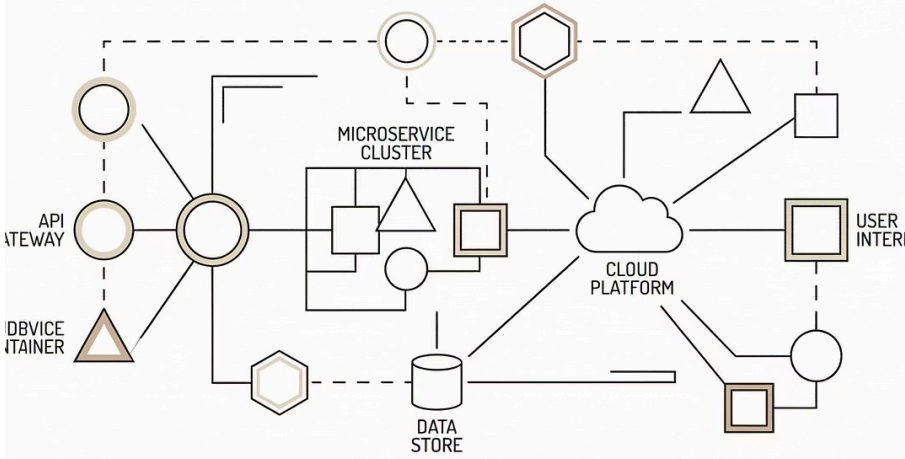
GUIDETRANSLATOR V7

ENGINEERING UPDATE

GuideTranslator v7 — Von Single-Segment Sales-Tool zur Multi-Segment SaaS-Plattform

Vollständiger technischer Fahrplan: 7 Phasen, 6 Segmente, Rollen-Management, Stripe-Billing, Nutzungs-Tracking und automatisierte Follow-ups. Status, Architektur-Entscheidungen und Implementierungsreihenfolge für das gesamte Team.



Ausgangslage: Wo wir stehen

Die aktuelle **sales.guidetranslator.com** ist ein Single-Segment Sales-Tool, ausschließlich für Kreuzfahrt-Enterprise konzipiert. Die Architektur hat ihren Zweck erfüllt — ist aber für die nächste Stufe nicht skalierbar.

React SPA

State-Routing statt echtem Router. Kein react-router.

2 Monolithen

App.jsx (853 Z.) + Admin.jsx (1.500 Z.) — schwer wartbar.

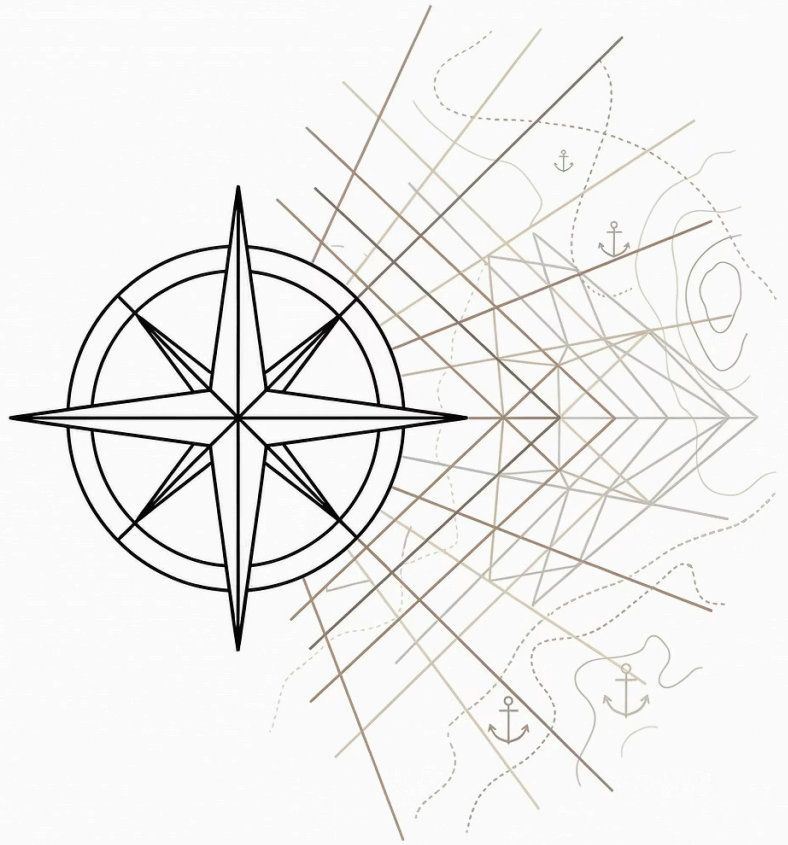
Supabase (4 Tabellen)

Minimales Schema. Keine Auth, kein Rollen-Konzept.

Vercel + Resend

Serverless Functions, E-Mail über Resend. Nur Kreuzfahrt.

🧑‍💼 **Für Ulrich:** Stell dir vor, wir haben aktuell einen einzelnen Laden mit einem einzigen Regal (Kreuzfahrt). Jetzt bauen wir ein Kaufhaus mit 6 Abteilungen, Kassensystem, Kundenkarten und automatischer Nachbestellung.



Die Vision: Multi-Segment SaaS-Plattform

Das Ziel von Version 7 ist die Transformation zu einer vollwertigen SaaS-Plattform mit **6 Zielgruppen**, dediziertem Rollen-Management, integriertem Billing über Stripe und automatisierten Sales-Workflows. Die Plattform soll jedes Segment mit eigener Landing, eigenem Kalkulator und eigenen E-Mail-Templates bedienen — bei gemeinsamem Technologie-Kern.

6 Segmente

Stadtführer, Agentur, Veranstalter, Kreuzfahrt, Großveranstalter, Fintutto Single

Rollen-Management

Super-Admin, Admin, Sales, Customer, Sub-Account — feingranular pro Segment

Stripe-Billing

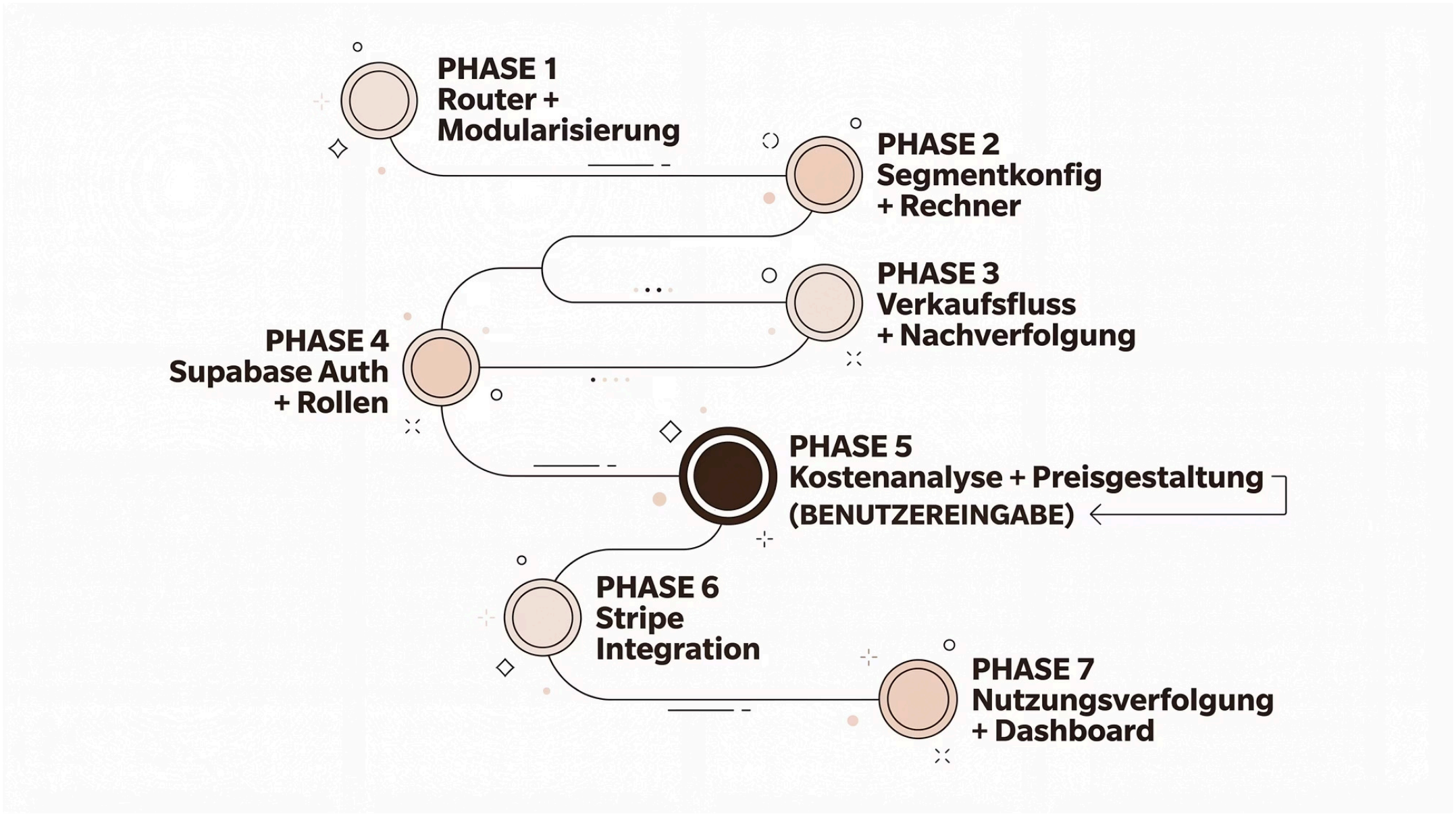
Checkout, Subscriptions, Self-Service Portal, Add-On Credits

Usage-Tracking

Minuten, Sprachen, Hörer — live im Kunden-Dashboard

7 Phasen — Der Masterplan

Die Umsetzung ist in **7 klar abgegrenzte Phasen** gegliedert. Jede Phase hat definierte Eingaben, Ausgaben und Abhängigkeiten. Die Reihenfolge optimiert für maximale Parallelisierung und minimale Blocker.



Phasen 1–3 können **sofort** starten — keine externen Abhängigkeiten. Phase 4 kann parallel zu 2+3 beginnen. Phasen 5–7 erfordern Klärung (Stripe, Kosten, App-API).

> 🧑‍💼 **Für Ulrich:** Denk an einen Hausbau: Erst kommt das Fundament (Phase 1), dann können Wände und Dach gleichzeitig gebaut werden (Phase 2+3+4). Strom und Wasser (Phase 5–7) kommen, sobald die Grundstruktur steht.

PHASE 1

Refactoring — Router + Modularisierung

Ziel: Codebase aufbrechen, `react-router-dom` einführen. Die bestehende Funktionalität bleibt identisch — aber die Architektur wird modular und erweiterbar. `App.jsx` schrumpft von **853 auf ~40 Zeilen**.

Extrahierte Shared-Module

Datei	Inhalt
`lib/tokens.js`	Design Tokens T, font, fontSans
`lib/helpers.js`	fmt, fmtEur, fmtPct, fmtDate, generateToken
`lib/supabaseHelpers.js`	upsertLead, loadLeadByEmail, saveCalculation
`lib/constants.js`	Pipeline-Stufen, Note-Types, Tags
`components/Icon.jsx`	Icon-Komponente
`components/FormField.jsx`	Formular-Feld-Komponente
`components/Slider.jsx`	Kalkulator-Slider

Extrahierte Seiten

Datei	Quelle
`pages/Landing.jsx`	App.jsx Z.378–509
`pages/Register.jsx`	App.jsx Z.531–588
`pages/Calculator.jsx`	App.jsx Z.593–730
`pages/Saved.jsx`	App.jsx Z.735–766
`pages/Contact.jsx`	App.jsx Z.771–853

PHASE 1 FORTSETZUNG

Routen-Architektur + Admin-Aufteilung

Die neue Routing-Struktur nutzt `react-router-dom` mit segment-parametrisierten Pfaden. Der Admin-Monolith wird in eigenständige Views mit Nested Routes aufgeteilt.

Neue Routen

- `/` → Segment-Hub (Auswahl)
- `/:segment` → Segment-Landing
- `/:segment/register` → Registration
- `/:segment/calculator` → Kalkulator
- `/:segment/saved` → Gespeicherte Kalkulationen
- `/:segment/contact` → Anfrage/Angebot
- `/howto` → App-Anleitung (NEU)
- `/admin/*` → Admin-Dashboard
- `/dashboard/*` → Kunden-Dashboard (spätere Phase)

Admin aufteilen

01

AdminLayout.jsx

Shell + Nav + Nested Routes

02

Contacts.jsx

ContactsList (Z.444–626)

03

ContactDetail.jsx

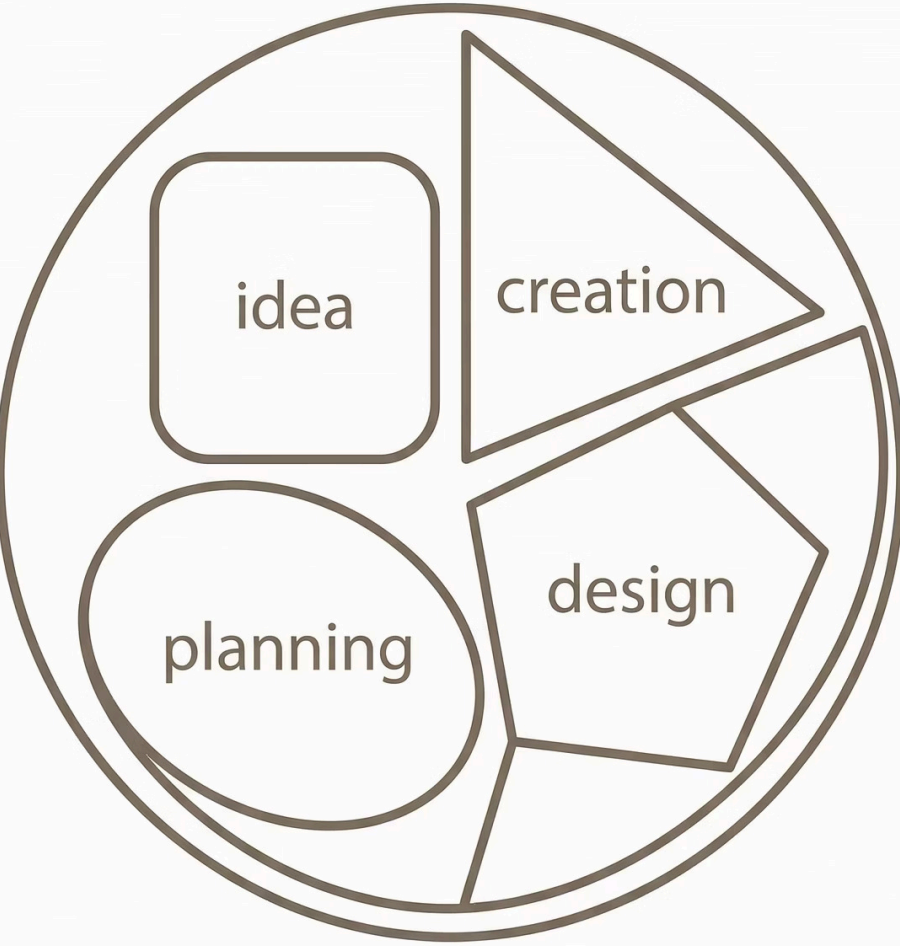
Detail-View (Z.778–1071)

04

Activity.jsx + EmailModal.jsx

ActivityLog, Email- & Bulk-Modal

Verifikation: `npm run build` OK, alle Routen navigierbar, bestehende Funktionalität identisch.



PHASE 2

Segment-Konfiguration + Differenzierte Kalkulatoren

Kern von Phase 2 ist die neue Datei `src/config/segments.js` — sie definiert alle 6 Segmente mit jeweils eigenen Kalkulator-Parametern, Farben, Hero-Inhalten, Pain-Points und E-Mail-Templates. Jedes Segment erhält eine vollständig konfigurierbare Berechnungs-Engine.

PHASE 2 | SEGMENT-ÜBERSICHT

6 Segmente im Detail

Segment	Kalkulator-Parameter	Farbe	Zielgruppe
Stadtführer	Touren/Monat, Min/Tour, Sprachen, Hörer/Tour	#27ae60	Einzelne Guides & kleine Büros
Agentur	Guides, Touren/Guide/Monat, Min/Tour, Sprachen, Hörer	#2a9bc0	Reiseagenturen mit Guide-Pool
Veranstalter	Events/Monat, Stunden/Event, Sprachen, Teilnehmer	#e67e22	Event-Organisatoren
Kreuzfahrt	Bestehend (ships, pax, excursions…)	#c8a84e	Kreuzfahrt-Enterprise
Großveranstalter	Venues, Events/Venue, Stunden, Sprachen, Teilnehmer	#9b59b6	Multi-Venue Betreiber
Fintutto Single	Minuten/Monat, Sprachen, Hörer	#1a6b8a	Fintutto-Einzelkunden

Jedes Segment definiert: `hero` (Headline, Stats), `painPoints`, `calcParams` (Felder, Defaults, Min/Max), `calcFormula`, `emailTemplates` und `registerFields`. Die bestehende Kreuzfahrt-Logik wird 1:1 aus dem aktuellen `Calculator` extrahiert.

DB-Änderung: `gt_leads` + `gt_calculations` bekommen `segment TEXT DEFAULT 'kreuzfahrt'`.

> 🧑‍💼 **Für Ulrich:** Bisher konnte die Website nur Kreuzfahrt-Preise berechnen. Jetzt bekommt jede unserer 6 Kundengruppen ihren eigenen maßgeschneiderten Preisrechner — mit passenden Fragen und Formeln. Wie 6 verschiedene Kassenzettel-Vorlagen für 6 verschiedene Geschäftstypen.

PHASE 3

Sales-Flow: Angebot → Test → Follow-up

Phase 3 erweitert den Sales-Funnel um einen **Post-Offer Flow**: Nach dem Speichern einer Kalkulation und Absenden einer Anfrage sieht der Lead eine dedizierte PostOffer-Seite mit klaren nächsten Schritten.

Kalkulation speichern

Anfrage wird gesendet

PostOffer.jsx

✅ Bestätigung anzeigen

Jetzt testen / Später

→ App oder .ics Download

Follow-up

Cron nach 7 Tagen

Neue Dateien

- `src/pages/PostOffer.jsx` — Post-Offer mit Test/Später Buttons
- `src/pages/HowTo.jsx` — Schritt-für-Schritt App-Anleitung
- `src/lib/icsGenerator.js` — .ics Kalender-Generator

DB-Änderungen

3 neue Spalten auf `gt_leads`:

- `offer_created_at` TIMESTAMPTZ
- `tested_at` TIMESTAMPTZ
- `test_reminder_sent_at` TIMESTAMPTZ

Erweiterte Pipeline

01

Neu → Eingeladen → Registriert

02

Kalkulation → Angebot erstellt

03

Getestet | Testet später → Erinnert

04

Demo → Verhandlung → Gewonnen/Verloren

Cron Job: `api/check-followups.js` — Täglich 08:00: Leads mit Angebot > 7 Tage + nicht getestet → Follow-up vorbereiten.

PHASE 4

Supabase Auth + Rollen-System

Größter technischer Eingriff. Die bisherige Passwort-Authentifizierung wird vollständig durch Supabase Auth ersetzt. Neue Tabellen `gt_roles` und `gt_organizations` bilden das Fundament für feingranulares Rollen- und Organisations-Management.

PHASE 4 | SCHEMA + IMPLEMENTIERUNG

Auth-Schema + Neue Komponenten

Neue DB-Tabellen

```
CREATE TABLE gt_roles (
  id UUID PRIMARY KEY,
  user_id UUID REFERENCES auth.users(id),
  role TEXT CHECK (role IN (
    'super_admin','admin','sales',
    'customer','sub_account'
  )),
  segment TEXT,
  organization_id UUID REFERENCES gt_organizations(id),
  created_by UUID,
  UNIQUE(user_id)
);

CREATE TABLE gt_organizations (
  id UUID PRIMARY KEY,
  name TEXT NOT NULL,
  segment TEXT NOT NULL,
  owner_user_id UUID,
  stripe_customer_id TEXT
);
```

Neue Dateien

AuthContext.jsx

Supabase Auth Session + Rollen-Lookup

ProtectedRoute.jsx

AdminRoute, SalesRoute, CustomerRoute

Login.jsx

Unified Login für alle Rollen

api/invite-user.js

Admin erstellt User mit Rolle per E-Mail

api/seed-admin.js

Einmalig: Alexander (super_admin) + Ulrich (admin)

Migration: `api/migrate-leads.js` überführt bestehende Leads einmalig in Supabase Auth Users.

🧑‍💼 **Für Ulrich:** Bisher hatte jeder das gleiche einfache Passwort-System. Ab Phase 4 bekommt jeder Nutzer seine eigene Rolle: Du wirst „Admin", Alexander „Super-Admin", Kunden sehen nur ihr eigenes Dashboard. Wie Schlüsselkarten in einem Hotel — jede öffnet nur die richtige Tür.

⚠ PHASE 5 ⚠️ BLOCKER

Kostenanalyse + Pricing — Erfordert User-Input

Phase 5 ist der **zentrale Blocker** für die nachfolgenden Phasen. Bevor Code geschrieben wird, müssen externe Kosten verifiziert und finale Preise festgelegt werden.

Google Cloud

Translation/TTS Kosten verifizieren

Server-Kosten

app.guidetranslator.com Hosting

Stripe Status

✅ Geklärt: Account ist LIVE

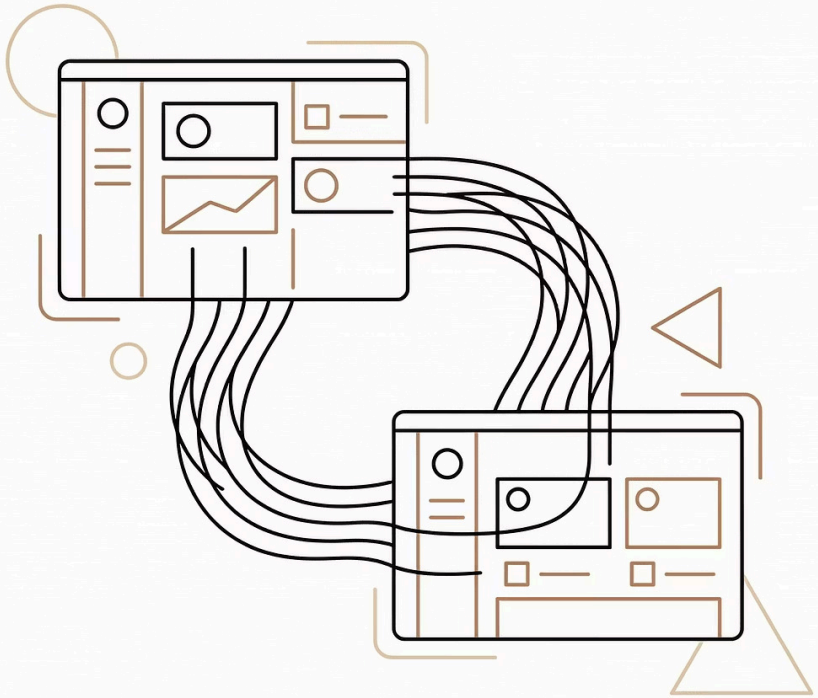
Endgültige Preise

Pro Tier finalisieren

Vorgeschlagene Tier-Struktur

Tier	Preis/Monat	Minuten	Sprachen	Hörer	Anonym	Free
Anonym	Kostenlos	5 (einmalig, IP)	1	1	✅	—
Free	Kostenlos	20/Monat	1	1	—	✅
Starter	€19	120	3	5	—	—
Pro	€49	500	10	25	—	—
Business	€149	2.000	20	100	—	—
Enterprise	€499	10.000	30	500	—	—
Custom	Anfrage	Unbegrenzt	Alle	Unbegrenzt	—	—

Add-Ons: +100 Min (€9), +500 Min (€39), +5 Sprachen (€15/mo), +50 Hörer (€19/mo)



PHASE 6

Stripe Integration — Billing-Engine

Phase 6 implementiert die komplette Zahlungsinfrastruktur: Checkout, Webhooks, Self-Service Portal und die zugehörigen DB-Tabellen für Subscriptions und Credits.
Dependencies: `stripe` (Server) + `@stripe/stripe-js` (Client).

PHASE 6 | SCHEMA + API

Stripe: Dateien, Tabellen, Webhooks

Neue API-Endpunkte

1

stripe-checkout.js

Erstellt Checkout Session basierend auf gewähltem Tier + Segment

2

stripe-webhook.js

Verarbeitet: `subscription.updated`, `checkout.completed`, `payment_failed`

3

stripe-portal.js

Kunden Self-Service Portal für Plan-Änderungen & Rechnungen

DB-Schema

```
CREATE TABLE gt_subscriptions (
  id UUID PRIMARY KEY,
  organization_id UUID REFERENCES gt_organizations(id),
  stripe_subscription_id TEXT UNIQUE,
  plan_id TEXT NOT NULL,
  segment TEXT NOT NULL,
  status TEXT DEFAULT 'active',
  minutes_limit INTEGER DEFAULT 20,
  languages_limit INTEGER DEFAULT 1,
  listeners_limit INTEGER DEFAULT 1,
  current_period_start TIMESTAMPTZ,
  current_period_end TIMESTAMPTZ
);

CREATE TABLE gt_credits (
  id UUID PRIMARY KEY,
  organization_id UUID REFERENCES gt_organizations(id),
  credit_type TEXT CHECK (credit_type IN
   ('minutes','languages','listeners')),
  amount INTEGER NOT NULL,
  remaining INTEGER NOT NULL,
  stripe_payment_id TEXT,
  expires_at TIMESTAMPTZ
);
```

🧑‍💼 **Für Ulrich:** Stripe ist wie ein automatisches Kassensystem. Kunden wählen ihren Plan, bezahlen online, und das System bucht monatlich automatisch ab. Wenn jemand mehr Minuten braucht, kauft er Add-Ons dazu — wie Zusatzpakete beim Handy-Vertrag.

PHASE 7

Nutzungs-Tracking + Kunden-Dashboard

Phase 7 schließt den Kreis: Die App (app.guidetranslator.com) meldet per API-Key-Auth Nutzungsdaten, das System prüft Limits und das Kunden-Dashboard visualisiert den Verbrauch in Echtzeit.

DB-Tabellen

```
CREATE TABLE gt_usage (
  id UUID PRIMARY KEY,
  organization_id UUID,
  user_id UUID,
  usage_type TEXT CHECK (usage_type IN
    ('minutes','languages','listeners')),
  amount NUMERIC NOT NULL,
  session_id TEXT,
  recorded_at TIMESTAMPTZ DEFAULT now()
);

CREATE TABLE gt_anonymous_usage (
  id UUID PRIMARY KEY,
  ip_hash TEXT NOT NULL UNIQUE,
  minutes_used NUMERIC DEFAULT 0,
  first_used TIMESTAMPTZ DEFAULT now()
);
```

API: api/track-usage.js — Von app.guidetranslator.com aufgerufen, prüft Limits gegen Subscription.

Kunden-Dashboard Views

Overview.jsx

Nutzung, Kontingente, Quick Actions auf einen Blick

Usage.jsx

Detail nach Tag/Guide — Drilldown in den Verbrauch

Team.jsx

Sub-Accounts anlegen und verwalten

Billing.jsx

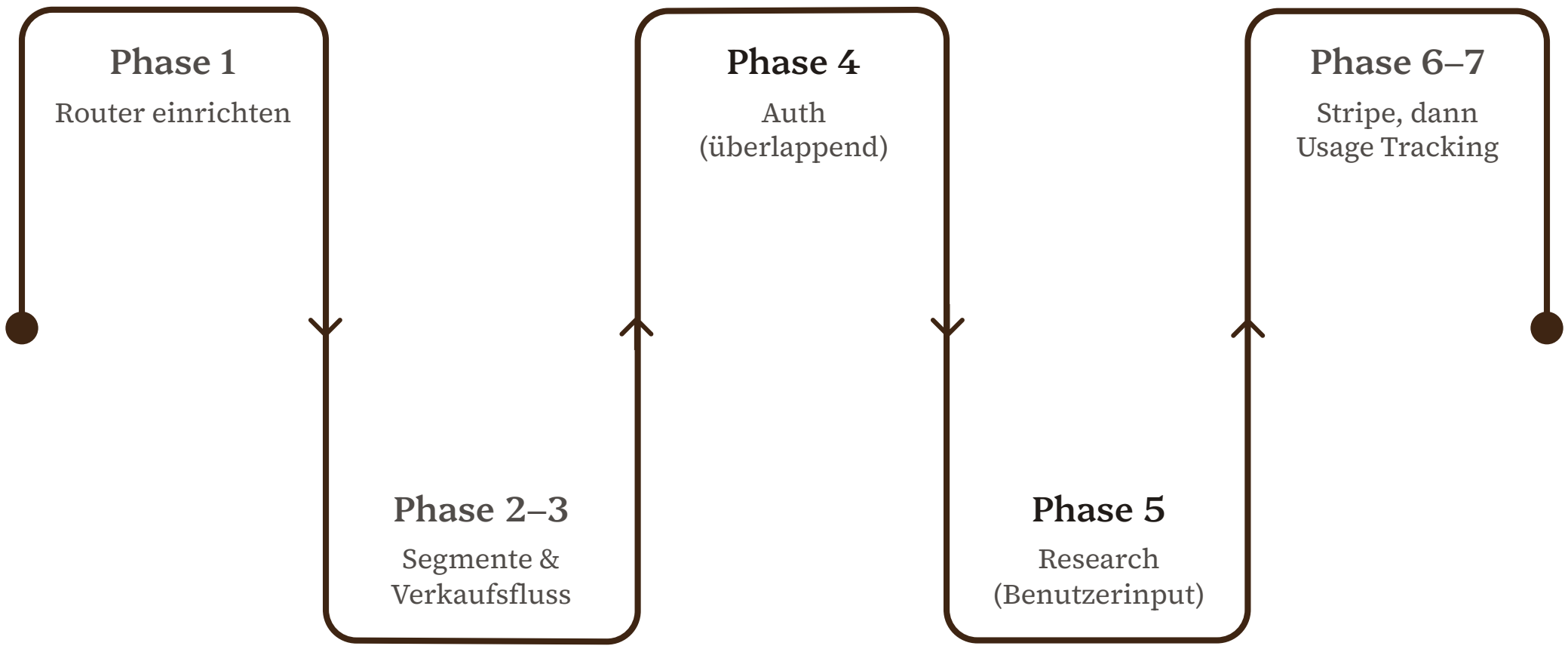
Stripe Portal, Credits, Plan-Übersicht

Hörer: Kein Login erforderlich. QR-Code → App. Concurrent Count per Session-UUID.

Implementierungsreihenfolge + Abhängigkeiten

Die folgende Übersicht zeigt die exakte Reihenfolge und Abhängigkeiten aller Phasen. Die empfohlene Sequenz maximiert Parallelisierung und minimiert Wartezeiten.

#	Phase	Beschreibung	Blockiert von	Status
1	Phase 1	Router + Modularisierung	—	🟢 Sofort startbar
2	Phase 2	Segment-Config + Kalkulatoren	Phase 1	🟢 Sofort nach P1
3	Phase 3	Sales-Flow + Follow-up	Phase 1	🟢 Parallel zu P2
4	Phase 4	Supabase Auth + Rollen	Phase 1	🟡 Parallel zu P2+3
5	Phase 5	Kostenanalyse + Pricing	USER-INPUT	🔴 Blocker
6	Phase 6	Stripe Integration	Phase 4 + 5	🔴 Wartet auf P5
7	Phase 7	Usage-Tracking + Dashboard	Phase 4 + 6	🔴 Wartet auf P6



Empfohlene Reihenfolge: 1 → 2+3 (parallel) → 4 → 5 (Recherche) → 6 → 7

Verifikation pro Phase

Jede Phase hat klar definierte Akzeptanzkriterien. Die Verifikation stellt sicher, dass nichts kaputt geht und jede Stufe eigenständig deploybar bleibt.

1

Phase 1

`npm run build` OK, alle Routen navigierbar, bestehende Funktionalität identisch

2

Phase 2

6 Segment-Landings laden korrekt, dynamische Kalkulatoren funktionieren mit segment-spezifischen Formeln

3

Phase 3

PostOffer zeigt Test/Später, .ics Download funktioniert, Pipeline-Stufen im Admin korrekt

4

Phase 4

Login funktioniert, Admin kann Accounts anlegen, Rollen-Guards greifen für alle Routes

5

Phase 6

Stripe Checkout → Subscription aktiv → Limits korrekt gesetzt in gt_subscriptions

6

Phase 7

Usage-API erreichbar, Dashboard zeigt Verbrauch, Limits werden bei Überschreitung enforced

Offene Fragen + Blocker

Einige kritische Fragen sind bereits geklärt, andere blockieren die finalen Phasen. Hier der aktuelle Stand.

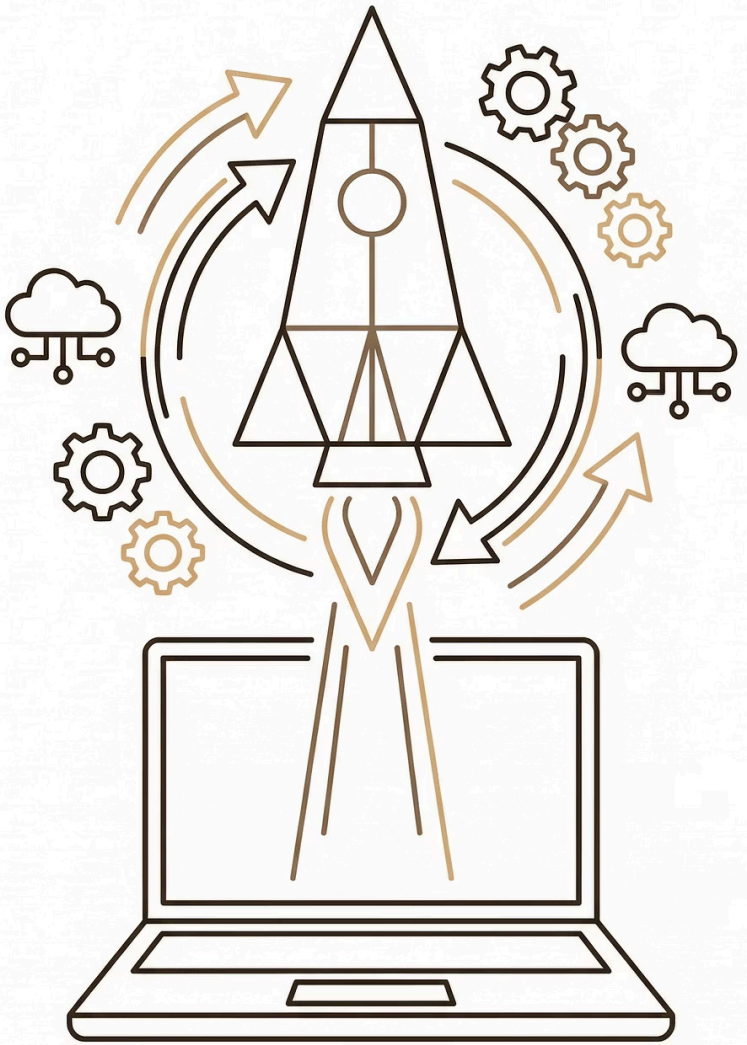
✅ Geklärt

- **Stripe Account:** Existiert und ist LIVE (kein Test-Modus)
- **Implementierung:** Erst komplett durchplanen, dann umsetzen

🔴 Noch offen (BLOCKER Phase 5–7)

- **app.guidetranslator.com:** Welche Technologie? Kann sie API-Calls senden?
- **Google Cloud Kosten:** Dashboard mit echten Nutzungsdaten vorhanden?
- **portal.fintutto.cloud:** Shared Auth gewünscht oder separate Accounts?

🧑‍💼 **Für Ulrich:** Das meiste können wir sofort bauen. Aber für die automatische Abrechnung (Phase 6+7) brauchen wir noch echte Zahlen: Was kostet uns Google Cloud pro Übersetzungsminute? Und kann die eigentliche App (die die Guides nutzen) mit unserem neuen System „sprechen"? Das muss geklärt werden, bevor wir dort weitermachen.



Nächste Schritte — Jetzt starten

Die Architektur steht, der Plan ist klar. Drei Phasen können **sofort** begonnen werden — ohne externe Abhängigkeiten.

Sofort: Phase 1 starten

Router + Modularisierung — Fundament für alles Weitere

Danach: Phase 2+3 parallel

Segmente + Sales-Flow gleichzeitig entwickeln

Parallel klären: Offene Fragen

App-API, Google Kosten, Fintutto Auth — Blocker für Phase 5–7 auflösen

> **Von einem Single-Segment Sales-Tool zur vollständigen Multi-Segment SaaS-Plattform — in 7 systematischen Phasen.**

7 Phasen — Von Kreuzfahrt-Tool zur Multi-Segment SaaS Plattform

Eine Übersicht der geplanten sieben Phasen zur Transformation vom spezialisierten Kreuzfahrt-Tool zu einer vielseitigen Multi-Segment SaaS-Plattform.

Phase	Was	Dateien
Phase 1	Router + Modularisierung (App.jsx aufbrechen)	~15 neue Dateien
Phase 2	6 Segmente + dynamische Kalkulatoren	segments.js + 3 Pages
Phase 3	Sales-Flow (Angebot → Test/Später → Auto-Follow-up)	PostOffer, HowTo, Cron
Phase 4	Supabase Auth + Rollen (Admin → Sales → Kunde)	Auth-System, 3 DB-Tabellen
Phase 5	Kostenanalyse + Pricing finalisieren	Recherche, kein Code
Phase 6	Stripe Integration (Live Account vorhanden)	3 API-Endpunkte, 2 DB-Tabellen
Phase 7	Usage-Tracking + Kunden-Dashboard	4 Dashboard-Pages, Usage-API