

ExtensionReach

Lead-Discovery, Ansprache und Content für ein Trading-Journal, gebaut als Vollsistem

ExtensionReach ist die Wachstumsmaschine für Entrio, ein KI-gestütztes Trading-Journal. Das System findet passende Trader über sechs Quellen, recherchiert jeden einzelnen, schlägt eine Nachricht vor, begleitet das Gespräch bis zur Beta-Anmeldung und hält die Social-Kanäle mit faktengeprüftem Content versorgt. Vier Laufzeiten teilen sich eine Datenbank: eine React-Web-App, zwei Chrome-Extensions, fünfzehn Edge Functions und acht geplante Workflows. Konzept, Architektur, Umsetzung und Betrieb in einer Hand.

36'644

Zeilen TypeScript, JavaScript und SQL

15

Edge Functions auf Deno

34 / 38

Tabellen und Migrationen in PostgreSQL

3

KI-Agenten mit zusammen 31 Tools

2

Chrome-Extensions nach Manifest V3

8

geplante Workflows auf GitHub Actions

Gabriel Suter · Einzelumsetzung · Konzept, Architektur, Frontend, Backend, Extensions, KI, Betrieb

React 18 · TypeScript · Supabase · Deno · Chrome MV3 · OpenAI · Claude auf AWS Bedrock · Januar 2026 bis heute

kontakt@gabrielsuter.dev · gabrielsuter.dev · Stand September 2026

1 Kurzfassung

Ein frühes Produkt braucht seine ersten aktiven Nutzer, und die sitzen verstreut auf LinkedIn, in Substack-Newslettern, auf Medium, TradingView, Reddit, Discord und in Foren. ExtensionReach macht daraus einen Ablauf: finden, recherchieren, Nachricht vorbereiten, freigeben, Antwort erfassen, Deal führen. Dazu kommt eine Content-Pipeline, die zweimal pro Woche faktengeprüfte Beiträge samt Slides erzeugt. Getragen wird das von einer React-Web-App, zwei Chrome-Extensions, fünfzehn Edge Functions und acht Workflows auf einer gemeinsamen PostgreSQL-Datenbank; drei Agenten mit Function Calling arbeiten auf denselben Daten, einer streamt Claude über einen abgesicherten Bedrock-Proxy.

Der Kern in einem Satz

Die Agenten dürfen alles vorbereiten und nichts absenden: Jede Nachricht an einen echten Menschen braucht eine Freigabe in der Oberfläche, jede erzeugte Behauptung einen zweiten, unabhängigen Prüfschritt, bevor sie gespeichert wird.

2 Ausgangslage und Ziel

ExtensionReach ist eines von fünf Projekten in einem Produkt-Ökosystem. Entrio ist die Daten- und Strukturebene: ein Trading-Journal, in dem ein Screenshot hochgeladen wird und die KI Symbol, Einstieg, Ausstieg, Stop und Risiko ausliest, dazu psychologische Daten pro Trade. Stravesta ist die geplante Intelligenzebene darüber, ExtensionReach die Wachstumsmaschine davor: Es findet die Menschen, für die Entrio gebaut ist.

Das Problem war Handarbeit: suchen, Profile lesen, Nachrichten schreiben, die nicht nach Spam klingen, Nachfassen nicht vergessen und nebenher Social-Content produzieren. Daraus wurde eine Pipeline mit dem Menschen an den Stellen, an denen Urteil nötig ist: Lead freigeben, Nachricht freigeben, Deal-Stufe entscheiden.

Die Zielgruppen liegen als Personas in der gemeinsamen Wissensbasis und gehen direkt in die Prompts ein: der Forex-Daytrader auf dem Weg zur Prop-Firm-Challenge, der Krypto-Trader, der Swing-Trader und der Anfänger, jeder mit seinen Schmerzpunkten; die Suchprofile decken sechs Länder ab. Der North Star lautet «50 aktive Nutzer in 90 Tagen» (ADR-011, Februar 2026); jede Idee im Backlog wird daran gemessen, ob sie zu Akquisition oder Aktivierung beiträgt. Die Ansprache auf LinkedIn läuft manuell und in menschlichem Tempo; die Werkzeuge unterstützen und erfassen.

Vier Leitprinzipien, im Code sichtbar

- **Mensch in der Schleife.** Nichts geht ohne Freigabe an eine reale Person. Agenten dürfen vorbereiten, klassifizieren und vorschlagen; schreibende Aktionen auf die Ansprache sind gesperrt.
- **Wahrhaftigkeit vor Eloquenz.** Die Prompts verbieten erfundene Bezüge, sind auf die tatsächlichen Produktfunktionen geerdet, und vor dem Speichern läuft eine Faktenprüfung.
- **Alles hat einen Schalter und ein Limit.** Ein Hauptschalter, zehn Funktionsschalter, Tageslimits und Verzögerungsbereiche in einer Settings-Tabelle, damit sich Verhalten ohne Deployment ändern lässt.
- **Günstig im Betrieb.** Freie Kontingente, wo möglich, manuelle Auslösung für bezahlte APIs, Rendering auf freien CI-Minuten.

Meine Rolle

- **Produkt und Konzept:** Positionierung, Personas, Priorisierung gegen eine Kennzahl, Entscheide als Architecture Decision Records.
- **Frontend:** React 18 mit TypeScript, 17 Feature-Seiten, TanStack Query, Realtime, Streaming-Chat, Stufenfilter und Datentabellen, PWA.
- **Backend:** Schema über 38 Migrationen, Row-Level-Security-Policies, 15 Edge Functions in Deno, Webhook-Empfänger, asynchrone Jobs.
- **Extensions:** zwei Chrome-Extensions nach Manifest V3 in reinem JavaScript, Service Worker, Side-Panel-API, Shadow-DOM-Suche.
- **KI:** drei Agenten auf zwei Anbietern, agentische Schleifen, SSE-Streaming, Bildeingabe, Prompt-Erdung, Faktenprüfung, bewertete Entwürfe.
- **Betrieb:** acht GitHub-Actions-Workflows, Playwright-Rendering auf CI, Deployments, Secrets-Verwaltung, Telegram-Meldungen.

Meilensteine

Wann	Was
31. Januar 2026	MVP: Web-App mit Supabase-Schema, Entwurfsgenerierung über GPT, Chrome-Extension mit Popup und Seitenpanel.
4. bis 18. Februar 2026	Tägliche Lead-Discovery über Edge Function und GitHub Action, Review-Ablauf, Community Finder, Anreicherung.
19. bis 27. Februar 2026	TradingView, Feedback Hub, Substack, Deal Flow und Deal-Tracker-Extension, Claudia und Ale, PWA, Supabase Auth, Carla-Proxy auf Bedrock.
4. bis 7. März 2026	Länderfilter, automatisierte Content-Pipeline, Bildgenerierung mit zwei Anbieterwechseln, Erdung der Prompts.
Ende März 2026	Carla mit Streaming, LinkedIn-Posteingang mit Bulk-Import und Live-Erfassung, Tool-Gleichstand beider Oberflächen.
10. Mai 2026	Rendering auf Playwright in GitHub Actions, Telegram-Meldungen, Fabrication Guard, Node-22-Runner.

Seither laufend: Apify-Discovery und -Anreicherung, Discord-Community-Tracking, Outreach-Kalender für Reddit und Discord, X als Content-Typ, Workshop-Board.

3 Funktionsumfang

Lead-Discovery aus sechs Quellen

Leads kommen über sechs Kanäle ins System, jeder mit eigenem Zeitplan, Quellen-Tag und Qualitätsfilter; die Filterung nach Quelle ist pro Discovery-Seite erzwungen.

Quelle	Mechanismus	Zeitplan	Besonderheit
LinkedIn (RapidAPI)	Suchprofile mit Keyword-Rotation, Geo-Codes, Analyse von Headline und About-Text	täglich, Edge Function über GitHub Action	Qualitätsscore 0 bis 100 (Headline 40, About 30, URL 10, Basis 20), Mindestwert 35, 25 Leads pro Tag
LinkedIn (Apify)	Google-Search-Actor mit <code>site:linkedin.com/in</code> , Ergebnisse per signiertem Webhook	täglich, über Schalter	Ein zentraler <code>apify-webhook</code> verteilt Discovery, Anreicherung und Post-Scraping
Substack	Autorensuche in Finance und Technology mit fünf Keyword-Presets	monatlich	400 Autoren, rund ein Drittel trading-relevant
Medium (RapidAPI)	Autorensuche über zehn Trading-Themen	nur manuell	189 Autoren im freien Kontingent
TradingView	zwölf symbolbezogene RSS-Feeds, Profilabruf, Social-Links aus gerenderten Daten	GitHub Action	208 Trader gescannt, 46 mit Social-Links, Filter gegen Signalverkäufer
Reddit und Communities	Node-Skripte und Edge-Function-Scraper über Subreddits und Foren	wöchentlich und viermal täglich	Beta-Tester-Kandidaten mit erkennbarer Absicht

Anreicherung mit Beta Fit Score

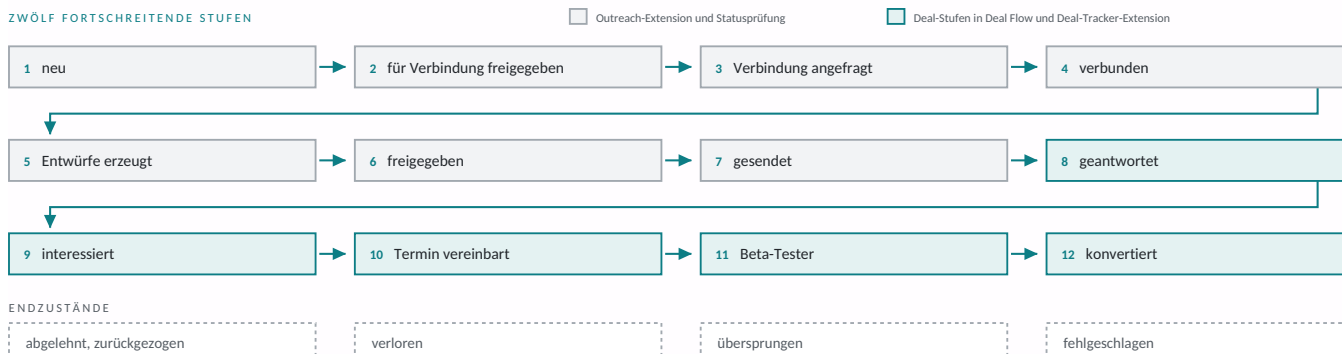
Jeder aussichtsreiche Lead bekommt vor der ersten Zeile Text einen Rechschritt. Perplexity Sonar Pro recherchiert Social-Profile, Handelsstil, Märkte, Werkzeuge und Communities und extrahiert Beta-Signale: wird bereits ein Journal genutzt, welches, werden Schmerzpunkte genannt, sucht die Person aktiv nach einer Lösung. Daraus berechnet sich deterministisch ein Beta Fit Score von 0 bis 100, farbcodiert in der Review-Oberfläche.

Entwürfe mit Bewertungsraster

GPT-4.1 erzeugt pro Lead drei Nachrichtenvarianten für eine von vier Absichten, auf Englisch oder in deutschem Du. Der System-Prompt verbietet erfundene Bezüge und legt den Recherchekontext bei, damit jedes persönliche Detail auf eine Quelle zurückführbar ist. Bewertet wird auf einem gewichteten Raster von 0.0 bis 1.0: Treffsicherheit der Bezüge 40 Prozent, Antwortleichtigkeit 30 Prozent, Tonalität 20 Prozent, Antwortwahrscheinlichkeit 10 Prozent.

Review und Pipeline

Die Seite Batch Review zeigt jeden Lead mit seinen Entwürfen auf einem Schirm: aufklappbare Recherche, editierbarer Text, Score-Badges, Freigeben oder Überspringen, dazu Sammelfreigabe und Sammelversand. Eine Freigabe legt einen Nachrichtendatensatz an und plant einen Job für die Extension; die Pipeline-Seite zeigt denselben Bestand nach Stufen gefiltert, mit Massenaktionen.



Der Lebenszyklus eines Leads. Die Stufen bis «gesendet» führen die Outreach-Extension und die Statusprüfung, die hervorgehobenen Deal-Stufen die Deal-Flow-Seite und die Deal-Tracker-Extension.

Antwortklassifikation und Deal Flow

Eingehende Antworten werden aus dem LinkedIn-Posteingang erfasst oder manuell nachgetragen, auch aus Screenshots, die GPT-4.1 Vision in strukturierten Text überführt. Jede Antwort wird in sieben Kategorien eingeordnet, von positiv über beschäftigt und Einwand bis «kein Trader», und auf Signale hin gelesen: Journaling-Gewohnheit, Werkzeug, Schmerzpunkt, Handelsstil. Der Klassifikator schlägt eine Deal-Stufe vor, aus dem Verlauf entstehen drei Nachfass-Entwürfe, und pro Deal laufen Notizen, nächste Schritte und Nachfassdaten mit.

Cold Email

Für Leads ausserhalb von LinkedIn reichert Apollo.io E-Mail, Firma und Position aus einer LinkedIn-URL an. GPT-4.1 schreibt daraus eine vierstufige Sequenz, also Aufmacher, Nachfassen, Sozialnachweis und weicher Abschluss, personalisiert mit dem Quellkontext; eine Oberfläche mit vier Reitern endet in einer CSV im Format von Smartlead.

Community-Intelligence

Eine eigene Edge Function durchsucht elf Plattformen nach Trading-Communities: Reddit, TradingView, Futures.io, Forex Factory, BabyPips, MyFXBook, Telegram und vier Discord-Verzeichnisse. Jede Community erhält einen Relevanzwert von 0 bis 100 aus Keyword-Stufen, Mitgliederzahlen, Aktivität, Sprache und Spam-Indikatoren; ein Rate Limiter pro Plattform hält Anfragen pro Minute und Mindestabstände ein. Derselbe Lauf zieht Reddit-Nutzer mit Beta-Tester-Absicht heraus, Wettbewerbernennungen samt Stimmung, Content-Ideen und Produkterkenntnisse. Ergänzend schieben F5Bot und n8n Keyword-Alarme aus Reddit, Hacker News und Lobsters in eine Webhook-Edge-Function, zu der GPT-4o-mini eine Antwort vorschlägt. Im Feedback Hub transkribiert Whisper Aufnahmen, GPT-4o-mini zieht Zusammenfassung, Stimmung und Massnahmen heraus.

Content-Pipeline

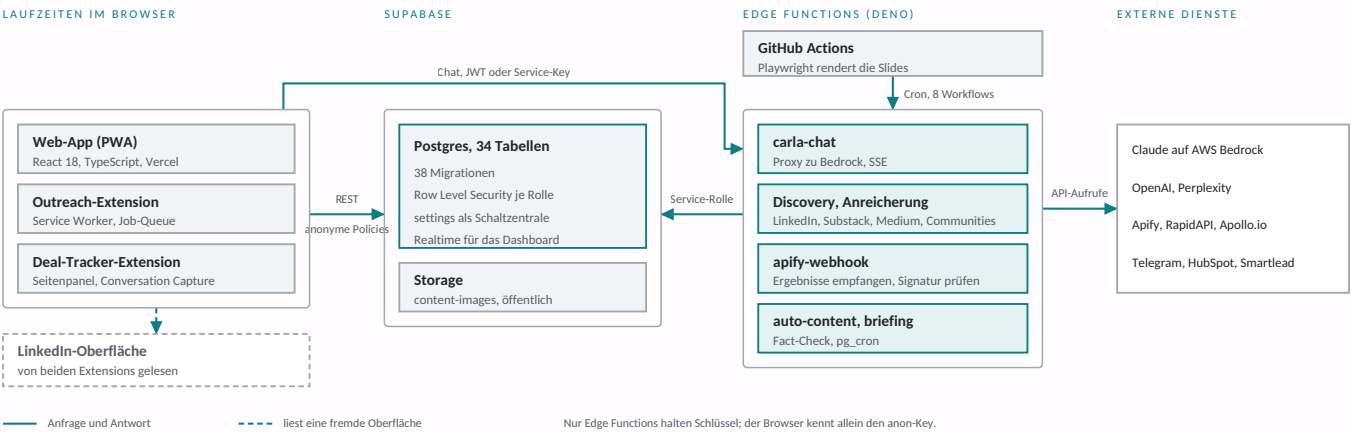
Zweimal pro Woche erzeugt `auto-content` Beiträge für LinkedIn und Instagram ohne manuelle Eingabe. Sechs Schritte: Perplexity recherchiert, worüber Trader diese Woche sprechen; GPT-4.1 schreibt zwei Ideen mit je 6 bis 8 Slides, geerdet auf reale Entrio-Funktionen; Perplexity prüft Zahlen und Behauptungen gegen, flankiert vom Fabrication Guard; aus gemeinsamen Brand-Tokens entstehen Slides im Format 1080 × 1350 px als HTML, die Playwright auf dem GitHub-Runner fotografiert; zuletzt wandern die PNG in den Storage und Telegram erhält eine Zusammenfassung.

Dashboard, Briefing und HubSpot

Das Dashboard zeigt Tageskennzahlen, Pipeline-Status und einen Live-Aktivitätsstrom über Supabase Realtime; die Einstellungen führen API-Schlüssel, Schalter und Limits. Ein tägliches Morning Briefing läuft aus `pg_cron` und schreibt Pipeline-Zahlen, überfällige Deals und jüngste Antworten in den Carla-Chat. Die HubSpot-Anbindung ist optional: Bei einer Freigabe entsteht ein Kontakt mit der Nachricht als Notiz, bei einer Antwort werden Klassifikation und Signale übertragen.

4 Architektur

Vier Laufzeiten teilen sich ein einziges Supabase-Projekt. Web-App und Extensions sprechen PostgreSQL über REST und Realtime an; die Edge Functions tragen alles, was serverseitig laufen muss, also Geheimnisse, lange Recherchen und Webhooks; GitHub Actions liefert Zeitsteuerung und einen kopflosen Browser fürs Rendern.



Vier Laufzeiten, eine Datenbank. Browser und Edge Functions treffen sich in Supabase: der Browser über REST mit eng gefassten anonymen Policies, die Edge Functions mit der Service-Rolle. Nur die Edge Functions sprechen mit den externen Diensten und halten deren Schlüssel; der Chat läuft über den Proxy carla-chat, die Zeitsteuerung über GitHub Actions.

Technologie

Bereich	Technologie
Frontend	React 18.3, TypeScript 5.8, Vite 5, React Router 6, TanStack React Query 5, React Hook Form mit Zod, Tailwind CSS 3, shadcn/ui auf Radix UI, Recharts, vite-plugin-pwa
Backend	Supabase: PostgreSQL mit Row Level Security, Auth, Storage, Realtime, Edge Functions auf Deno, Zeitsteuerung über <code>pg_cron</code>
Extensions	Chrome Manifest V3, Service Worker, <code>chrome.alarms</code> , <code>chrome.scripting</code> , Side-Panel-API, Content Scripts, MutationObserver, Shadow-DOM-Suche
KI	OpenAI GPT-4.1 mit Function Calling und Vision, GPT-4o-mini, Whisper; Claude Sonnet 4.6 über AWS Bedrock; Perplexity Sonar Pro
Datenbeschaffung	RapidAPI für LinkedIn und Medium, Apify-Actors mit Webhooks, Apollo.io, RSS-Feeds, eigene HTML-Scraper, F5Bot- und n8n-Webhooks
Automatisierung	GitHub Actions mit acht Workflows, Playwright-Rendering auf CI, Telegram Bot API, Vercel-Deployment
Integrationen	HubSpot CRM, CSV-Export für Smartlead, Telegram

Bereich	Technologie
Werkzeuge	ESLint 9 mit typescript-eslint, Vitest mit Testing Library und jsdom, Supabase CLI, Bun und npm

Verantwortlichkeiten

Komponente	Verantwortet	Spricht mit
Web-App	alle Review- und Entscheidungsoberflächen, Dashboards mit Realtime, Agenten-Chats, Content-Kalender, Einstellungen, clientseitige KI-Aufrufe	Supabase, OpenAI, Perplexity, HubSpot
Outreach-Extension	Erfassung und Statusprüfung auf LinkedIn, Abarbeitung der Job-Queue, Popup und Seitenpanel für manuelle Entwürfe	Supabase REST, LinkedIn-DOM
Deal-Tracker-Extension	der Ablauf nach der Antwort: Lead-Erkennung über die URL, Conversation Capture, Stufenwechsel, Nachfass-Entwürfe, Carla mit clientseitigen Tools	Supabase REST, <code>carla-chat</code>
Edge Functions	alles mit Geheimnissen oder langer Laufzeit: Discovery, Anreicherung, Webhooks, Content, Briefing, Bedrock-Proxy, Audio	PostgreSQL mit Service-Rolle, externe APIs
GitHub Actions	Zeitsteuerung, Auswertung der Ergebnisse, kopfloses Rendern mit Playwright, Fehlermeldungen	Edge Functions, Storage, Telegram

Die drei Agenten

Agent	Modell und Laufzeit	Zweck und Tools
Carla Outreach	Claude Sonnet 4.6 über AWS Bedrock mit Cross-Region-Inference-Profile, SSE-Streaming, Bildeingabe. In der Web-App und im Deal-Tracker-Seitenpanel.	15 Tools. Lesend: Pipeline-Überblick, heutige Aktivität, überfälliges Nachfassen, Deal-Zusammenfassung, Tagesbriefing, Lead-Details und -Suche, LinkedIn-Nachrichten, Lead-Aktivität, Firmenkontext. Schreibend: Nachricht erzeugen, Status und Nachfassdatum setzen, Entwurf speichern, Lead aus einem Screenshot erfassen. Der System-Prompt verlangt, den echten Verlauf zu lesen, bevor eine Antwort vorgeschlagen wird.
Claudia Pipeline	GPT-4.1 mit OpenAI Function Calling, Temperatur 0.7, bis zu fünf Züge.	7 ausschliesslich lesende Tools über die Pipeline, erzeugt beim Laden ein Briefing. Vorläufer von Carla: Der Dienst bleibt im Code, die Route führt heute auf Carla.
Ale Content	Zwei Backends: Perplexity Sonar Pro für Recherche, GPT-4.1 fürs Schreiben, Temperatur 0.8, bis zu zehn Züge.	9 Tools: Trends recherchieren, Wissensbasis laden, Lead-Erkenntnisse holen, Content erzeugen, Bild-Prompt und Bild erzeugen, Reddit-Kommentare erzeugen, Kalender schreiben und lesen. Führt den Ablauf von der Idee bis zum Veröffentlichen über sieben Kanäle.

So arbeitet die Carla-Runtime

- Der Client baut eine Anfrage an die Claude Messages API mit System-Prompt, Verlauf, Tool-Schemata und optionalen Bildblöcken und schickt sie an `carla-chat`.
- Die Funktion authentifiziert mit einem Supabase-JWT für die Web-App oder mit einem Service-Key-Header für die Extension ohne Benutzersitzung, leitet mit dem AWS SDK an Bedrock weiter und übersetzt den Ereignisstrom in Server-Sent Events.

3. Der Client liest die SSE-Deltas und schreibt den Text live in die Chat-Blase. Ist ein `tool_use`-Block vollständig, läuft das Tool lokal gegen Supabase, sein Ergebnis geht als `tool_result` zurück.
4. Die Schleife läuft, bis das Modell keine Tools mehr aufruft oder die Zugbegrenzung greift. Jede Nachricht wird gespeichert, so überlebt der Verlauf ein Neuladen und ist zwischen beiden Oberflächen geteilt.

Zwei Entwurfsentscheide tragen das: **dünnere Proxy, dicker Client**, die Tools laufen im Client und die Edge Function trägt keine Fachlogik; und **zwei Tool-Sätze, ein Vertrag**, beide Oberflächen führen identische Tool-Namen und -Schemata.

Edge Functions

Function	Aufgabe
<code>daily-discovery</code>	Führt die Suchprofile gegen die LinkedIn-API, bewertet, dedupliziert, protokolliert den Lauf.
<code>check-connections</code>	Zweimal täglich: löst offene Verbindungen auf, markiert angenommen oder nach sieben Tagen abgelehnt.
<code>enrich-lead</code>	Startet parallele Apify-Läufe für Profil und Beiträge, überspringt frisch angereicherte Leads.
<code>apify-webhook</code>	Zentraler Empfänger für Apify-Ereignisse, holt die Datensätze und verteilt sie.
<code>enrich-emails</code>	Apollo.io-Anreicherung für die E-Mail-Ansprache.
<code>substack-discovery</code> , <code>medium-discovery</code>	Autorensuche mit Keyword-Rotation und Stapelverarbeitung.
<code>discover-communities</code>	Elf Scraper mit Rate Limiter und Bewertung, extrahiert Leads und Wettbewerber.
<code>community-alert-webhook</code> , <code>generate-response</code>	Nimmt F5Bot- und n8n-Alarme entgegen, erzeugt einen Antwortvorschlag mit GPT-4o-mini.
<code>process-feedback-audio</code>	Whisper-Transkription und strukturierte Extraktion von Feedback.
<code>carla-chat</code>	Authentifizierter Proxy zu Claude auf AWS Bedrock mit SSE-Streaming.
<code>auto-content</code> , <code>cleanup-calendar</code>	Content-Erzeugung mit Fact-Check und Slide-HTML, Kalenderpflege und Neurendern.
<code>morning-briefing</code>	Täglicher <code>pg_cron</code> -Lauf, schreibt ein Briefing in die Carla-Konversation.

Geteilte Module unter `shared/` stellen den Telegram-Helfer und die Brand-Tokens bereit, mit denen Edge Function, Web-App und CI-Renderer dieselben Slides erzeugen.

Geplante Workflows

Workflow	Zeitplan (UTC)	Aufgabe
Daily Lead Discovery	täglich 07:00	ruft die LinkedIn-Discovery auf, meldet die Zahl neuer Leads
Check Connection Status	täglich 08:00 und 20:00	löst offene Einladungen auf
Auto Content Generation	Montag und Donnerstag 07:00	ruft <code>auto-content</code> , rendert die Slides mit Playwright auf einem Node-22-Runner, lädt die PNG in den Storage

Workflow	Zeitplan (UTC)	Aufgabe
Weekly Community Discovery	Montag 08:00	startet die elf Scraper, prüft vorab die Secrets
Community Research	viermal täglich	Reddit-Recherche nach Beiträgen mit hoher Absicht
Reddit Community Discovery	Dienstag 09:00	Scan von Subreddits und Communities
Substack Lead Discovery	1. des Monats 08:00	monatliche Autorensuche in Stapeln
Medium Lead Discovery	nur manuell	Cron bewusst deaktiviert, um das RapidAPI-Kontingent zu schonen

Sicherheit und Zugriff

Thema	Massnahme
Authentifizierung	Supabase Auth mit E-Mail und Passwort, beschränkt auf eine Allowlist aus einer Umgebungsvariable; eine geleakte URL kann keine Konten anlegen.
Autorisierung	Row Level Security auf jeder Tabelle: authentifizierte Policies für die Web-App, anonyme nur für die Tabellen der Extensions.
LLM-Proxy	Der Bedrock-Proxy akzeptiert ein verifiziertes Supabase-JWT oder einen Service-Key-Header. AWS-Zugangsdaten verlassen die Edge Function nie.
Geheimnisse	Fremdschlüssel in Supabase-Secrets, GitHub-Secrets oder der Settings-Tabelle; Extension-Konfigurationen sind gitignoriert, eingecheckt sind nur Beispielvorlagen.
Webhooks	Der Apify-Webhook prüft ein geteiltes Geheimnis, bevor er die Datenbank berührt.
Fremddienste	CRM-Abgleich und Meldungen fallen weich aus: Sie protokollieren eine Warnung und blockieren den Hauptablauf nie.

Bewusst schlank gehalten

Kein bezahlter Rendering-Dienst, kein eigener Server, keine geplanten Aufrufe kostenpflichtiger Endpunkte. Die Regel lautet: Was Geld kostet, wird von Hand ausgelöst, deshalb ist der Medium-Zeitplan auskommentiert. Meldungen laufen über einen geteilten Telegram-Helfer im HTML-Parse-Mode, weil Markdown an Unterstrichen in Bezeichnern zerbrach.

5 Technische Kernprobleme

Sieben Stellen, an denen das Projekt Untersuchung statt einer Bibliothek verlangte.

AGENTEN-LAUFZEIT

Ein Agent, zwei Laufzeiten, keine Zugangsdaten im Client

Problem

Derselbe Agent sollte in der React-Web-App und im Vanilla-JS-Seitenpanel laufen. Ein Service Worker hat aber keine Benutzersitzung, und AWS-Zugangsdaten dürfen weder im Browser noch im Erweiterungspaket liegen. Dazu kam, dass Claude über die direkten Modell-IDs auf Bedrock nicht erreichbar war.

Lösung

`carla-chat` ist ein dünner Proxy: Er authentifiziert über ein verifiziertes Supabase-JWT oder einen Service-Key-Header, leitet an Bedrock weiter und übersetzt den Ereignisstrom in Server-Sent Events. Die Tools laufen im Client, der Proxy trägt keine Fachlogik. Erreichbar wurde das Modell erst über die ID des Cross-Region-Inference-Profiles, und die JWT-Prüfung musste auf die sitzungsgebundene Benutzerabfrage umgestellt werden.

Warum es zählt

Ein Dienst, der nur Identität prüft und weiterreicht, bleibt austauschbar. Wer Geheimnisse an einer Stelle hält, muss nur eine absichern; wer die Fachlogik draussen lässt, wechselt den Anbieter, ohne zwei Laufzeiten anzufassen.

RENDERING

Zweimal den Anbieter verloren, am Ende bei null Kosten

Problem

Die Content-Pipeline braucht fertige Bildslides. Das freie Kontingent von Gemini Imagen fiel aus, `htmlcsstoimage` deckelte bei 50 Bildern pro Monat: Beide Male stand die Pipeline still, weil die Darstellung beim Anbieter lag.

Lösung

Die Slides sind seither eigenes HTML mit gemeinsamen Brand-Tokens, gerendert mit Playwright auf dem GitHub-Runner, pixelgleich und auf freien CI-Minuten. Dasselbe CSS teilen Edge Function, Web-App und Renderer, sodass Vorschau und Ausgabe nicht auseinanderlaufen.

Warum es zählt

Ein Format, das man selbst besitzt, überlebt den Anbieter. Markup und CSS sind portabel, ein Bild-Endpunkt ist es nicht; wer die Darstellung im eigenen Quelltext hält, tauscht den Renderer aus statt das Produkt.

ZUGRIFFSKONTROLLE

Row Level Security, in Stufen gewachsen

Problem

Das erste Schema erlaubte anonymen Zugriff, weil es ein Prototyp für eine Person war. Mit Supabase Auth war das nicht mehr tragbar, also stellte Migration 026 sämtliche Policies auf «nur authentifiziert» um. Danach waren die Extensions ausgesperrt: Ein Service Worker hat keine Benutzersitzung, die er vorweisen könnte.

Lösung

Migration 029 gibt anonymen Zugriff gezielt zurück, für genau die Tabellen der Extensions, und begründet das im Migrationskommentar. Nummerierte Migrationen machen jede Stufe sichtbar.

Warum es zählt

Berechtigungen werden nicht einmal gesetzt, sie entwickeln sich mit dem Produkt. Eine Ausnahme mit Begründung in der Migration ist eine Entscheidung; dieselbe Ausnahme ohne Begründung ist ein Fund im nächsten Audit, den niemand mehr einordnen kann.

FREMDE OBERFLÄCHE

LinkedIn ignoriert programmatische Klicks

Problem

LinkedIn ist eine React-Single-Page-App mit gehashten Klassennamen. Ein `element.click()` löst dort nichts aus: Die Handler hängen an Pointer- und Mausereignissen mit echten Koordinaten.

Lösung

Die Extension sendet die vollständige Ereignisfolge aus `pointerdown`, `mousedown`, `pointerup`, `mouseup` und `click` mit den Koordinaten des Zielelements. Namen und Headline liest ein TreeWalker innerhalb von `<main>`. Schaltflächen ohne Text erkennt sie über `aria-label`, in Deutsch und Englisch, mit Wortgrenzen, damit «500+ connections» nicht als «connect» gelesen wird.

Warum es zählt

Eine fremde Oberfläche ist keine Schnittstelle. Wer sie liest, muss ihr Ereignismodell verstehen, statt der Abkürzung zu vertrauen, die der Browser anbietet. Und Erkennung über semantische Attribute hält länger als über Klassennamen.

LEBENSZYKLUS

Zustand, den es nur in einer Ausführung gibt

Problem

Der Inhalt eines Dropdowns liess sich nicht zuverlässig auslesen: Zwei Skript-Einspritzungen, eine zum Öffnen und eine zum Lesen, verlieren den Zustand dazwischen. Und LinkedIn rendert Menüs nur, wenn der Tab aktiv ist.

Lösung

Eine einzige `executeScript`-Ausführung klickt, wartet innerhalb der Seite über ein Promise auf die Einträge und gibt sie in einem Zug zurück. Der Tab wird vorher aktiviert, das Ergebnis verifiziert statt angenommen.

Warum es zählt

Die Grenze zwischen zwei Ausführungen ist eine echte Grenze. Wer sie übersieht, sucht den Fehler in der eigenen Logik, obwohl er im Lebenszyklus liegt; flüchtiger Zustand muss dort gelesen werden, wo er entsteht.

NEBENLÄUFIGKEIT**Zeitlimits und Ergebnisse, die später kommen****Problem**

Die Anreicherung mehrerer Leads lief in das Zeitlimit der Edge Function, und der externe Dienst antwortet, wann er will, nicht innerhalb einer Anfrage.

Lösung

Der Lauf wurde in fünf Stapel zu zehn Leads geteilt. Die Apify-Anreicherung startet und antwortet sofort; das Ergebnis kommt später über einen Webhook, der den Lauf anhand der Run-ID zuordnet und richtig einsortiert. Der Zwischenstand steht in `apify_runs`.

Warum es zählt

Sobald eine fremde Laufzeit im Spiel ist, ist die synchrone Antwort eine Annahme und keine Eigenschaft. Wer den Auftrag von seinem Ergebnis trennt und den Zwischenstand in der Datenbank hält, überlebt jedes Zeitlimit und kann nach einem Absturz dort weitermachen, wo der Vorgang stand.

WAHRHAFTIGKEIT**Erzeugter Content, der sich vom Produkt entfernt****Problem**

Automatisch erzeugte Beiträge beschrieben Funktionen, die es nicht gibt, und führten Statistiken ohne Beleg. Beides wäre öffentlich geworden, im Namen des Produkts.

Lösung

Die Prompts sind auf die tatsächliche Funktionsliste geerdet, ein Fabrication Guard weist erfundene Produktmerkmale zurück, und vor dem Speichern prüft ein eigener Perplexity-Durchgang Zahlen und Behauptungen nach. Auto-Content ist auf zwei Plattformen begrenzt.

Warum es zählt

Ein Sprachmodell erzeugt Plausibles, nicht Wahres. Die Prüfung gehört deshalb nicht in denselben Aufruf, der die Ausgabe erzeugt, sondern in einen zweiten, unabhängigen Schritt davor. Vor der Veröffentlichung kostet ein Fehlalarm einen Durchlauf; danach kostet er Glaubwürdigkeit.

Weitere gelöste Probleme

Problem	Lösung
Audiodateien über dem Whisper-Limit	Clientseitige Zerlegung in WAV-Abschnitte von zwölf Minuten, sequenzielle Transkription, zusammengefügtes Transkript.
Kodierte LinkedIn-Mitglieds-IDs	Rund zehn Prozent der Member-URNs sind veraltet. Vanity-URLs haben Vorrang, ein 404 gilt als fehlgeschlagener Job statt als Wiederholungsgrund.
Typografie in gerenderten Slides	Lange Gedankenstriche zerbrachen das Zeilenlayout. Ein geteilter Reinigungsschritt normalisiert die Interpunktion vor Rendering und Anzeige.
Scraper an den Plattformgrenzen	Ein Rate Limiter pro Plattform mit Budget an Anfragen pro Minute und Mindestabstand, je Seite eingestellt.

Problem	Lösung
Schaltfläche im Einladungsdialog nicht auffindbar	Sie liegt in einem Shadow-DOM. Eine rekursive Suche über alle Shadow-Roots und Frames mit acht Wiederholungen findet sie, danach bestätigt eine Verifikationsphase den Zustand «Pending».

6 Datenmodell

Das Schema wuchs über 38 vorwärtsgerichtete Migrationen, jede mit einem Kommentar zur Absicht; die 34 Tabellen fallen in fünf Gruppen.

Gruppe	Tabellen	Besonderheit
Outreach-Kern	<code>leads</code> , <code>drafts</code> , <code>messages</code> , <code>conversations</code> , <code>daily_batches</code> , <code>connection_jobs</code> , <code>search_profiles</code> , <code>discovery_runs</code> , <code>settings</code>	Ein Lead trägt Status, Quelle, Qualitätsscore, Web-Recherche als JSON, Beta Fit Score und Deal-Felder.
Anreicherung	<code>apify_runs</code> , <code>linkedin_profiles</code> , <code>linkedin_posts</code> , <code>linkedin_companies</code> , <code>linkedin_connections</code> , <code>linkedin_campaigns</code> , <code>linkedin_checklist</code>	Apify-Läufe werden nach Zweck geführt und vom Webhook aufgelöst.
Community-Intelligence	<code>communities</code> , <code>reddit_leads</code> , <code>reddit_research_posts</code> , <code>competitor_mentions</code> , <code>content_ideas</code> , <code>integration_requests</code> , <code>feature_votes</code> , <code>product_insights</code> , <code>community_alerts</code> , <code>response_templates</code> , <code>workshops</code>	Relevanz- und Engagement-Bewertung, Vorlagen mit UTM-Verfolgung.
Discovery-Quellen	<code>tv_traders</code> , <code>email_drafts</code>	TradingView-Profil mit Social-Links, vierstufige E-Mail-Sequenzen.
Agenten und Content	<code>agent_conversations</code> , <code>agent_messages</code> , <code>content_calendar</code> , <code>knowledge_documents</code> , <code>feedback_entries</code>	Geteilter Chat-Verlauf; der Kalender führt Plattform, Content-Typ und Bild-URLs.

Die Settings-Tabelle als Schaltzentrale

`settings` ist ein Schlüssel-Wert-Speicher und trägt drei Dinge zugleich: API-Schlüssel; die Schalter, also einen Hauptschalter und zehn Funktionsschalter; und die Limits von Leads und Sendungen pro Tag bis zum Prüfintervall. Weil alle vier Laufzeiten dieselbe Tabelle lesen, ändert sich Verhalten ohne Deployment.

Dazu kommt der Storage-Bucket `content-images`: öffentlich lesbar für die gerenderten Slides, Schreiben nur authentifiziert.

7 Qualität und offene Punkte

Massnahme	Was sie leistet
Typisierung	TypeScript durchgehend mit typisierten Datenbankmodellen, ESLint 9 mit typescript-eslint und React-Hooks-Regeln.
Schichtung	Ein Service-Modul pro Fremdsystem oder Fachthema; geteilte Module statt Kopien.
Architecture Decision Records	Jedes grössere Vorhaben hat ein ADR mit geprüften Optionen und Begründung, über alle fünf Projekte hinweg.
Migrations-Kommentare	Migrationen und Edge Functions beginnen mit Auslöser, Eingaben und Ausgaben.
Notizen pro Projekt	Meilensteine, offene Themen, Bewertung der Lead-Quellen, auch der verworfenen wie MQL5 hinter Cloudflare.
Git-Hygiene	Sprechende Commit-Nachrichten im Imperativ, Feature-Branches, nummerierte Migrationen.

Zum Arbeitsweg

AI-augmented Workflow: Der Autor nutzt KI-Coding-Assistenten für Exploration und Umsetzung und verantwortet dabei Architektur, Review, Test in der echten LinkedIn-Umgebung und alle Entscheidungen selbst.

Offen

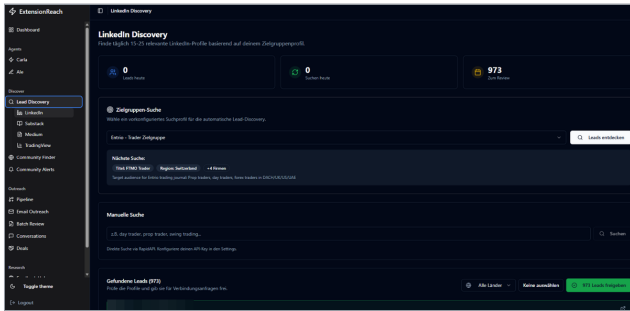
- **Automatisierte Tests.** Vitest, Testing Library und jsdom sind eingerichtet, die Abdeckung ist ein Platzhalter; verifiziert wurde manuell gegen die echte LinkedIn-Oberfläche und die Produktionsdatenbank.
- **Single-Tenant.** Keine Nutzertrennung in den Policies; für ein internes Werkzeug vertretbar, für mehrere Nutzer bräuchte es ein `user_id`-Modell.
- **Zwei parallele Entwurfspfade.** Web-App über OpenAI, Extension-Popup über n8n, historisch gewachsen.
- **Reddit-API-Zugang.** Noch nicht freigeschaltet; die Recherche läuft über die öffentlichen Endpunkte.

Geplant

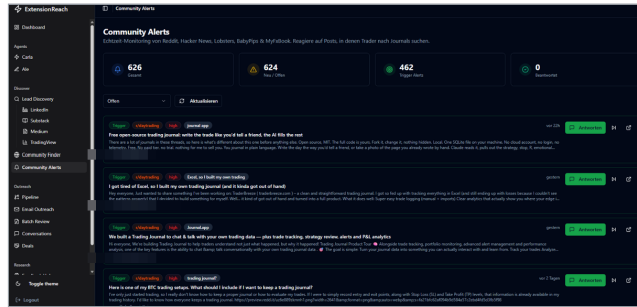
- **Colin**, ein Projekt-Intelligence-Agent über alle fünf Projekte, der Aufgaben an Carla und Ale delegiert (ADR-016, ADR-017).
- **Auto-Publishing** nach LinkedIn und Medium, wöchentliche Trend-Meldungen.
- **Analytics und Mehrbenutzerbetrieb:** Konversion über die Zeit, A/B-Test der Entwurfsvarianten, `user_id`-Modell in den Policies.

8 Einblicke

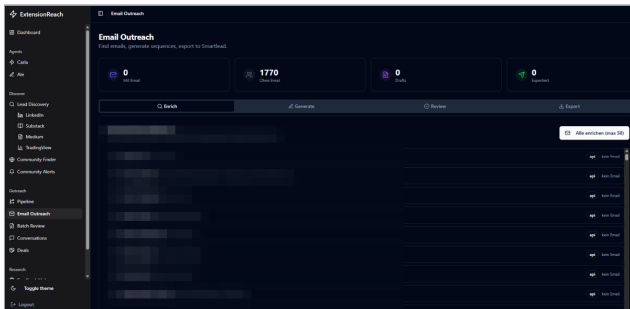
Fünf Oberflächen aus dem laufenden Betrieb: die Beschaffung der Leads, das Monitoring der Communities, der zweite Kanal per E-Mail, der Agent mit seinem Tagesbriefing und die Batch Review, in der jede Nachricht ihre Freigabe bekommt.



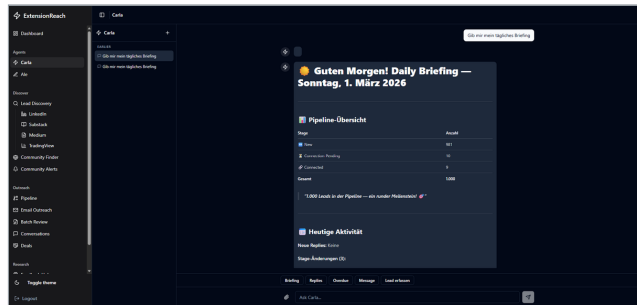
Lead Discovery, LinkedIn. Suchprofil mit Titel, Region und Firmen, dazu die gefundenen Leads, einzeln oder gesammelt freizugeben.



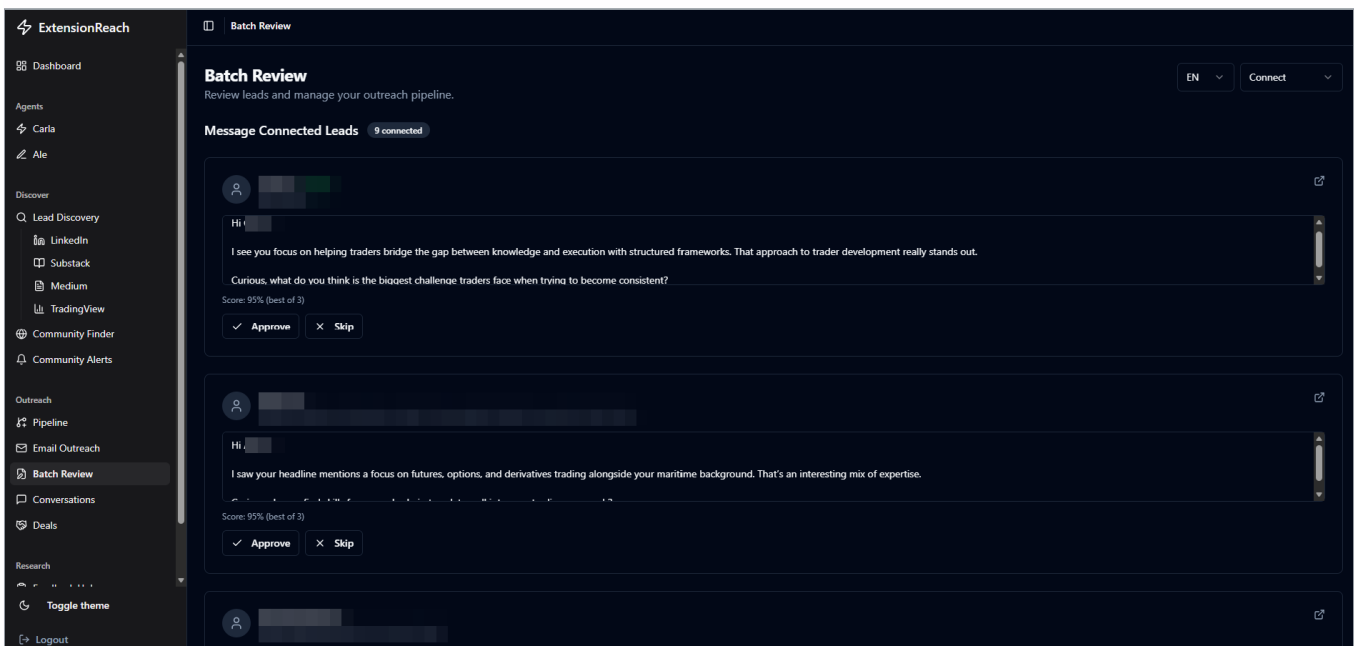
Community Alerts. 626 Treffer aus fünf Plattformen, davon 462 aus Trigger-Begriffen, je mit Quelle, Priorität und Stichwort.



Email Outreach. Anreichern, erzeugen, prüfen, exportieren; Apollo.io holt Adressen zu Leads, die nur ein LinkedIn-Profil haben.



Carla. Das Tagesbriefing als gerendertes Markdown, mit Pipeline-Tabelle, Gesprächshistorie und Schnellaktionen.



Batch Review. Der Kern des Ablaufs: jeder Lead mit aufklappbarer Recherche, drei bewerteten Entwürfen und der Freigabe, ohne die keine Nachricht das Haus verlässt.

9 Was das Projekt zeigt

Fähigkeit	Wo sie sichtbar wird
Frontend-Entwicklung	17 React-Seiten mit TypeScript, TanStack Query, Realtime, Zod, PWA, Streaming-Chat, Stufenfilter
Backend und Daten	34 Tabellen, 38 Migrationen, Row-Level-Security-Policies, <code>pg_cron</code> , 15 Edge Functions
KI-Entwicklung	Function-Calling-Agenten auf zwei Anbietern, SSE-Streaming, geerdete Prompts, Faktenprüfung
Fremde Oberflächen lesen	Zwei Manifest-V3-Extensions: Service Worker, Side-Panel-API, MutationObserver, Shadow-DOM
Datenbeschaffung	Elf eigene Scraper mit Rate Limiting und Bewertung, RSS, Apify-Actors
Betrieb und Automatisierung	Acht Workflows mit Cron und Fehlermeldung, Playwright auf CI, Secrets-Verwaltung
Sicherheit	Allowlist, JWT-Verifikation, dual authentifizierter Proxy, gewachsene RLS
Produkt und Prozess	Personas in den Prompts, Entscheide als ADR, eine Kennzahl, menschliche Freigabe

Stand	September 2026 · in aktiver Weiterentwicklung
Kontakt	Gabriel Suter · kontakt@gabrielsuter.dev · gabrielsuter.dev
Quellcode	Privates Repository, Einsicht und Demo auf Anfrage
Betrieb	extensionreach.com auf Vercel, Backend auf Supabase