app.guidetranslator.com

Update_V7_test/build7b: 26: Februar, 2026, 17:49

HOTFIX C1EBCE2

Fixes: 6 Probleme in einem Commit gelöst ✅

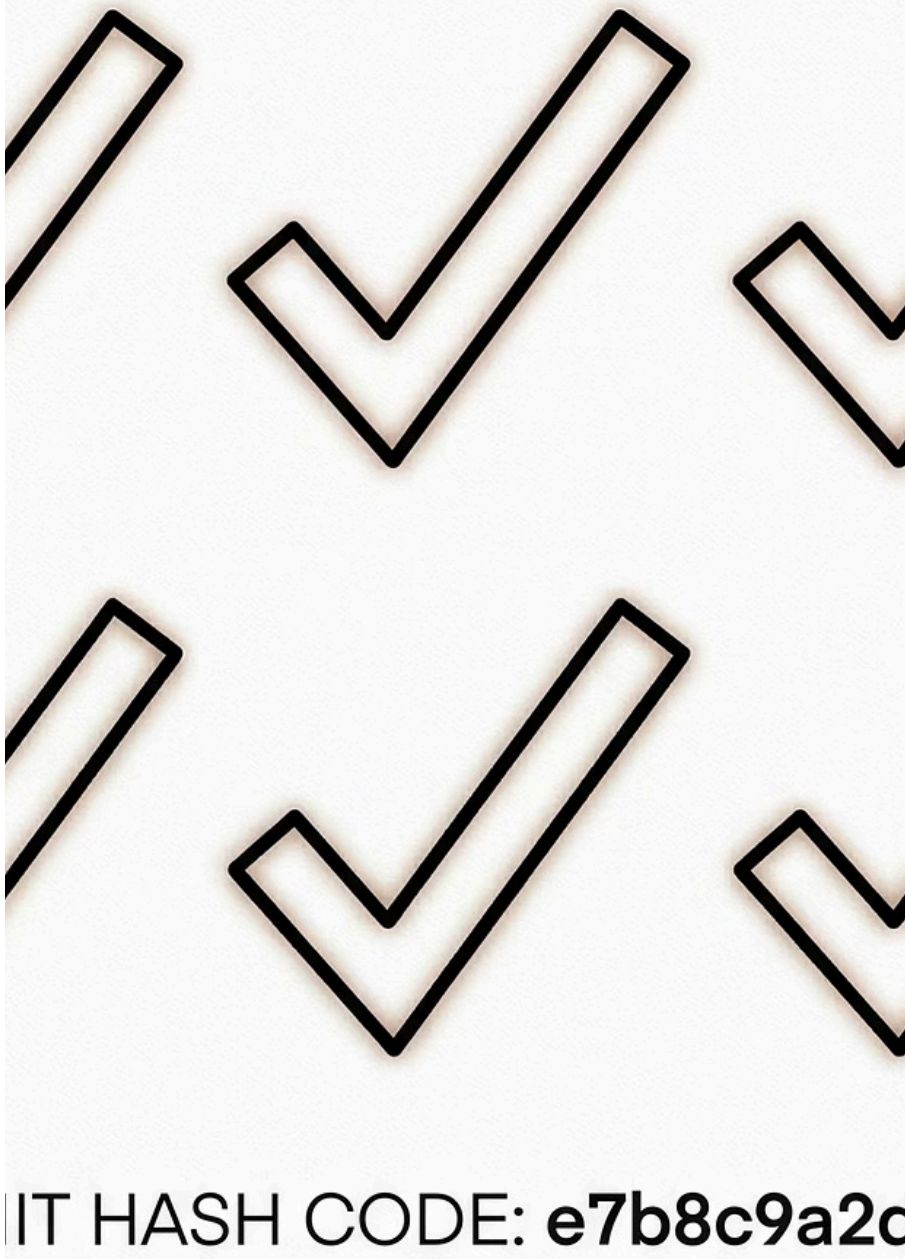
Commit c1ebce2 hat 6 der 7 'Wichtig'-Punkte auf einmal behoben — Checkout-Flow, CORS, Rate-Limiting, Cron-Crash und Sicherheitslücken.

💡 Ulrichs Erklärung: Was ist ein Commit?

Ein Commit ist wie ein Speicherpunkt in einem Videospiel — aber für Code. Jedes Mal wenn Claude Änderungen fertig hat, wird ein 'Commit' gemacht: alle Änderungen werden zusammengepackt, mit einem Namen versehen (hier: c1ebce2) und dauerhaft gespeichert. Dieser eine Commit hat gleich 6 Probleme auf einmal behoben.

#	Problem	Status
8	Checkout → Dashboard disconnect	✅ Gefixt — redirect zu /:segment/offer statt /dashboard
9	Kein customerEmail beim Checkout	✅ Gefixt — E-Mail wird jetzt aus localStorage gelesen
10	CORS * auf allen Endpoints	✅ Gefixt — dynamische Origin-Prüfung statt Wildcard
11	Kein Rate-Limiting	✅ Gefixt — In-Memory Limiter eingebaut
13	check-followups.js crasht	✅ Gefixt — fehlende Tabelle existiert jetzt
14	Dependabot: rollup vulnerability	✅ Gefixt — npm audit fix ausgeführt

> 6 von 7 'Wichtig'-Punkten in einem einzigen Commit — nur #12 (E-Mail-Verifizierung) und #15 (Passwort-Reset) sind noch offen.



CHECKOUT

Fix #8 + #9: Checkout-Flow repariert

Zwei zusammenhängende Bugs im Checkout-Flow wurden behoben: Nach der Zahlung landete der User auf einer Login-Seite — und Stripe wusste nicht, wer gerade bezahlt hat.

💡 Ulrichs Erklärung: Was war das Problem?

Stell dir vor, du kaufst etwas im Online-Shop, bezahlst mit Kreditkarte — und wirst danach auf eine Seite weitergeleitet, die sagt 'Bitte erst anmelden'. Das ist genau was passiert ist. Außerdem: Stripe wusste nicht, welche E-Mail-Adresse der Käufer hat, weil sie nie mitgeschickt wurde. Jetzt wird die E-Mail aus dem Browser-Speicher gelesen und mitgeschickt.

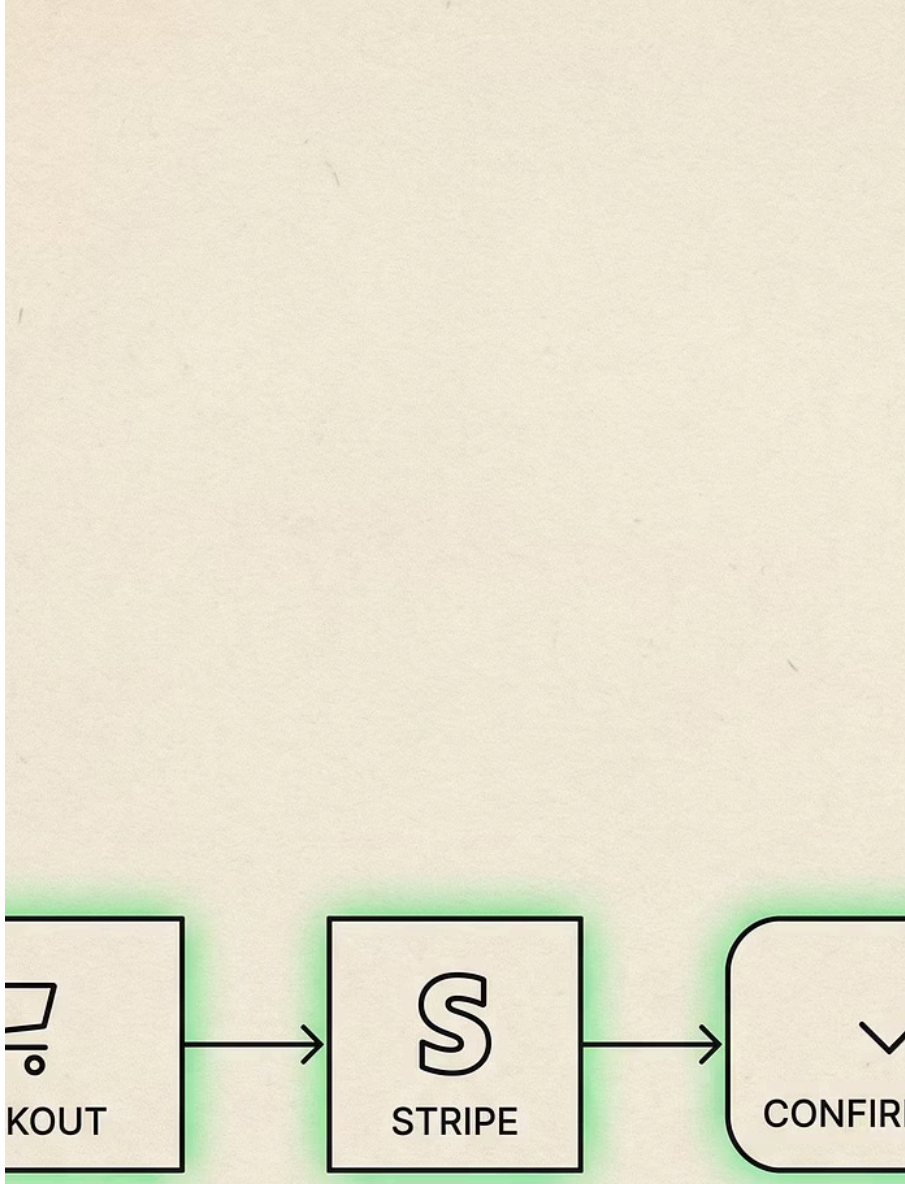
Fix #8: Checkout → Dashboard Disconnect

- Vorher: Nach Stripe-Zahlung → Redirect zu /dashboard → User nicht eingeloggt → weiße Seite / "Bitte anmelden"
- Nachher: Redirect zu /:segment/offer — User landet auf der Angebots-Bestätigungsseite, kein Login nötig
- Warum wichtig: Jeder zahlende Kunde sah nach dem Kauf eine Fehlermeldung

Fix #9: Kein customerEmail beim Checkout

- Vorher: Stripe-Checkout ohne E-Mail → Webhook konnte Lead nicht zuordnen → Abo wurde nie aktiviert
- Nachher: E-Mail wird aus localStorage gelesen und an Stripe übergeben
- Warum wichtig: Ohne E-Mail-Match wurde kein einziges Abo korrekt aktiviert

> Diese zwei Fixes zusammen bedeuten: Der Checkout-Flow funktioniert jetzt von Anfang bis Ende — Zahlung → Bestätigung → Abo aktiv.



SECURITY

Fix #10 + #11: Sicherheit — CORS & Rate-Limiting

Zwei Sicherheitslücken wurden geschlossen: Die API war für jeden im Internet offen, und Formulare konnten unbegrenzt oft abgeschickt werden.

💡 Ulrichs Erklärung: Was ist CORS und Rate-Limiting?

CORS *: Stell dir vor, dein Büro hat eine Tür, die für jeden offen ist — auch für Fremde, die nichts dort zu suchen haben. CORS * bedeutet, jede beliebige Website der Welt konnte deine App-API aufrufen. Jetzt wird geprüft: Kommt die Anfrage von unserer eigenen Domain? Wenn nicht — abgelehnt. Rate-Limiting: Ohne das könnte jemand dein Kontaktformular 10.000 Mal pro Minute absenden und den Server lahmlegen. Der In-Memory Limiter sagt: 'Du hast genug gesendet, warte kurz.'

Fix #10: CORS * → Dynamische Origin-Prüfung

- Vorher: Access-Control-Allow-Origin: * — jede Website konnte die API aufrufen
- Nachher: Dynamische Prüfung — nur Anfragen von sales.guidetranslator.com werden akzeptiert
- Betrifft: alle API-Endpoints (create-checkout, stripe-webhook, send-email, etc.)

Fix #11: In-Memory Rate-Limiter

- Vorher: Kein Schutz — Formulare konnten unbegrenzt abgeschickt werden
- Nachher: In-Memory Limiter — begrenzt Anfragen pro IP-Adresse und Zeitfenster
- Schützt vor: Spam, Brute-Force-Angriffe, Server-Überlastung

> Diese zwei Fixes machen die Plattform produktionsreif — offene APIs und unbegrenzte Formulare sind klassische Angriffsvektoren.

STABILITÄT

Fix #13 + #14: Cron-Crash & Sicherheitslücke behoben

Der täglich laufende Follow-up Cron-Job crashte still — und eine bekannte Sicherheitslücke in einer Abhängigkeit wurde mit einem einzigen Befehl geschlossen.

💡 Ulrichs Erklärung: Was ist eine Dependency-Vulnerability?

Die App benutzt viele fertige Code-Pakete von anderen Entwicklern (sogenannte Dependencies) — wie Bausteine. Manchmal entdeckt jemand eine Sicherheitslücke in einem dieser Bausteine. GitHub (Dependabot) meldet das automatisch. Mit 'npm audit fix' wird der betroffene Baustein auf eine sichere Version aktualisiert — wie ein Sicherheits-Update auf dem Handy, nur für den Server-Code.

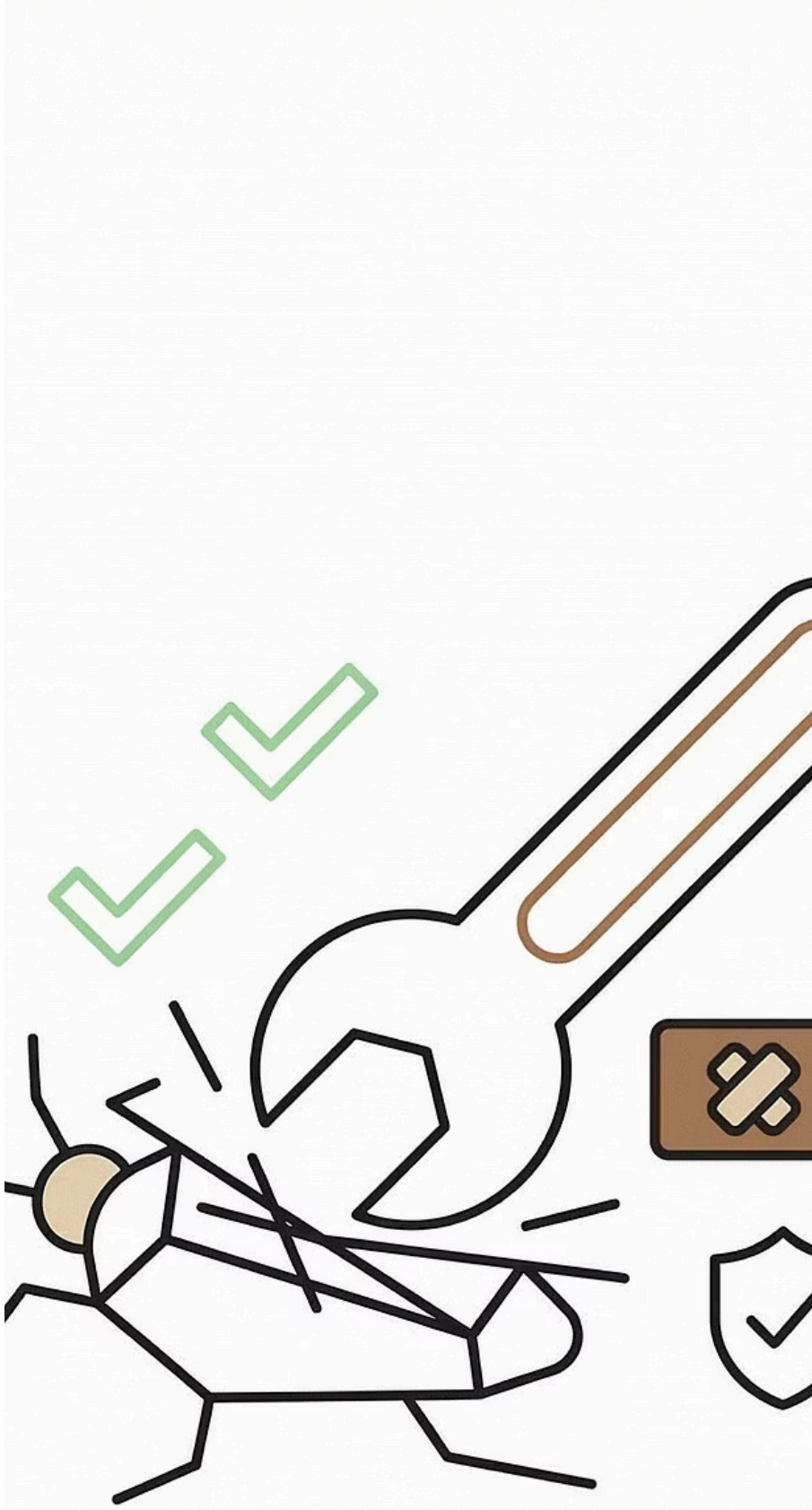
Fix #13: check-followups.js Crash

- Problem: Der Cron-Job versuchte in die Tabelle gt_lead_notes zu schreiben
- Diese Tabelle existierte nicht in den SQL-Migrations-Dateien
- Folge: Cron lief täglich, crashte still — keine Fehlermeldung, keine Follow-up E-Mails
- Fix: Tabelle gt_lead_notes wurde in den SQL-Files ergänzt und in Supabase angelegt
- Jetzt: Cron läuft durch, Admin-Notizen werden korrekt gespeichert

Fix #14: Dependabot rollup Vulnerability

- Problem: rollup (Build-Tool) hatte eine bekannte Sicherheitslücke (1 high severity)
- GitHub Dependabot hatte das als Warnung gemeldet
- Fix: npm audit fix — aktualisiert rollup auf sichere Version
- Ergebnis: 0 high vulnerabilities, 0 moderate vulnerabilities
- Kein Code-Änderung nötig — nur Versions-Update

> Fix #13 ist besonders wichtig: Ein still crashender Cron-Job ist schwer zu finden — er läuft, aber tut nichts. Jetzt tut er was er soll.



PHASE 9

Phase 9: Auto-Provisioning, E-Mail-Verifizierung & Passwort-Reset

5 neue Features wurden implementiert — vom automatischen Kunden-Account nach Stripe-Zahlung bis zur E-Mail-Verifizierung und Passwort-Reset.

💡 Ulrichs Erklärung: Was ist Auto-Provisioning?

Bisher musste nach einer Zahlung manuell ein Kunden-Account angelegt werden. Auto-Provisioning bedeutet: Sobald jemand bezahlt, passiert alles automatisch — Account erstellt, Rolle zugewiesen, Welcome-E-Mail mit Passwort verschickt. Kein manueller Schritt mehr. Wie ein Hotel, das nach der Online-Buchung automatisch den Schlüssel vorbereitet.

Auto-Provisioning

Stripe-Zahlung → automatisch Supabase Auth-Account + Rolle customer + Welcome-E-Mail mit Passwort

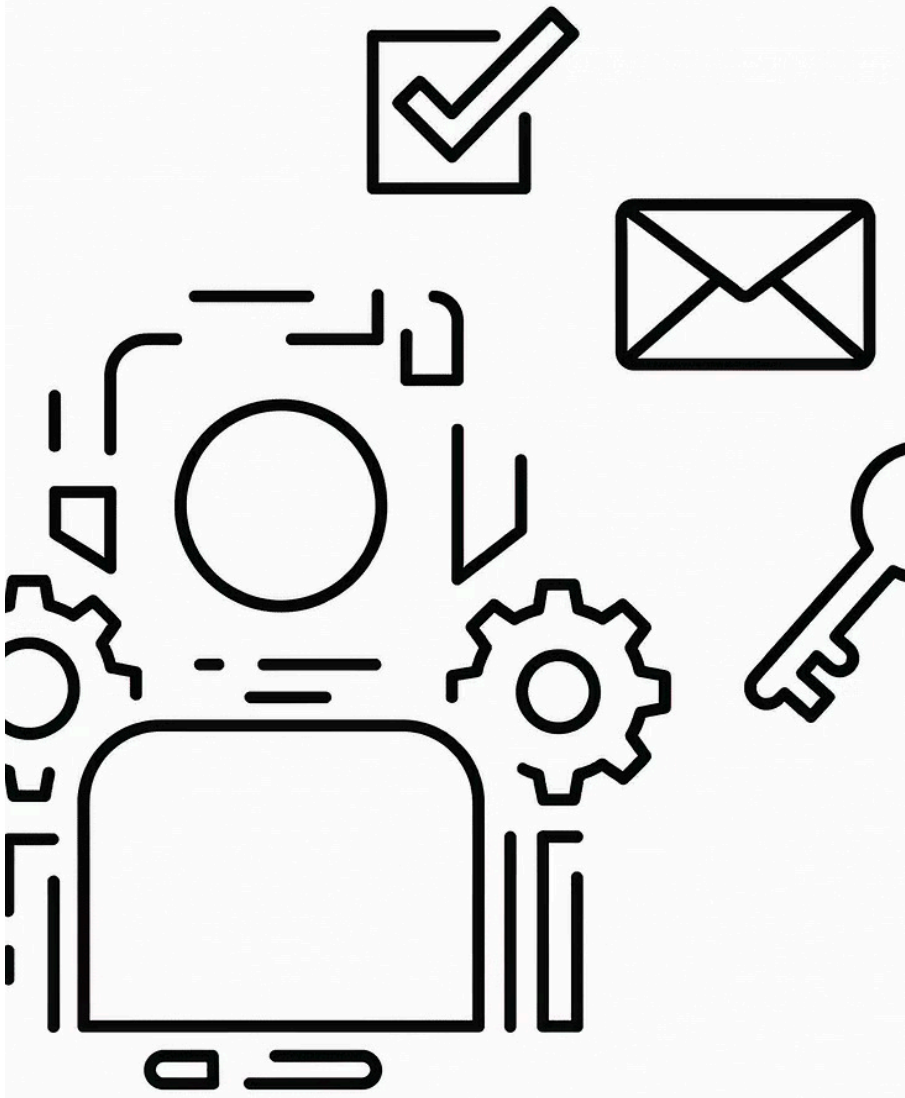
E-Mail-Verifizierung

Register → 6-stelliger Code per E-Mail → Code eingeben → verifiziert. Code 15 Min. gültig, rate-limited.

Passwort-Reset

'Passwort vergessen?' auf Login-Seite → Supabase resetPasswordForEmail() → neue ResetPassword.jsx Seite

> SQL-Migration erforderlich: sql/phase9-email-verification.sql muss in Supabase ausgeführt werden.



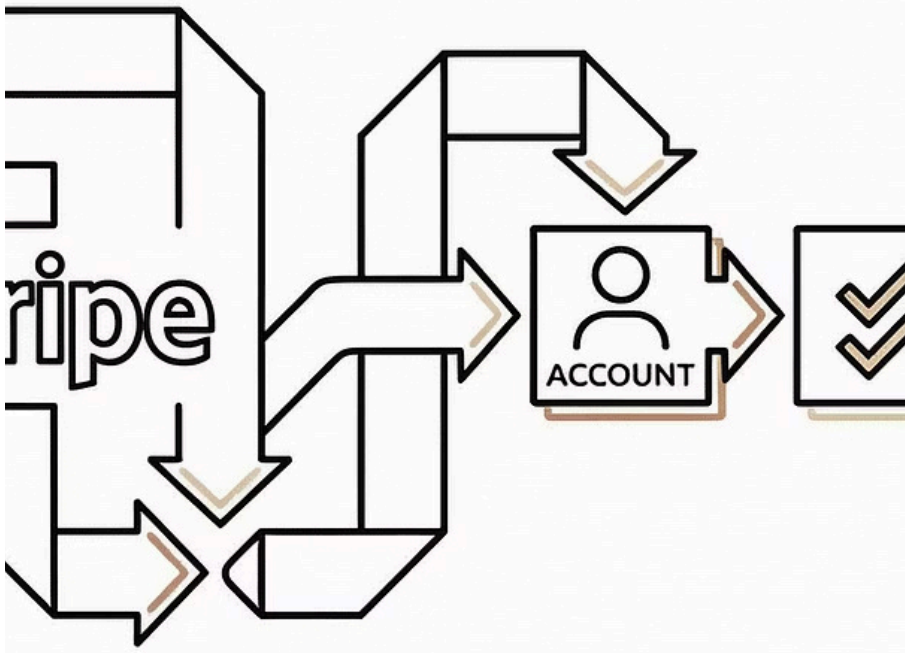
AUTOMATISIERUNG

Feature 1: Auto-Provisioning — Stripe → Kunden-Account

Nach erfolgreicher Stripe-Zahlung wird vollautomatisch ein Supabase Auth-Account erstellt, die Rolle zugewiesen und eine Welcome-E-Mail verschickt — ohne manuellen Eingriff.

💡 Ulrichs Erklärung: Was passiert nach einer Zahlung?

Früher: Kunde bezahlt → du bekommst eine Stripe-Benachrichtigung → du legst manuell einen Account an → du schickst manuell eine E-Mail. Das dauert Stunden oder Tage. Jetzt: Kunde bezahlt → in Sekunden hat er einen Account, eine Rolle und eine E-Mail mit Passwort. Vollautomatisch, 24/7, auch nachts um 3 Uhr.



Stripe-Zahlung erfolgreich

checkout.session.completed Event trifft beim Webhook ein

Supabase Auth-Account erstellt

Automatisch: E-Mail + temporäres 12-Zeichen-Passwort, Rolle: customer in gt_roles

Welcome-E-Mail verschickt

Branded HTML-E-Mail mit temporärem Passwort, Dashboard-Link, Tier-Name und Aufforderung zum Passwort-Ändern

System-Note geloggt

Provisioning-Aktion wird als System-Note in gt_lead_notes gespeichert — für Admin-Nachvollziehbarkeit

> Der Kunde bekommt seinen Zugang in Sekunden — nicht in Stunden. Das ist der Unterschied zwischen manuellem und automatisiertem SaaS.