

Gado Imkerei

Website und Webshop für eine Schweizer Imkerei, von der Idee bis zum Zahlungs-Webhook

Die Gado Imkerei verkauft Honig aus eigener Ernte, direkt an Kundschaft in der Region. Aus dem Verkauf auf Zuruf wurde ein vollständiger Kaufweg: online oder vor Ort bestellen, per Karte, TWINT oder bar bezahlen, abholen oder liefern lassen. Konzept, Design, Frontend, Datenbank, Zahlung und Betrieb in einer Hand.

4

Wochen von Setup bis fertigem Shop

≈8'000

Zeilen TypeScript, SQL und CSS

4

Edge Functions (Deno) für Zahlung,
Meldung, Konto

3

Postgres-Tabellen mit Row Level Security

3

Bestellkanäle: Formular, WhatsApp,
E-Mail-Fallback

2

Audit-Funde, beide behoben

Gabriel Suter · Einzelumsetzung · Konzept, Design, Frontend, Backend, Zahlung, Betrieb

React 19 · TypeScript · Tailwind v4 · Supabase · Stripe · Vercel · Juli bis September 2026

kontakt@gabrielsuter.dev · gabrielsuter.dev · Kundin: Imkerin in Däniken SO, Direktverkauf von Blüten- und Waldhonig

1 Kurzfassung

Website und Webshop für eine Imkerei in Däniken SO. Kundschaft aus der Region findet Sorten, Preise und Abholung, bestellt mit oder ohne Konto und bezahlt online oder vor Ort. Die Imkerin erhält jede Bestellung per E-Mail und SMS und führt den Betrieb über eine geschützte Auswertung selbst.

Der Kern in einem Satz

Der Server glaubt dem Browser nichts: Betrag, Mailtext und Empfänger werden aus Datenbank und Katalog neu berechnet, und nur der signierte Stripe-Webhook darf «bezahlt» setzen.

2 Ausgangslage und Ziel

Die Kundin hatte weder Website noch Online-Verkauf. Honig wurde auf Zuruf verkauft, per Telefon und bei Abholung. Ziel war keine «schöne Imkerei-Seite», sondern eine lokal sichtbare Direktverkaufsseite, auf der in drei Sekunden klar ist, **wo** es Honig gibt, **was** er kostet und **warum** man dem Betrieb vertraut.

Aus dieser Positionierung («lokal, premium, direkt») ist die Seitenstruktur abgeleitet: zuerst der Kaufweg mit Preisen und Abholung, danach Person, Qualität und FAQ. Im Verlauf des Projekts entschied die Kundin, Versand und Online-Zahlung zu wollen. Damit wurde aus der One-Page-Site ein Shop mit Datenbank, Zahlungsanbieter und Meldewegen.

Meine Rolle

- **Strategie und Konzept:** Positionierung, Seitenstruktur nach Conversion-Logik, Entscheide mit der Kundin vorbereitet.
- **Design:** eigenes Farbsystem in OKLCH mit gemessenen Kontrasten, Typografie Fraunces/Geist, SVG-Illustrationen, responsive bis zur mobilen Sticky-Leiste.
- **Frontend:** React 19 mit TypeScript, Warenkorb, Kasse, Kontodialog, Admin-Auswertung.
- **Backend:** Postgres-Schema, Row-Level-Security-Policies, vier Edge Functions, Migrationen per CLI.
- **Integrationen:** Stripe Checkout mit Webhook und Erstattung, Resend für E-Mail, ASPSMS für SMS, WhatsApp als Bestellkanal.
- **Recht, SEO, Betrieb:** Rechtstexte nach Schweizer Recht, JSON-LD, Sitemap, Vercel-Deployment, Anleitungen für die Kundin.

3 Funktionsumfang

- **Shop:** Produktkarten mit Sorte, Jahrgang, Verfügbarkeit und Preis; Warenkorb im Browser; Kasse mit Abholung oder Versand, Wunschtermin oder Adressfeldern, Versandpauschale.
- **Zahlung:** online über Stripe (Karte, TWINT) oder bei Abholung (TWINT, bar); Betrag serverseitig nachgerechnet; Erstattungen per Webhook.
- **Konto:** Registrierung mit E-Mail und Passwort, Bestellhistorie, Passwort ändern, Konto löschen per Edge Function; Gastbestellung bleibt möglich.
- **Bestellkanäle:** Formular, WhatsApp mit dem Warenkorb als fertiger Nachricht, E-Mail-Fallback ohne Backend.
- **Benachrichtigung:** Bestellmeldung an Imkerei und Kundschaft per E-Mail, SMS an die Imkerei mit Direktlink; pro Bestellung garantiert nur einmal.

- **Admin-Auswertung:** geschützte Seite mit Umsatz, Sortenverteilung, Kanälen, Zahlungsstatus und Statuspflege je Bestellung.
- **Recht und SEO:** Impressum, Datenschutz, Bestellbedingungen; Lebensmittel-Deklaration vor dem Kauf; JSON-LD, Sitemap, OG-Bild, Seitentitel je Route.
- **Marketing und Go-live:** QR-Sticker auf den Gläsern zählen Scans; ein Schalter zeigt öffentlich «im Aufbau», bis die Kundin freigibt.

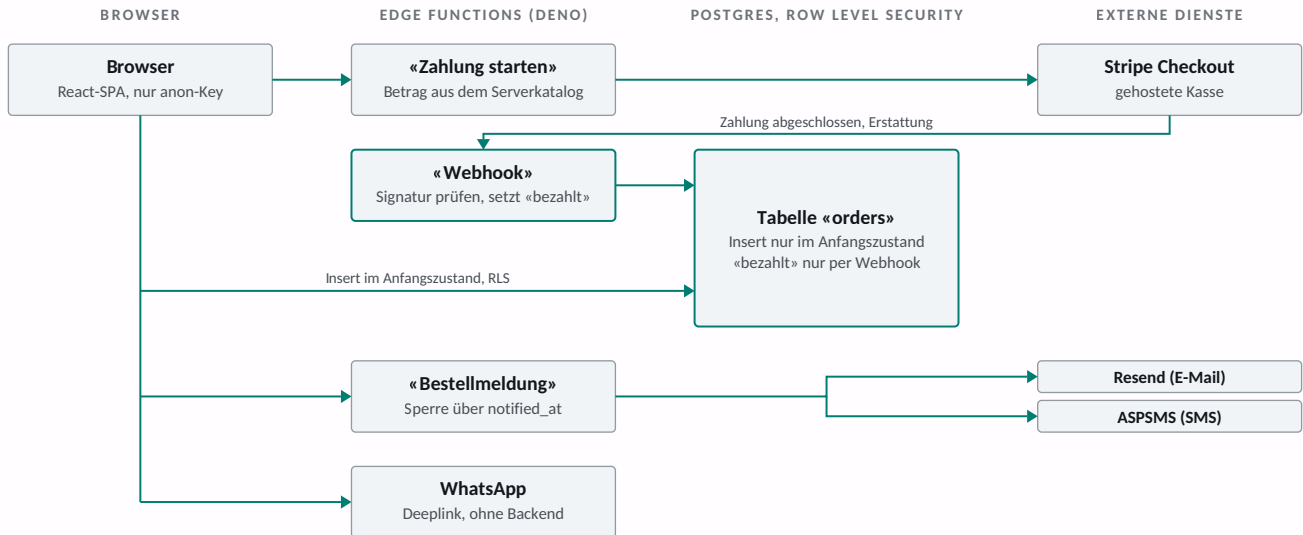
4 Architektur

Bereich	Technologie
Frontend	React 19, TypeScript (strict), Vite 8, Tailwind CSS v4, shadcn auf Base UI, lucide-react, Fontsource (Fraunces, Geist)
Backend	Supabase (Postgres, Row Level Security, Auth), Edge Functions in Deno/TypeScript, SQL-Migrationen über die Supabase CLI, Region Zürich
Zahlung und Kommunikation	Stripe Checkout (Karte, TWINT) mit signierten Webhooks und Erstattung, Resend (E-Mail, eigene Domain), ASPSMS (SMS), WhatsApp-Deeplinks
Qualität und Betrieb	ESLint, tsc als Build-Gate, eigene Prüfskripte vor jedem Build, Git/GitHub mit Feature-Banches, Vercel mit SPA-Rewrites, WebP-Bildpipeline

Schichten

Schicht	Aufgabe
Browser	React-SPA auf Vercel, Warenkorb in <code>localStorage</code> . Spricht nur mit dem öffentlichen anon-Key.
Supabase	Postgres mit den Tabellen <code>orders</code> , <code>admins</code> , <code>qr_scans</code> , Row Level Security auf jeder Tabelle, Auth mit eigenen Mail-Templates, vier Edge Functions in Deno.
Stripe	Gehostete Kasse, Webhooks, Erstattung.
Resend	E-Mail an Betrieb und Kundschaft.
ASPSMS	SMS an die Imkerin.

Systembild



Nur der signierte Webhook schreibt «bezahlt»; der Browser spricht die Datenbank ausschliesslich über den anon-Key und die Insert-Policy an.

Vier Grundsätze

- **Der Server glaubt dem Browser nichts.** Betrag, Mailtext und Empfänger werden serverseitig neu berechnet.
- **Zugriff über Row Level Security.** Gäste dürfen nur einfügen, Kundschaft liest nur Eigenes, Admins über eine security-definer-Funktion.
- **Graceful Degradation.** Ohne Supabase-Keys läuft die Seite weiter: Bestellungen gehen per WhatsApp oder E-Mail raus. Ohne SMS-Zugang wird der SMS-Weg still übersprungen.
- **Eine Quelle pro Wahrheit.** Inhalte in `site.ts`, Rechtliches in `legal.ts`, Serverkatalog in einer geteilten Deno-Datei. Offene Angaben blenden ihre Anzeige aus, statt als Platzhalter live zu gehen.

Edge Functions

Function	Aufgabe
Zahlung starten	Nimmt nur die Bestell-ID entgegen, rechnet den Betrag aus dem Serverkatalog nach und bricht bei Abweichung ab. Legt die gehostete Stripe-Kasse an.
Stripe-Webhook	Prüft die Signatur, setzt als Einziger «bezahlt», verarbeitet Erstattungen über die <code>order_id</code> in den Payment-Intent-Metadaten.
Bestellmeldung	E-Mail an Imkerei und Kundschaft, SMS an die Imkerei mit Direktlink auf die Bestellung. Pro Bestellung garantiert nur einmal.
Konto löschen	Löscht das eigene Konto serverseitig, weil der Browser das mit dem öffentlichen Schlüssel nicht darf.

Routen

Route	Zweck
Startseite	Kaufweg mit Preisen und Abholung, danach Person, Qualität und FAQ
Kasse	Abholung oder Versand, Zahlung online oder vor Ort, Bestellung mit oder ohne Konto
Admin-Auswertung	geschützt; Umsatz, Sortenverteilung, Kanäle, Zahlungsstatus, Statuspflege je Bestellung
Impressum, Datenschutz, Bestellbedingungen	drei Rechtsseiten aus einer einzigen Config, erreichbar auch vor dem Go-live

5 Technische Kernprobleme

Sechs Stellen, an denen das Projekt über «Formular speichert in Datenbank» hinausging.

SICHERHEIT

Bestellung als «bezahlt» einschleusen

Problem

Mit dem öffentlichen anon-Key liess sich per API eine Bestellung mit Status «bezahlt» und beliebigem Betrag anlegen. Die Datenbank antwortete mit 201.

Lösung

Die Insert-Policy verlangt jetzt einen definierten Anfangszustand: unbezahlt, ohne Zahlungsreferenz, ohne Meldezeitpunkt, Status offen oder neu. Nur der Stripe-Webhook setzt «bezahlt». Der Angriff bekommt 401, echte Bestellungen weiterhin 201.

Warum es zählt

Ein öffentlicher Schlüssel ist per Definition in jeder Hand. Sicherheit entsteht nicht durch Verstecken des Schlüssels, sondern dadurch, dass die Datenbank selbst weiss, welcher Anfangszustand erlaubt ist. Gefunden hat die Lücke der Test gegen die echte Instanz, nicht das Lesen des Codes.

ZAHLUNG**Bezahlen ohne Kartendaten im eigenen Code****Problem**

Kartendaten sollten den eigenen Code nie berühren, und der Betrag darf nicht aus dem Browser stammen, wo er manipulierbar wäre.

Lösung

Gehostetes Stripe Checkout per Redirect statt Stripe.js. Der Browser schickt nur die Bestell-ID, die Edge Function rechnet den Betrag aus dem Serverkatalog nach und bricht bei Abweichung ab. Eine Falle dabei: Session-Metadaten wandern nicht auf die Zahlung. Ohne `order_id` in den Payment-Intent-Metadaten hätte der Webhook Erstattungen nie einer Bestellung zuordnen können. Gefunden im Sandbox-Test, behoben vor dem ersten echten Franken.

Warum es zählt

Wer Kartendaten nie hält, hat keinen PCI-Umfang zu verteidigen, und ein serverseitig nachgerechneter Betrag lässt sich im Browser nicht fälschen. Die Metadaten-Falle zeigt, warum die Erstattung von Anfang an mitgetestet gehört: Eine Zahlung, die man nicht zurückordnen kann, ist im Betrieb wertlos.

NEBENLÄUFIGKEIT**Genau eine Meldung pro Bestellung****Problem**

Die Meldefunktion muss für Gäste erreichbar sein. Ohne Sperre könnte jeder beliebig oft Mails und SMS auslösen.

Lösung

Die Spalte `notified_at` ist die Sperre, gesetzt per bedingtem UPDATE (`where notified_at is null`), das die Datenbank atomar entscheidet. Gäste dürfen das Feld beim Einfügen nicht setzen, sonst könnte ein Angreifer die Meldung vorab abschalten und unbemerkt bestellen.

Warum es zählt

Zwei gleichzeitige Aufrufe können nicht beide gewinnen, wenn die Datenbank entscheidet; eine Sperre im Anwendungscode hätte genau diese Lücke. Und die zweite Hälfte zeigt, dass eine Sperre nur schützt, wenn niemand sie vorab umlegen kann.

BUILD**Zwei Laufzeiten, ein Katalog****Problem**

Deno kann nichts aus dem Frontend-Quellcode importieren. Preise, Versandkosten und Mailadressen müssen deshalb doppelt stehen, und der Server darf dem Browser keinen dieser Werte glauben.

Lösung

Ein Prüfskript vergleicht beide Seiten vor jedem Build und bricht bei Abweichung ab. Sonst fiele die Drift erst auf, wenn eine Zahlung fehlschlägt oder eine Bestellung in einem Postfach landet, in das niemand schaut.

Warum es zählt

Doppelt gehaltene Werte laufen auseinander; das ist keine Frage des Ob, sondern des Wann. Ein Build-Gate macht die Drift zum lauten Fehler im Editor statt zum stillen Fehler beim Kunden.

RECHT

Schweizer Recht statt deutschem Muster

Problem

In der Schweiz gibt es kein gesetzliches Widerrufsrecht für Online-Käufe. Lebensmittel-Pflichtangaben und der Säuglingshinweis müssen im Fernabsatz vor dem Kauf sichtbar sein, gesundheitsbezogene Aussagen zu Honig sind verboten. Ein deutsches AGB-Muster hätte alles drei falsch gemacht.

Lösung

Deklaration direkt bei den Produkten und an der Kasse, Rechtsseiten aus einer einzigen Config, erreichbar auch vor dem Go-live. Klärung von Rechtsform und MWST-Pflicht mit der Kundin.

Warum es zählt

Ein Rechtstext aus dem falschen Land ist schlimmer als keiner: Er verspricht Rechte, die es nicht gibt, und verschweigt Pflichten, die gelten. Bei Lebensmitteln kommt die Deklarationspflicht dazu, die nicht im Impressum, sondern vor dem Kaufknopf stehen muss.

DETAILS

Was man erst im Betrieb merkt

Problem und Lösung

Vier Punkte, die kein Test vorab zeigt, je mit der Behebung:

- **WhatsApp-Popup:** Das Fenster muss synchron im Klick-Handler öffnen, nach einem `await` blockieren Mobile-Browser das Popup. Lösung: WhatsApp öffnet vor dem ersten `await`, das Speichern läuft danach.
- **SMS-Länge:** Der Mittelpunkt «·» ist kein GSM-7-Zeichen und halbiert eine SMS auf 70 Zeichen. Lösung: Der Text wird vor dem Versand auf GSM-7 gebracht.
- **Kontrast:** Der Amber-Akzent erreichte auf Creme nur 2.9:1. Lösung: ein eigenes, dunkleres Token für Balken.
- **Umsatzauswertung:** Porto im Warenwert hätte die Auswertung aufgebläht. Lösung: Warenwert und Porto getrennt gespeichert.

Warum es zählt

Keiner dieser Punkte steht in einer Dokumentation. Man findet sie nur, indem man den Kaufweg auf dem Handy wirklich geht, den SMS-Text wirklich zählt und den Kontrast wirklich misst. Das ist der Unterschied zwischen fertig gebaut und fertig geprüft.

6 Datenmodell

Tabelle	Zweck und Besonderheit
<code>orders</code>	Jede Bestellung, aus dem Formular oder per WhatsApp. Gäste dürfen nur einfügen, und nur im definierten Anfangszustand (siehe Kernproblem Sicherheit). Warenwert und Porto sind getrennt gespeichert, damit die Umsatzauswertung nicht durch Versandkosten aufgebläht wird. <code>notified_at</code> ist zugleich die Sperre gegen Doppelmeldungen.
<code>admins</code>	Wer die Auswertung sehen und den Bearbeitungsstand setzen darf. Geprüft über eine security-definer-Funktion, nicht im Browser.
<code>qr_scans</code>	Zählt die Scans der QR-Sticker auf den Gläsern, nach Quelle.

7 Qualität und offene Punkte

Massnahme	Was sie leistet
Build-Gates	tsc, ESLint und zwei eigene Prüfskripte vor jedem Build. Weichen Preise oder Mailadressen zwischen Website und Server ab oder wird eine SMS zu lang, bricht der Build ab.
Test gegen die echten Dienste	Stripe-Sandbox inklusive Erstattung, Webhook-Signaturen, Resend-Zustellung, Policies mit dem anon-Key.
Schema als Migrationen	Über die Supabase CLI, nachvollziehbar statt Klicks im Dashboard.
Dokumentation für die Kundin	Setup-Anleitung, Einrichtung von E-Mail-Domain und SMS, SQL-Skripte für Admin-Freischaltung und Testdaten.
Git-Hygiene	Sprechende Commits auf Deutsch, Feature-Banches, keine Secrets im Repository, Env-Vars getrennt für lokal und Vercel.

Was der Audit gefunden hat

Eine offene Insert-Policy, über die sich Bestellungen als «bezahlt» einschleusen liessen, und ein Secret, das durch ein Leerzeichen statt eines Gleichheitszeichens zum Namen statt zum Wert geworden war und deshalb im Klartext lag. Beides behoben, der betroffene Schlüssel rotiert. Dass es hier steht, ist Absicht: Genau dafür ist ein Audit zum Schluss da.

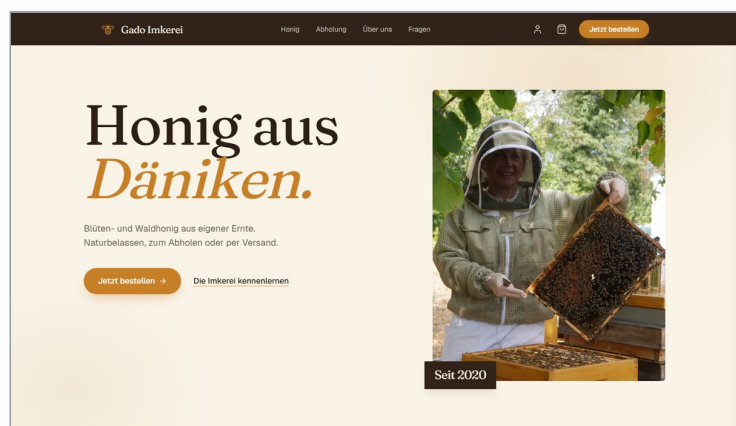
Abnahme

Stripe-Verifikation, Vercel-Konfiguration und Admin-Freischaltung sind erfolgt. Die Seite ist seit September 2026 öffentlich live.

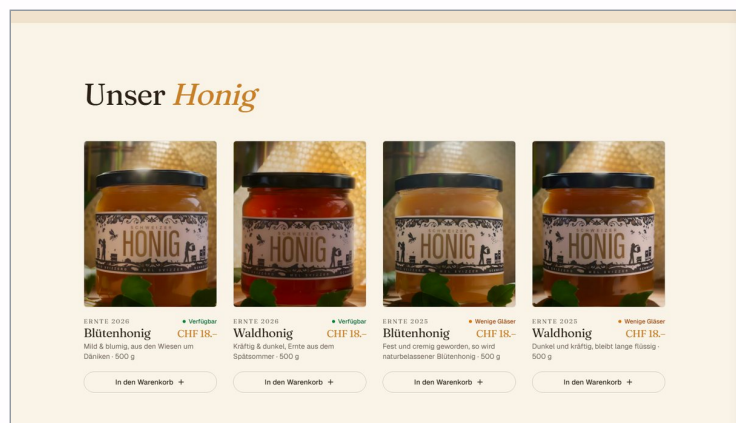
8 Einblicke



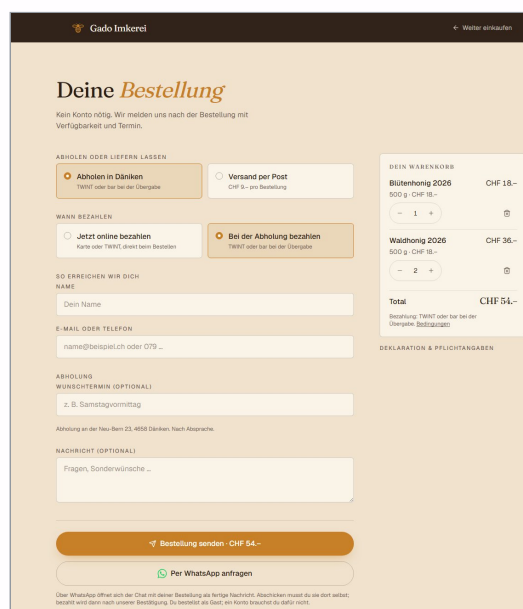
Mobile: Text und Button ohne Scrollen sichtbar und tippbar.



Hero: Ort, Nutzen, Kaufweg und Vertrauen in drei Sekunden, ein starkes Foto statt Slideshow.



Sorten mit Jahrgang, Verfügbarkeit und Preis, ein eigenes Foto pro Ernte.



Kasse: Abholung oder Versand, Zahlung online oder vor Ort, Bestellung mit oder ohne Konto.

9 Was das Projekt zeigt

Fähigkeit	Wo sie sichtbar wird
Strategie und Konzept	Positionierung «lokal, premium, direkt», daraus die Seitenstruktur nach Conversion-Logik; Entscheide mit der Kundin vorbereitet statt vorweggenommen
Design mit Messung	Eigenes Farbsystem in OKLCH, Kontraste gemessen statt geschätzt; der Amber-Akzent bekam ein eigenes Token, als 2.9:1 nicht reichte
Sicherheit an der richtigen Stelle	Row Level Security mit definiertem Anfangszustand, Betrag serverseitig nachgerechnet, Webhook-Signatur als einzige Quelle für «bezahlt»
Nebenläufigkeit in der Datenbank	Bedingtes UPDATE als atomare Sperre gegen Doppelmeldungen, statt einer Sperre im Anwendungscode
Fremde Dienste anbinden	Stripe Checkout mit Webhook und Erstattung, Resend, ASPSMS, WhatsApp-Deeplinks; jeder Weg fällt geordnet aus, wenn er fehlt
Recht als Fachdomäne	Schweizer Recht statt deutschem Muster: kein Widerrufsrecht, Lebensmittel-Deklaration vor dem Kauf, verbotene Gesundheitsaussagen
Prüfen statt annehmen	Build-Gates für doppelt gehaltene Werte, Tests gegen die echten Dienste, Sicherheitsaudit mit zwei realen Funden
Übergabe	Anleitungen und SQL-Skripte, mit denen die Kundin den Betrieb selbst führt

Stand	September 2026 · Live seit September 2026
Kontakt	Gabriel Suter · kontakt@gabrielsuter.dev · gabrielsuter.dev
Quellcode	Privates GitHub-Repository, Einsicht auf Anfrage
Live	gadoimkerei.ch