth/youtube` hinzufügen.
3. **Credentials** → optional einen zweiten OAuth-Client erstellen (oder den gleichen wie GBP verwenden):
4. Authorized redirect URI: `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=youtube`
5. Konstanten sind dieselben (`ESA_GOOGLE_APP_ID` + `ESA_GOOGLE_APP_SECRET`).
6. In ESA: **Connect YouTube** → Consent → Channel erscheint als ESA-`ChannelAccount`.

Scope: `https://www.googleapis.com/auth/youtube`.

Stolpersteine

- **Nur `description_update`-Mode funktioniert:** Per-Post: `mode=description_update`, `video_id` Pflicht, `prepend` (Bool — Default Append). Perfekt für Show-Notes oder Affiliate-Links an existierende Videos. `mode=video_upload` und `mode=community_post` returnen ehrlich `SendResult::failure`.
- **`video_upload` ist real schwer:** Braucht den sensitiven `youtube.upload`-Scope, Google-Branding-Verification UND eine Resumable-Multipart-Upload-Pipeline. Outside-of-Scope für die Driver-Iteration.
- **`community_post` ist NICHT API-erreichbar:** Google hat dafür keine öffentliche API — nur YouTube Studio im Browser. Der Driver sagt das ehrlich.
- **Quota Units:** Description-Update = 50 Units pro Call. 10.000 Units/Tag → ~200 Updates/Tag (mehr als genug).

5c.14 Blogger

Auth-Typ: OAuth 2.0 (authorization code + refresh) via Google Cloud **Developer-Portal:** https://console.cloud.google.com (gleiches Projekt wie GBP + YouTube) **App-Review nötig:** Nein **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://accounts.google.com/o/oauth2/v2/auth`	Browser-Redirect (shared Google-OAuth)
POST	`https://oauth2.googleapis.com/token`	Code → Access + Refresh
GET	`https://www.googleapis.com/blogger/v3/users/self/blogs`	Blogs des Users listen
POST	`https://www.googleapis.com/blogger/v3/blogs/<blog-id>/posts/?isDraft=<bool>`	Post publishen

App-Registrierung (Schritt für Schritt)

1. Im gleichen Google-Cloud-Projekt: **APIs & Services → Library → Blogger API v3** aktivieren.

2. **OAuth consent screen** → Scope `https://www.googleapis.com/auth/blogger` hinzufügen.
3. **Credentials** → bestehenden OAuth-Client erweitern oder neuen erstellen mit Redirect: `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=blogger`.
4. Konstanten: dieselben (`ESA_GOOGLE_APP_ID` + `ESA_GOOGLE_APP_SECRET`).
5. In ESA: **Connect Blogger** → eine ESA-`ChannelAccount` pro Blog (eine Google-Identität kann mehrere Blogs haben — jeden separat verbinden).

Scope: `https://www.googleapis.com/auth/blogger`.

Stolpersteine

- **HTML-Body wird direkt akzeptiert:** ESA lässt den Post durch den `the_content`-Filter, Blogger nimmt das Ergebnis 1:1. Keine Markdown-Konvertierung nötig.
- **Pro Blog ein Account:** Du musst pro Blogger-Blog einen separaten ESA-`ChannelAccount` connecten, sonst weiß ESA nicht, wohin posten.
- **Blogger ist Maintenance-Mode-Service:** Google entwickelt Blogger nicht aktiv weiter. Die API ist seit Jahren stabil, aber rechne nicht mit neuen Features.

5c.15 Facebook Pages

Auth-Typ: OAuth 2.0 (Meta-OAuth, Long-Lived-User-Token → Perpetual-Page-Token) **Developer-Portal:** https://developers.facebook.com/apps/ **App-Review nötig:** Ja — `pages_manage_posts` ist eine Review-Permission. **Connect funktional, Posten BLOCKIERT** bis App Review abgeschlossen. **Kosten:** kostenlos (App-Review-Submission selbst kostet nichts, aber Aufwand)

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://www.facebook.com/v21.0/dialog/oauth`	Browser-Redirect zum User-Consent
GET	`https://graph.facebook.com/v21.0/oauth/access_token`	Code → Short-Lived-User-Token
GET	`https://graph.facebook.com/v21.0/oauth/access_token?grant_type=fb_exchange_token`	Short → Long-Lived (~60 d)
GET	`https://graph.facebook.com/v21.0/me/accounts?fields=id,name,access_token,category`	Pages des Users + perpetual Page-Access-Tokens
POST	`https://graph.facebook.com/v21.0/<page-id>/feed`	**(BLOCKIERT)** Post publishen

App-Registrierung (Schritt für Schritt)

1. Geh zu https://developers.facebook.com/apps/ → **Create App**.
2. **Use case** wählen: **Other** → **Business** (für Pages-Zugang).
3. App-Name `EasySocialAutopilot` → **Create app**.
4. App-Dashboard → **Add product** → **Facebook Login for Business** (Settings):
5. Valid OAuth Redirect URI: `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=facebook`
6. **App settings → Basic** → App-ID + App-Secret kopieren.
7. **App Review → Permissions and Features** → anfordern:
8. `pages_show_list` (Standard-Access, ohne Review)
9. `pages_read_engagement` (Standard-Access)
10. `pages_manage_posts` (**Advanced Access — REVIEW PFLICHT**)
11. `business_management` (Advanced Access — Review nötig wenn du Business-Manager-Pages willst)
12. Für `pages_manage_posts`-Review: Screencast, Privacy Policy, Use-Case-Beschreibung einreichen. **Dauer: 2-6 Wochen.**
13. In `wp-config.php` (geteilt mit Instagram + Threads):

```
php define( 'ESA_META_APP_ID', 'deine-meta-app-id' ); define( 'ESA_META_APP_SECRET', 'dein-meta-app-secret' );
```

1. In ESA: **Connect Facebook Pages** → Consent → Pages erscheinen als `ChannelAccount`s.

Scopes: `pages_show_list, pages_read_engagement, pages_manage_posts, business_management`.

Stolpersteine

- **Posting ist STUB:** Der Driver `send()` returnt `SendResult::failure('POSTING_BLOCKED: Facebook App Review for pages_manage_posts scope')`. Connect, Page-Discovery + Token-Management laufen real — Posten wird live, sobald du den Stub entfernst und Review durch ist.
- **Long-Lived-User-Token ~60 Tage:** Wenn du den nicht refreshest, sind die Page-Tokens auch tot. Der Driver baut den User-Token einmal um — Page-Tokens sind dann perpetual.
- **App-Modus:** Live-Mode + öffentlicher Privacy-Policy + URL-Verifikation für Review zwingend. Dev-Mode-Apps können nur App-Admins als Test-User.
- **`business_management`-Scope** ist nur nötig für Pages, die in einem Business Manager liegen — sonst weglassen.

5c.16 Instagram

Auth-Typ: OAuth 2.0 via Meta-Login + Instagram-Business-Account-Verknüpfung über Facebook Page **Developer-Portal:** https://developers.facebook.com/apps/ (gleiche Meta-App wie Facebook + Threads) **App-Review nötig:** Ja — `instagram_content_publish` ist Review-Pflicht **plus** der User muss einen Instagram-Business-Account mit einer FB-Page verknüpft haben. **Connect funktional, Posten BLOCKIERT. Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://www.facebook.com/v21.0/dialog/oauth`	Browser-Redirect (shared Meta-OAuth)
GET	`https://graph.facebook.com/v21.0/oauth/access_token`	Code → Access-Token
GET	`https://graph.facebook.com/v21.0/me/accounts?fields=instagram_business_account`	IG-Business-Account-Discovery via FB-Pages
GET	`https://graph.facebook.com/v21.0/<ig-user-id>?fields=username,profile_picture_url`	IG-Identity-Lookup
POST	`https://graph.facebook.com/v21.0/<ig-user-id>/media`	**(BLOCKIERT)** Step 1 — Media-Container erzeugen
POST	`https://graph.facebook.com/v21.0/<ig-user-id>/media_publish`	**(BLOCKIERT)** Step 2 — Container publishen

App-Registrierung (Schritt für Schritt)

1. Dieselbe Meta-App wie Facebook Pages (siehe 5c.15) — IG braucht keinen separaten App-Eintrag.
2. **App Review → Permissions** zusätzlich anfordern:
3. `instagram_basic` (Standard-Access)
4. `instagram_content_publish` (**Advanced — REVIEW PFLICHT**)
5. User-seitig: dein Instagram-Account muss
6. auf **Business** oder **Creator** umgestellt sein (Settings → Account type),
7. mit einer **Facebook Page** verknüpft sein (Settings → Linked accounts).
8. Konstanten: dieselben (`ESA_META_APP_ID` + `ESA_META_APP_SECRET`).
9. In ESA: **Connect Instagram** → Consent → Driver fragt `/me/accounts?fields=instagram_business_account` ab und listet die verknüpften IG-Business-Accounts.

Scopes: `pages_show_list, pages_read_engagement, business_management, instagram_basic, instagram_content_publish`.

Stolpersteine

- **Posting ist STUB:** Wie Facebook — Connect funktioniert, `send()` returnt `POSTING_BLOCKED`.
- **2-Step Publishing:** Container-API mit Polling auf `status_code=FINISHED` ist Pflicht. Synchrone Single-Call-Publishs gibt's nicht.
- **Image-URL muss öffentlich + HTTPS** sein, kein Localhost-Staging. JPEG, 8 MB max für Single-Image, 100 MB für Video.
- **Kein DMs, keine Stories, keine Carousels** in der ersten Iteration. Nur Single-Image-Feed-Posts.
- **App Review für Instagram + Facebook in einem Submit** möglich — spart Zeit.

5c.17 Threads

Auth-Typ: OAuth 2.0 (Threads-eigenes OAuth, **nicht** Facebook-Login — andere Auth- und Token-Hosts) **Developer-Portal:** https://developers.facebook.com/apps/ (Threads als Product zur App hinzufügen) **App-Review nötig:** Ja — `threads_content_publish`. Threads-API ist generell Limited Rollout. **Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://threads.net/oauth/authorize`	Browser-Redirect (NICHT facebook.com!)
POST	`https://graph.threads.net/oauth/access_token`	Code → Short-Lived-Token
GET	`https://graph.threads.net/access_token?grant_type=th_exchange_token`	Long-Lived-Exchange (~60 d)
GET	`https://graph.threads.net/v1.0/me?fields=id,username,threads_profile_picture_url`	Identity-Lookup
POST	`https://graph.threads.net/v1.0/<th-user-id>/threads`	**(BLOCKIERT)** Step 1 — Container
POST	`https://graph.threads.net/v1.0/<th-user-id>/threads_publish`	**(BLOCKIERT)** Step 2 — Publish

App-Registrierung (Schritt für Schritt)

1. Dieselbe Meta-App wie Facebook + Instagram. Im App-Dashboard: **Add product → Threads API → Set up**.
2. **Threads → Settings:**
3. Redirect Callback URLs: `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=threads`
4. Deauthorize Callback URL: ebenfalls deine Domain (Pflichtfeld)
5. **App Review** für `threads_content_publish` einreichen (analog zu IG).
6. Konstanten: dieselben (`ESA_META_APP_ID` + `ESA_META_APP_SECRET`) — aber der Driver baut die Authorize-URL gegen `threads.net` , nicht `facebook.com` .
7. In ESA: **Connect Threads** → Consent.

Scopes: `threads_basic, threads_content_publish` .

Stolpersteine

- **Posting ist STUB:** `POSTING_BLOCKED: Threads API access (currently limited rollout)`.
- **Andere Hosts als Facebook:** Auth über `threads.net`, Token + Graph über `graph.threads.net`. Wer das verwechselt, kriegt CORS-Fehler oder 404.
- **Long-Lived-Exchange ist ein eigener GET-Call** (`grant_type=th_exchange_token`), keine POST. Driver macht das richtig — bei manueller Inspection nicht verwirren lassen.
- **2-Step Container-Publish** wie Instagram — gleiches Polling-Pattern.

5c.18 TikTok

Auth-Typ: OAuth 2.0 (authorization code + refresh) **Developer-Portal:** https://developers.tiktok.com/apps **App-Review nötig:** Ja zweistufig — Sandbox → Production-Audit für `video.publish`. **Connect funktional, Posten BLOCKIERT. Kosten:** kostenlos

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://www.tiktok.com/v2/auth/authorize/`	Browser-Redirect zum User-Consent
POST	`https://open.tiktokapis.com/v2/oauth/token/`	Code → Access + Refresh
GET	`https://open.tiktokapis.com/v2/user/info/`	Identity-Lookup
POST	`https://open.tiktokapis.com/v2/post/publish/video/init/`	**(BLOCKIERT)** Step 1 — Upload-Session initialisieren
PUT	`<upload_url aus Step 1>`	**(BLOCKIERT)** Step 2 — Video-Bytes hochladen
POST	`https://open.tiktokapis.com/v2/post/publish/status/fetch/`	**(BLOCKIERT)** Step 3 — Status pollen

App-Registrierung (Schritt für Schritt)

1. Geh zu https://developers.tiktok.com/apps → **Connect** mit deinem TikTok-Account → **Manage apps → Create an app**.
2. App-Basisinfos: Name `EasySocialAutopilot`, Description, Category.
3. **Configuration:**
4. Redirect domain: `<deine-domain>`
5. Redirect URI: `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=tiktok`
6. Terms of Service URL + Privacy Policy URL: Pflicht
7. **Products → Login Kit + Content Posting API** aktivieren.
8. Scopes: `user.info.basic` (sofort), `video.publish` (Sandbox sofort, Production nach Audit).
9. **Client Key + Client Secret** aus dem App-Dashboard.
10. In `wp-config.php`:

```
php define( 'ESA_TIKTOK_CLIENT_KEY', 'deine-client-key' ); define( 'ESA_TIKTOK_CLIENT_SECRET', 'dein-client-secret' );
```

1. In ESA: **Connect TikTok** → Consent.

Scopes: `user.info.basic, video.publish`.

Stolpersteine

- **TikTok hat KEINE Text-only-Posts.** Jeder Publish braucht ein Video. Der Driver fail-fast wenn `options['video_url']` fehlt — schon VOR dem Posting-Block.
- **Posting ist STUB:** Auch wenn du `video_url` setzt, returnt der Driver `POSTING_BLOCKED: TikTok Developer Sandbox + Production Review`.
- **Sandbox = Privacy SELF_ONLY:** Vor dem Production-Audit sind alle Posts nur für deinen eigenen Account sichtbar. Nicht erschrecken.
- **Audio-License-Check:** TikTok lehnt Videos mit nicht-lizenzierter Hintergrundmusik ab — diese Pre-Validation läuft serverseitig, kommt als Async-Status-Failure.
- **Refresh-Token rotiert:** Bei jedem Refresh kommt ein neuer Refresh-Token zurück — Driver speichert den neuen, alter wird invalid.

5c.19 X (Twitter)

Auth-Typ: OAuth 2.0 mit **PKCE** (Public oder Confidential Client) **Developer-Portal:** https://developer.x.com/en/portal/dashboard (alias developer.twitter.com) **App-Review nötig:** Nein für API-Zugang — aber **Basic-Tier ist Paid** ($100+/Monat) für Posting. Free-Tier erlaubt nur Read + Login. **Connect funktional, Posten BLOCKIERT pending Paid-Subscription. Kosten:** Free $0 (read-only); Basic $100/Monat; Pro $5000/Monat. Posting braucht mindestens Basic.

API-Endpoints (was unser Driver aufruft)

METHODE	ENDPOINT	ZWECK
GET	`https://twitter.com/i/oauth2/authorize`	Browser-Redirect mit PKCE-Challenge (S256)
POST	`https://api.twitter.com/2/oauth2/token`	Code + Verifier → Access + Refresh (HTTP-Basic für Confidential-Clients)
GET	`https://api.twitter.com/2/users/me`	Identity-Lookup
POST	`https://api.twitter.com/2/tweets`	**(BLOCKIERT)** Tweet publishen, max. 280 Zeichen Free / 25.000 Premium

App-Registrierung (Schritt für Schritt)

1. Geh zu https://developer.x.com/en/portal/dashboard → wenn neu: **Sign up for Free Account** → Use-Case ausfüllen.
2. **Projects & Apps → + Add App** → App-Name `EasySocialAutopilot`.
3. **App Settings → User authentication settings → Set up:**
4. Type: **Web App, Automated App or Bot** (= Confidential Client mit Secret)
5. App permissions: **Read and write**
6. Callback URI: `https://<deine-domain>/wp-admin/admin-post.php?action=esa_oauth_callback&channel=x`
7. Website URL: deine Domain
8. **Keys and tokens → OAuth 2.0 Client ID and Client Secret** kopieren (Client Secret wird einmal angezeigt).
9. In `wp-config.php`:

```
php define( 'ESA_X_CLIENT_ID', 'deine-oauth2-client-id' ); define( 'ESA_X_CLIENT_SECRET', 'dein-oauth2-secret' ); // optional, nur für Confidential-Client
```

1. In ESA: **Connect X** → Consent. Driver baut PKCE-Challenge: `code_verifier` = 64 zufällige Bytes (base64url), `code_challenge` = `base64url(SHA256(verifier))`, Methode `S256`.

Scopes: `tweet.read, tweet.write, users.read, offline.access` (Pflicht für Refresh-Token).

Stolpersteine

- **Posting ist STUB:** `POSTING_BLOCKED: X API v2 Basic tier ($100+/month) for posting`. Connect, Token-Refresh, Identity laufen — Posting wird live, sobald du den Paid-Tier subscribed UND den Stub entfernst.
- **Free-Tier Schreib-Limit:** Selbst mit Paid-Tier — Free hat 1.500 Tweets/Monat, Basic 50.000.
- **OAuth-Endpoints heißen noch `twitter.com`:** Trotz Rebrand zu X.com werden die alten Domains weiter bedient. Nicht ändern.
- **Refresh-Token rotiert nicht:** Anders als TikTok bleibt der Refresh-Token gültig, aber Access-Token verfällt nach 2 h. Driver refreshed eagerly.
- **API v1.1 wird abgeschaltet:** Wer noch alte Snippets aus v1.1-Zeiten hat, kann sie nicht mehr nutzen. ESA ist v2-only.

5c.20 XING

Auth-Typ: keiner — API discontinued **Developer-Portal:** **nicht mehr existent** (`dev.xing.com` ist tot seit 2022) **App-Review nötig:** nicht möglich **Kosten:** —

API-Endpoints (was unser Driver historisch aufrufen würde)

METHODE	ENDPOINT (NICHT MEHR ERREICHBAR)	ZWECK
POST	`https://api.xing.com/v1/users/me/status_message`	Status-Update publishen (404 seit 2022)
OAuth 1.0a	`https://api.xing.com/v1/request_token` etc.	Historischer Auth-Flow (auch tot)

App-Registrierung (Schritt für Schritt)

Nicht möglich. XING hat seine öffentliche Developer-API 2022 ohne Ersatz abgeschaltet.

Manueller Workaround:

1. Nutze **XING Business Manager** im Browser zum Scheduling von Company-Page-Posts.
2. Persönliche Status-Updates aus WordPress per Copy-Paste übertragen.
3. Wenn die API jemals zurückkommt: der Stub-Code in `src/Channels/Xing/` ist so gebaut, dass die Real-Implementierung 1:1 austauschbar ist. Die OAuth-1.0a-Implementierung von `\EasySocialAutopilot\Channels\Tumblr\TumblrOAuth1` ist recyclebar.

Keine `wp-config.php`-Konstanten.

Stolpersteine

- **Account-Status bei Connect:** Der Driver wirft eine klare `RuntimeException` mit der Discontinued-Message — du kannst keinen XING-Account in ESA verbinden.
- **`refreshIfNeeded()`** markiert bestehende Pre-2022-Accounts (falls überhaupt jemand welche aus Legacy-Daten hat) als `status='expired'`.
- **Kein offizielles ETA für Re-Launch.** XING hat 2024 mehrfach kommuniziert, dass eine neue API "evaluated" werde — keine Veröffentlichungs-Zusage.

Einen Post veröffentlichen (Phase-4-Flow)

Wie ein WordPress-Post automatisch in die Queue kommt — und welche Hebel du pro Post hast.

Auto-Enqueue ist live

Phase 4 liefert das Stück, das in Phase 3 noch gefehlt hat: einen Listener auf `transition_post_status`. Sobald du einen Post von `draft` auf `publish` setzt (egal ob im Block-Editor, Classic-Editor, per REST-API oder per WP-CLI), passiert Folgendes:

SCHRITT	AKTION
1	`PostHookListener::onTransition` feuert.
2	Liest alle aktiven Accounts aus `wp_esa_channel_accounts` (`status='active'`).
3	Wendet die Per-Post-Selection an (siehe unten).
4	Pro Account: `QueueRepository::hasActiveJob($post_id, $account_id)`-Check — wenn schon ein pending/processing-Job existiert, wird kein Duplikat angelegt (Idempotenz beim Re-Save).
5	Pro Account: ein Insert in `wp_esa_queue` mit `status='pending'`, `scheduled_for=now()`.
6	Beim nächsten Cron-Tick (max. 5 Min später, oder sofort per Button — siehe unten) greift `QueueProcessor::run` die Jobs auf und dispatcht.

Per-Post-Override via Post-Meta

Zwei Post-Meta-Keys steuern die Verteilung pro Post:

META-KEY	TYP	WIRKUNG
`_esa_selected_accounts`	Array von `account_id`s	Wenn gesetzt: nur diese Accounts kriegen einen Queue-Job. Wenn leer/nicht gesetzt: alle aktiven Accounts (Default).
`_esa_skip_distribution`	`1` / `0`	Wenn `1`: dieser Post wird gar nicht verteilt — kein Queue-Insert, egal was sonst.

Heute setzt du diese Meta-Keys per Code (z. B. ACF, `add_post_meta`, REST-API, `wp post meta add`). **Eine UI-Meta-Box / Gutenberg-Sidebar dafür kommt in Phase 5** — bis dahin ist die Verteilung "alle aktiven Accounts kriegen alles, außer du sagst per Meta explizit was anderes".

Skip-Distribution-Beispiel

```
# Diesen Post nicht verteilen
wp post meta add 123 _esa_skip_distribution 1

# Nur an Mastodon-Account #5 und Bluesky-Account #8
wp post meta update 123 _esa_selected_accounts '[5,8]' --format=json
```

Cron jetzt ausführen — der manuelle Trigger

Wenn du nicht 5 Minuten warten willst, gibt es auf der Queue-Page einen Button **Cron jetzt ausführen**. Klick → `POST /wp-json/esa/v1/cron/run` triggert `QueueProcessor::run` synchron im Request-Kontext. Im Toast siehst du Before/After-Counts (z. B. *"3 jobs processed: 7 → 4 pending"*). Praktisch beim Testen, Demos und für ungeduldige Redakteure. Capability-geschützt: nur `manage_options`.

Alternativ über WP-CLI:

```
wp cron event run esa_process_queue
```

Oder per curl gegen den WP-Cron-Endpoint:

```
curl https://<deine-site>/wp-cron.php?doing_wp_cron
```

Was geschieht beim Send

Pro Queue-Job ruft `QueueProcessor` den passenden `ChannelDriver::send($post, $account, $payload)`. Was der Driver pro Channel tut, steht in Kapitel 5b. Erfolg → Job-Eintrag in `wp_esa_jobs` mit `remote_post_id` + `remote_post_url`. Misserfolg → Retry mit Backoff (siehe nächstes Kapitel).

Queue verstehen

Was die Queue-Page zeigt, wie Retry funktioniert und welche Backoff-Strategie der Cron-Processor fährt.

Die Queue-Page (`#/queue`) liest defaultmäßig pending Jobs (Status-Filter via REST-Query-Param `status=processing|done|failed`). Tabellen-Spalten:

SPALTE	QUELLE	INHALT
`#`	`wp_esa_queue.id`	Primary-Key, monospace.
Channel	`wp_esa_queue.channel_id`	UPPERCASE-Anzeige.
Post	Lookup `wp_posts.post_title`	Titel des zu postenden Beitrags.
Status	`wp_esa_queue.status`	Pill: pending / processing / done / failed.
Scheduled	`wp_esa_queue.scheduled_for`	Lokales Datums-/Zeitformat.
Attempts	`wp_esa_queue.attempts`	Anzahl bisheriger Versuche (0–4).
Aktionen	–	Buttons **Retry** und **Cancel**.

Status-Werte

STATUS	BEDEUTUNG
`pending`	Wartet auf den nächsten Cron-Tick (`scheduled_for ⩽ now()`).
`processing`	Cron-Worker hat ihn gegriffen und ruft gerade den Driver.
`done`	Erfolgreich publiziert, Eintrag liegt in `wp_esa_jobs`.
`failed`	Terminal fehlgeschlagen (max. Attempts erreicht oder unbedingt-final-Fehler). Failure-Eintrag in `wp_esa_jobs`.

Retry- und Backoff-Strategie

VERSUCH	BACKOFF BIS ZUM NÄCHSTEN VERSUCH
1 (gerade fehlgeschlagen)	5 Minuten (300 s)
2	15 Minuten (900 s)
3	60 Minuten (3 600 s)
4	180 Minuten (10 800 s)
nach Versuch 4	terminal `failed`, kein weiterer Retry

Konstanten: `QueueProcessor::BACKOFF = [1⇒300, 2⇒900, 3⇒3600, 4⇒10800]`, `QueueRepository::MAX_ATTEMPTS = 4`.

Retry-Button

Setzt manuell `status='pending'`, `attempts=0`, `last_error=NULL`, `scheduled_for=now()`. Auch terminal `failed`-Einträge können so reanimiert werden — sinnvoll, wenn z. B. eine API kurzfristig down war.

Cancel-Button

Löscht den Queue-Eintrag. Keine History-Spur in `wp_esa_jobs`.

Batch-Größe

Pro Cron-Tick verarbeitet `QueueProcessor::run` maximal `BATCH_SIZE = 10` Jobs. Größere Backlogs werden seriell über mehrere Ticks abgearbeitet — das schont Remote-Rate-Limits.

Mehrere Channels gleichzeitig

Der typische Phase-4-Workflow: ein Post fließt automatisch an alle aktiven Accounts.

Standard-Workflow

1. Du verbindest in Kapitel 5/5b z. B. einen Mastodon-Account, einen Bluesky-Account, einen LinkedIn-Account und einen Telegram-Bot.
2. Du veröffentlichst einen WP-Post.
3. `PostHookListener` legt **vier** Queue-Jobs an — einen pro aktivem Account.
4. Der nächste Cron-Tick (oder Button-Klick) dispatcht alle vier parallel an die jeweiligen APIs.
5. Im History-Tab siehst du vier Outcome-Einträge mit Links auf den jeweiligen Remote-Post.

Mit Per-Post-Override

Wenn du nur einen Channel pro Post willst:

```
wp post meta update 999 _esa_selected_accounts '[12]' --format=json
```

Nur Account `12` kriegt einen Job. Die anderen aktiven Accounts werden für diesen Post übersprungen.

Performance-Hinweise

ASPEKT	WERT / VERHALTEN
Cron-Intervall	5 Minuten (`esa_five_minutes`)
Batch-Size pro Tick	10 Jobs
Backoff	5 → 15 → 60 → 180 Minuten
Max-Attempts	4
Idempotenz	`hasActiveJob` -Check verhindert Duplikate beim Re-Save
Manueller Trigger	Button **Cron jetzt ausführen** auf `#/queue`

Bei großen Backlogs (z. B. nach Mass-Republish): mehrfach **Cron jetzt ausführen** drücken oder per WP-CLI `wp cron event run esa_process_queue` schleifen lassen.

History

Was in `wp_esa_jobs` landet, was bewusst nicht geloggt wird und wie du Remote-Posts wiederfindest.

Die History-Page (`#/history`) listet alle Jobs aus `wp_esa_jobs` — also Einträge mit Status `success` oder terminal `failed` . **Retries werden bewusst NICHT in History geschrieben**, nur das finale Outcome. Das hält die History rauscharm.

Spalten

SPALTE	QUELLE	HINWEIS
`#`	`wp_esa_jobs.id`	Primary-Key.
Channel	`wp_esa_jobs.channel_id`	UPPERCASE.
Post	Lookup `wp_posts.post_title`	Titel des Beitrags.
Status	`wp_esa_jobs.status`	Pill: `success` (grün) oder `failed` (rot).
Completed	`wp_esa_jobs.executed_at`	UTC, in lokale Zeit konvertiert.
Remote	`wp_esa_jobs.remote_post_url`	Externer Link, öffnet in neuem Tab.

Datenfelder, die in `wp_esa_jobs` landen

SPALTE	INHALT
`queue_id`	Verweis auf den ursprünglichen Queue-Eintrag.
`channel_id`	Z. B. `mastodon` , `bluesky` , `linkedin` .
`account_id`	Welcher Account gepostet hat.
`remote_post_id`	ID auf der Remote-Plattform.
`remote_post_url`	Klickbare URL.
`response_data`	Komplette API-Response als JSON-Blob (Forensik).
`status`	`success` oder `failed` .
`executed_at`	UTC-Timestamp des Cron-Runs.

Bei terminal `failed`-Einträgen ist `remote_post_id` leer und `response_data` enthält den letzten Fehlerkontext.

Logs & Debugging

Wo du nachschaust, wenn etwas klemmt.

Logs-Page

`#/logs` zeigt den Application-Log-Stream aus `wp_esa_logs`. Filter-Pills oben (All / Debug / Info / Warn / Error). Limit 200 Einträge pro Reload. Wird von HTTP-Schicht (`HttpClient`) und Channel-Driver-Code populated.

Apache-Error-Log

Das eigentlich interessante Log liegt im Web-Server-Container:

```
docker exec wp-esa tail -f /var/log/apache2/error.log
```

Suche nach Prefixes:

PREFIX	QUELLE	BEDEUTET
`[ESA HTTP]`	`HttpClient` Retry-Logik bei 429/5xx	Outbound-HTTP-Calls Richtung Channel-APIs.
`[ESA OAuth]`	`OAuthHandler::handle`	OAuth-Callbacks, die nicht durchgegangen sind.
`[ESA <Channel>]`	Driver-interne Hinweise	Z. B. `[ESA Mastodon]`, `[ESA LinkedIn]`.
`[ESA Hook]`	`PostHookListener`	Auto-Enqueue-Trace beim `transition_post_status`.
`[ESA Cron]`	`QueueProcessor`	Batch-Run-Trace mit Job-Count + Outcome.

Häufige Fehler

FEHLERMELDUNG	URSACHE	LÖSUNG
`Failed to register Mastodon app on <instance> (status …)`	Instance falsch geschrieben, nicht erreichbar.	Eingabe prüfen.
`<Channel> OAuth state is unknown or expired.`	Transient ist abgelaufen (TTL 600 s).	Connect neu starten.
`<Channel> token exchange failed (status 400/401)`	Client-Secret nicht akzeptiert oder Redirect-Mismatch.	App-Konfiguration prüfen, Redirect-URI vergleichen.
`<Channel> verify_credentials failed (status 401)`	Token von der Plattform widerrufen.	Account trennen und neu verbinden.
`no driver registered for channel: <id>`	Queue-Eintrag verweist auf nicht-existenten Channel.	`channel_id`-Spalte prüfen.
`account not found: <id>`	Account getrennt, aber Queue-Eintrag existiert noch.	Queue-Eintrag löschen.
`Cannot decrypt credentials: …`	`wp-config.php`-Salts wurden rotiert.	Account löschen, neu verbinden.
`POSTING_BLOCKED` (Facebook/Instagram/Threads/TikTok/X)	Plattform-seitige Freigabe fehlt.	Siehe "Unlock"-Sektion der jeweiligen Channel-README.
`Pinterest: featured image required`	Post hat kein Featured Image.	Featured Image setzen oder Pinterest aus Verteilung ausschließen.
`TikTok: video_url required`	Post hat kein `video_url` in den Options.	Per-Post-Option setzen oder TikTok ausschließen.

Sicherheit

Wie das Plugin mit Tokens, Permissions und Nonces umgeht.

Token-Verschlüsselung

Access-Tokens / Refresh-Tokens / App-Passwords / Bot-Tokens / Webhook-URLs / HMAC-Secrets — alles wird **nie im Klartext** gespeichert. `Helpers\Encryption` nutzt **AES-256-GCM** mit einem zur Laufzeit aus drei WordPress-Salts abgeleiteten Schlüssel:

BESTANDTEIL	QUELLE
Schlüssel-Material	SHA-256 über `AUTH_KEY` + `SECURE_AUTH_SALT` + `NONCE_KEY` aus `wp-config.php`
IV-Länge	12 Bytes (per `random_bytes`)
Tag-Länge	16 Bytes
Payload-Format	base64(IV ‖ TAG ‖ CIPHERTEXT)

Wichtige Konsequenz: Wenn du deine `wp-config.php`-Salts rotierst, werden alle vorhandenen Token-Blobs unentschlüsselbar. Bei der nächsten Validierung fliegt `Cannot decrypt credentials` und der Account muss neu verbunden werden. Bewusstes Sicherheitsfeature, kein Bug.

Capability-Checks

Alle Admin-Pages und alle REST-Endpoints prüfen `current_user_can('manage_options')`. Editoren, Autoren oder Subscribers haben keinerlei Zugriff. Eine Custom-Capability-Filter (z. B. damit Editoren auch posten dürfen) ist in Phase 4 nicht eingebaut.

Nonces

Alle mutierenden REST-Routen verlangen einen `X-WP-Nonce`-Header mit der Action `wp_rest`. Die React-SPA bekommt den Nonce beim Admin-Asset-Enqueue und sendet ihn automatisch mit.

Kein Phone-Home, kein Tracking

- Keine Aktivierungs-Beacons.

- Keine Lizenz-Server-Checks.
- Keine Telemetrie über genutzte Channels.
- Keine externen JS-Bundles (alles wird mit der React-SPA gebundled und ausgeliefert).
- Outbound-Traffic geht ausschließlich zu Channel-APIs, mit denen du dich aktiv verbunden hast.

Idempotency-Keys

Wo die Plattform es unterstützt (Mastodon prominent), sendet der Driver einen `Idempotency-Key`-Header der Form `esa—<account_id>-<post_id>-<md5_prefix>`. Damit verhindert der Remote-Server, dass identische Posts bei einem versehentlichen Retry doppelt erscheinen. Auf der ESA-Seite tut zusätzlich `QueueRepository::hasActiveJob` denselben Dienst.

Roadmap (Phase 5+)

Was als Nächstes kommt.

Phase 5 — Per-Post-UI

FEATURE	BESCHREIBUNG
Gutenberg-Sidebar-Panel	Per-Post-Channel-/Account-Selection direkt im Block-Editor. Setzt `_esa_selected_accounts` und `_esa_skip_distribution`.
Classic-Editor Meta-Box	Pendant für Classic-Editor-Nutzer.
Per-Post Custom-Text	Statt Auto-Format pro Channel ein eigener Text-Override.
Default-Selection global	Settings-Page-Sektion: welche Channels sind per Default aktiv für neue Posts.

Phase 6 — Scheduling

FEATURE	BESCHREIBUNG
UI-Date-Picker im Editor	`scheduled_for` per Post setzen, statt nur Sofort-Publish.
Calendar-View	Wochen-/Monats-Kalender für Editorial-Pläne.
Re-Share-Scheduling	Älteren Post nach X Wochen automatisch nochmal posten.

Phase 7 — White-Label

FEATURE	BESCHREIBUNG
Per-Workspace Branding	Eigener App-Name auf Mastodon-Authorize-Page, eigene Logos.
Multi-Site Network	Network-Activate mit Site-isolierten Channel-Settings.
Per-Tenant API-Keys	Eine Plugin-Installation, mehrere mandantenfähige Channel-Pools.

Phase 8 — Bulk-Ops & Analytics

FEATURE	BESCHREIBUNG
Multi-Account Bulk-Ops	"Trenne alle Mastodon-Accounts", "Verbinde alle neu".
Per-Channel-Quotas	Rate-Limiting pro Account / Stunde / Tag.
Engagement-Analytics	Likes/Reposts/Replies pro Outbound-Post via Channel-Read-APIs.
Best-Time-to-Post	Heuristik aus historischen Engagement-Daten.

Technische Referenz

Architektur, DB-Schema, REST-Endpoints — für Entwickler, die eigene Driver oder Integrationen bauen wollen.

Architektur in Stichworten

- **Autoload:** PSR-4 unter `EasySocialAutopilot\` aus `src/` (siehe `composer.json`).
- **Bootstrap:** `Plugin::boot()` läuft auf `plugins_loaded`, registriert Loader, AdminMenu, OAuthHandler, RestApi, QueueProcessor, PostHookListener.
- **Channel-Driver-Contract:** abstrakter `ChannelDriver` mit acht abstract methods — `getId`, `getName`, `getIcon`, `getOAuthStartUrl`, `handleOAuthCallback`, `refreshIfNeeded`, `validate`, `send`. Jeder neue Channel ist eine konkrete Subklasse.
- **Registry-Singleton:** `Channels\Registry::getInstance()` feuert beim ersten Zugriff die Action `esa_register_channels`, an die sich Built-ins (via `Channels\Bootstrap`) und **Drittanbieter** ihre eigenen Drivers hängen können. Das ist der offizielle Extension-Point.
- **HttpClient:** zentraler Wrapper um `wp_remote_*`. Eingebaute Retry-Semantik für 429 und 5xx (Honoring `Retry-After`), Multipart-Upload, JSON-Body.
- **PostHookListener:** Hook auf `transition_post_status`. Honoriert `_esa_skip_distribution` und `_esa_selected_accounts`. Idempotent via `hasActiveJob`.
- **DTOs:** `ChannelAccount`, `SendResult` — immutable, mit `fromDbRow` / `toPublicArray` / `success` / `failure`-Factories.
- **Migrations:** `dbDelta`-basiert, idempotent, versioniert über `wp_options['esa_db_version']`.

Wie der Bootstrap die 20 Driver registriert

`Channels\Bootstrap::registerBuiltins` hängt sich an `esa_register_channels` und instanziiert die mitgelieferten Drivers. Vereinfacht:

AKTION	WAS PASSIERT
`do_action('esa_register_channels', $registry)`	Wird vom Registry beim ersten `getInstance()`-Zugriff gefeuert.
`Bootstrap::registerBuiltins`	Default-Listener. Registriert die 20 Built-in-Drivers.
Third-Party-Listener	Eigene Drivers per `add_action('esa_register_channels', fn($r) => $r->register(new MyDriver()))`.

Bedingung für einen Custom-Driver: `extends ChannelDriver`, alle acht abstract methods implementiert, `getId()` returnt einen global eindeutigen Slug. Beim nächsten UI-Reload erscheint die Karte im Channels-Grid.

DB-Schema (kompakt)

TABELLE	WICHTIGSTE SPALTEN
`wp_esa_channels`	`id`, `channel_id` UNIQUE, `is_enabled` TINYINT, `settings` LONGTEXT
`wp_esa_channel_accounts`	`id`, `channel_id`, `remote_user_id`, `remote_username`, `avatar_url`, `credentials_encrypted` LONGTEXT, `scopes`, `status` ENUM(`active`, `expired`, `revoked`)
`wp_esa_queue`	`id`, `post_id`, `channel_id`, `account_id`, `payload`, `status` ENUM(`pending`, `processing`, `done`, `failed`), `attempts`, `priority`, `scheduled_for`, `last_attempt_at`, `last_error`
`wp_esa_jobs`	`id`, `queue_id`, `channel_id`, `account_id`, `remote_post_id`, `remote_post_url`, `response_data` LONGTEXT, `status` ENUM(`success`, `failed`), `executed_at`
`wp_esa_logs`	`id`, `level` ENUM(`debug`, `info`, `warning`, `error`), `context`, `message`, `meta`, `created_at`

REST-Endpoints (Namespace `esa/v1`)

METHODE	PFAD	ZWECK
GET	`/dashboard/stats`	Aggregierte StatTile-Zahlen (Phase-4 neu).
GET	`/channels`	Liste aller registrierten Channel-Driver inkl. `account_count`.
GET	`/channels/{id}/accounts`	Accounts pro Channel.
POST	`/channels/{id}/connect/start`	OAuth-/Token-Connect-Start.
POST	`/channels/{id}/connect/callback`	API-driven Callback (UI nutzt `admin-post.php`).
DELETE	`/accounts/{id}`	Account trennen.
GET	`/queue?status=pending&limit=50`	Queue-Einträge nach Status.
POST	`/queue/{id}/retry`	Job zurück auf `pending`.
DELETE	`/queue/{id}`	Job-Eintrag löschen.
POST	`/cron/run`	Cron sofort triggern (Phase-4 neu).
GET	`/jobs?account_id=&limit=50`	History aus `wp_esa_jobs`.
GET	`/logs?level=&limit=100`	Application-Log-Stream.

Browser-Callback-Endpoint (keine REST-Route, sondern `admin-post.php`):

PFAD	ZWECK
`/wp-admin/admin-post.php?action=esa_oauth_callback&channel={id}`	Stabile Redirect-URI für OAuth-Provider.

Cron

HOOK	SCHEDULE	INTERVAL	WORKER
`esa_process_queue`	`esa_five_minutes`	300 s	`QueueProcessor::run`

Glossar

Kurzdefinitionen der wichtigsten Begriffe aus diesem Handbuch.

BEGRIFF	DEFINITION
OAuth	Autorisierungsprotokoll, das einer Drittanwendung Zugriff auf ein Nutzerkonto gewährt, ohne dass das Passwort weitergegeben wird. EasySocialAutopilot nutzt OAuth-2-Authorization-Code-Flow (LinkedIn, Reddit, Pinterest, WordPress.com, Google-Family, Meta-Family, X) und OAuth-1.0a HMAC-SHA1 (Tumblr).
PKCE	*Proof Key for Code Exchange*. OAuth-2-Erweiterung mit `code_verifier` / `code_challenge`, schützt Public-Clients gegen Authorization-Code-Interception. ESA nutzt PKCE bei X (Twitter).
Confidential vs Public OAuth Client	Confidential Client besitzt ein Client-Secret und kann es geheim halten (Server-Side). Public Client nicht (Mobile, SPA). ESA-Drivers laufen serverseitig — alle sind Confidential.
App-Password	Provider-generiertes Sekundärpasswort, das man Apps gibt, ohne das Hauptpasswort preiszugeben. ESA nutzt App-Password bei Bluesky.
Bot-Token	Pre-shared Secret, das einem programmatischen Account-Identifier zugeordnet ist. ESA: Telegram.
Webhook	HTTPS-Endpoint, der unsolicited eingehende POSTs entgegennimmt. ESA-Webhook-Channel sendet generischen JSON-Payload — Empfänger entscheidet, was er damit tut.
Bearer	HTTP-Authentication-Schema: `Authorization: Bearer <token>`. Der Token-Holder wird unmodifiziert als Auth akzeptiert.
Long-Lived Token	Token mit Wochen- bis Monatslaufzeit, oft per Exchange aus einem Short-Lived Token erzeugt. ESA nutzt z. B. bei Meta und Threads.
Token	Access-Token, das der OAuth-Provider nach erfolgreicher Autorisierung ausstellt. Wird als Bearer-Header an die API mitgesendet.
Scope	Berechtigungsumfang eines Tokens. Pro Channel verschieden, dokumentiert in den jeweiligen READMEs.
App-Review	Plattform-seitiger Verifizierungsprozess (Meta, TikTok), in dem die App-Funktionalität geprüft wird, bevor Production-Scopes freigeschaltet werden. Wochen bis Monate Wartezeit üblich.
AES-256-GCM	Symmetrisches Verschlüsselungsverfahren mit integriertem Authentizitäts-Tag. Standard für moderne Token-At-Rest-Verschlüsselung.
Cron	Geplanter Hintergrund-Job. WordPress kennt einen "WP-Cron" (Request-getrieben) und System-Cron (zeitgesteuert per Server).

BEGRIFF	DEFINITION
Queue	Persistente Tabelle pending Jobs (`wp_esa_queue`), die der Cron-Worker abarbeitet.
Backoff	Wartezeit zwischen Retry-Versuchen, die mit jedem Fehlversuch wächst. Hier: 5 → 15 → 60 → 180 Minuten.
dbDelta	WordPress-Funktion zum idempotenten Anlegen/Migrieren von DB-Tabellen aus CREATE-TABLE-Statements.
Idempotency-Key	HTTP-Header, der dem Server signalisiert, dass identische Requests nur einmal verarbeitet werden sollen. Verhindert Doppel-Posts bei Retries.
Federation	Architektur des Fediverse: viele eigenständige Mastodon-/Pleroma-/Misskey-Instanzen, die über ActivityPub kommunizieren.
AT Protocol	*Authenticated Transfer Protocol.* Föderiertes Protokoll hinter Bluesky. PDS = Personal Data Server, hostet die User-Daten.
Toot / Status	Mastodon-Begriffe für einen einzelnen Beitrag. *Toot* ist die historische, *Status* die aktuelle API-Bezeichnung.
Instance	Eigenständiger Mastodon-Server (z. B. `mastodon.social`, `chaos.social`).
PDS	*Personal Data Server* im AT-Protocol — Bluesky-Pendant zur Mastodon-Instance. Default: `https://bsky.social`.
Transient	Kurzlebige Key-Value-Speicherung in WordPress, hier genutzt für OAuth-State (10 Min TTL).
PSR-4	PHP-Autoload-Standard, der Klassen-Namespaces auf Datei-Pfade abbildet.
HMAC-SHA256	Hashed Message Authentication Code. ESA-Webhook-Channel signiert Payloads optional mit `HMAC-SHA256(secret, body)`.
POSTING_BLOCKED	Marker im Code: OAuth-Connect funktioniert, aber `send()` returnt eine Failure, weil eine externe Freigabe fehlt. Siehe Kapitel 5b.4.

Über dieses Plugin

EasySocialAutopilot ist eine Open-Source-Initiative, getragen von **Uwe Hiltmann** (Internet-Unternehmensberater & KI-Consultant) und Contributors. Ziel: eine WordPress-Social-Distribution-Pipeline, die ohne Phone-Home, ohne Lizenz-Server und ohne SaaS-Abhängigkeit funktioniert.

DETAIL	WERT
Plugin-Version	0.4.0 (Phase 4 — Auto-Enqueue + 20 Channels)
Lizenz	GPL-2.0+
Repository	github.com/uh8888/easysocialautopilot
Text-Domain	`easy-social-autopilot`
Mindest-WordPress	6.0
Mindest-PHP	8.0

Feedback, Bug-Reports und Pull-Requests sind willkommen — nutze dafür den Issue-Tracker im GitHub-Repository.

Stand dieses Handbuchs: Phase 4. Auto-Enqueue produktiv, 20 Channel-Drivers registriert, 11 davon voll funktional posting-fähig, 5 Posting-blockiert pending external approval, 1 description-only (YouTube), 1 token-only (Medium), 1 Platzhalter (XING). Phase 5 bringt das Per-Post-UI im Gutenberg-Editor.