



web dev

DOKUMENTACJA TECHNICZNA I UŻYTKOWNIKA

web dev

System do automatycznego pozyskiwania klientów, generowania stron WWW dla lokalnych firm z pomocą Claude AI, zarządzania sprzedażą i automatycznego śledzenia przychodów.

web dev znajduje w Google Maps firmy, które nie mają strony internetowej, w ok. 30 sekund tworzy dla każdej z nich gotową stronę demo, wysyła właścicielowi spersonalizowaną ofertę e-mail lub SMS z linkiem do podglądu i płatności, publikuje opłaconą stronę pod domeną klienta, a wpłaty ze Stripe i z konta bankowego automatycznie trafiają do finansów i kalendarza sprzedaży.

Spis treści

Wprowadzenie

Część I — Dokumentacja użytkownika

- 1.1 Rejestracja i logowanie
- 1.2 Konfiguracja integracji (klucze API)
- 1.3 Konfiguracja skrzynki e-mail
- 1.4 Wyszukiwanie firm (lead generation)
- 1.5 Generowanie stron demo (Claude AI)
- 1.6 Podgląd, publikacja i pobieranie kodu
- 1.7 Własne domeny klientów
- 1.8 Outreach — wysyłka ofert e-mail i SMS
- 1.9 Śledzenie przychodów i finanse
- 1.10 Kalendarz sprzedaży
- 1.11 Pomoc i wsparcie
- 1.12 Ustawienia profilu

Część II — Dokumentacja techniczna

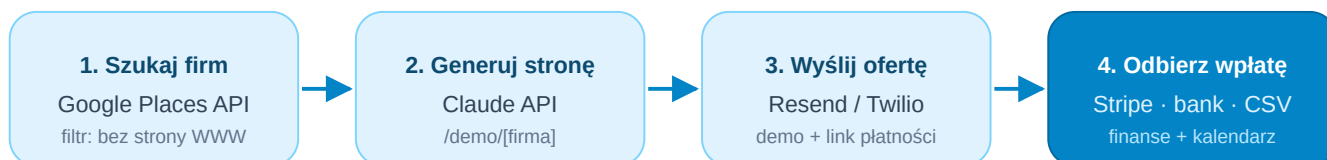
- 2.1 Stos technologiczny
- 2.2 Architektura
- 2.3 Struktura plików
- 2.4 Schemat bazy danych (Prisma)
- 2.5 Zmienne środowiskowe (.env)
- 2.6 Uruchomienie lokalne
- 2.7 Deployment (Vercel)
- 2.8 Integracje z API zewnętrznymi
- 2.9 Bezpieczeństwo
- 2.10 Plan wdrożenia krok po kroku
- 2.11 Rozwiązywanie problemów

Załącznik A — [schema.prisma](#)



Wprowadzenie — jak działa web dev

web dev automatyzuje cały proces sprzedaży prostych stron internetowych lokalnym firmom — od znalezienia potencjalnego klienta po zaksięgowaną wpłatę. Agencja lub freelancer pracuje w jednym panelu, a powtarzalne czynności (wyszukiwanie, copywriting, projekt, wysyłka, pilnowanie płatności) wykonuje system.



Opublikuj → produkcyjny URL + domena klienta · Pobierz kod → ZIP (HTML/CSS/JS lub Next.js)

Dla kogo

Freelancerzy i agencje webowe sprzedające strony wizytówkowe firmom usługowym: fryzjerom, warsztatom, gabinetom, restauracjom, rzemieślnikom.

Co automatyzuje

Wyszukiwanie leadów, tworzenie treści i projektu strony, pisanie ofert, wysyłkę e-mail/SMS, hosting, domeny, rejestrowanie płatności i kalendarz.

Model danych

Lead (firma) → Site (strona demo) → OutreachMessage (oferta) → Payment (płatność z kodem WD-XXXXXX) → BankTransaction / Stripe.

Interfejs

Nowoczesny glassmorphism, tryb jasny i ciemny, animowane tło z poświatą, płynne przejścia Framer Motion, pełna responsywność.

Główne moduły panelu

Moduł	Adres	Opis
Pulpit	/dashboard	Suma zarobków na żywo, KPI lejka, wykres przychodów, aktywność
Szukaj firm	/dashboard/leads	Google Places + zarządzanie leadami
Strony	/dashboard/sites	Podgląd · Opublikuj · Pobierz kod, domeny, płatność
Outreach	/dashboard/outreach	Wiadomości Claude, wysyłka i harmonogram
Finanse	/dashboard/finance	Stripe, Open Banking, CSV, wykresy, płatności
Kalendarz	/dashboard/calendar	Wpłaty, terminy, wysyłki, wydarzenia
Pomoc	/dashboard/help	Samouczki, FAQ, zgłoszenia
Ustawienia	/dashboard/settings	Profil, integracje, domeny, szablony

Część I

Dokumentacja użytkownika

1.1 Rejestracja i logowanie

Konto agencji zakładasz na stronie </register>. Weryfikacja adresu e-mail jest dwuetapowa:

- 1 Podaj imię, (opcjonalnie) nazwę agencji, e-mail i hasło (min. 8 znaków, wielka litera i cyfra). Podczas wpisywania adresu serwer sprawdza rekordy **MX** domeny — literówki typu *gmial.com* są odrzucane od razu.
- 2 Na podany adres wysyłany jest **6-cyfrowy kod OTP** (ważny 10 minut, maks. 5 prób). Wpisz go na stronie </verify> — możesz też wkleić cały kod naraz.
- 3 Zaloguj się hasłem lub wybierz **Magic Link** — jednorazowy link logowania wysyłany przez Resend.

Bez potwierdzenia adresu logowanie hasłem jest zablokowane. Kod można wysłać ponownie po 45 sekundach.

1.2 Konfiguracja integracji (klucze API)

W **Ustawienia → Integracje** każda agencja wprowadza własne klucze. Są one szyfrowane (AES-256-GCM) przed zapisem, a w interfejsie widoczny jest wyłącznie skrót, np. `sk-ant-...a1b2`. Po zapisaniu system wykonuje test połączenia i oznacza integrację jako *połączono*.

Integracja	Do czego służy	Minimum do startu
Anthropic Claude	Treść stron demo i wiadomości outreach	tak
Google Places API (New)	Wyszukiwanie firm	tak
Resend	Wysyłka e-maili z Twojej domeny	do e-maili
Twilio	Wysyłka SMS	do SMS
Stripe	Linki płatności + webhook	do płatności online
GoCardless	Automatyczne pobieranie przelewów (Open Banking)	opcjonalnie
Vercel / Cloudflare	Hosting stron i domeny klientów	opcjonalnie

Jeśli administrator skonfigurował klucze globalne w `.env`, integracja ma status *klucz globalny* — Twój własny klucz zawsze ma pierwszeństwo.

1.3 Konfiguracja skrzynki e-mail

- 1 Załóż konto w **Resend** i dodaj domenę wysyłkową — najlepiej subdomenę, np. `oferty.twojaagencja.pl`, aby nie narażać reputacji głównej domeny.
- 2 U rejestratora domeny dodaj rekordy **SPF**, **DKIM** i zalecany **DMARC** wskazane przez Resend. Poczekaj na status *Verified*.
- 3 W web dev: **Ustawienia** → **Integracje** → **Resend** — wklej klucz `re_...` i adres nadawcy.
- 4 W **Ustawienia** → **Szablony** ustaw nazwę nadawcy, adres Reply-To (odpowiedzi klientów trafią do Twojej skrzynki), stopkę z danymi firmy i informacją o rezygnacji.
- 5 Wyślij testową wiadomość do siebie z modułu Outreach i sprawdź, czy nie trafia do spamu.

1.4 Wyszukiwanie firm (lead generation)

Zakładka **Szukaj firm** korzysta z Google Places API (Text Search).

- 1 Wpisz **branżę** (np. *fryzjer*, *mechanik* — możesz kliknąć podpowiedź) i **lokalizację** (miasto lub dzielnicę).
- 2 Wybierz liczbę wyników (10–60) i pozostaw włączony filtr **Tylko bez strony**. Profile na Facebooku, Instagramie czy Booksy nie są traktowane jako strona WWW.
- 3 Kliknij **Szukaj**. Dla każdej firmy zapisywane są: nazwa, telefon, adres, godziny otwarcia, ocena, liczba opinii, link do Map Google oraz — jeśli został wykryty — e-mail (weryfikowany rekordami MX).
- 4 Otwórz **Szczegóły**, aby uzupełnić e-mail, zmienić status (Nowy → Strona gotowa → Wyślano ofertę → Odpowiedział → Klient) i dodać notatki.

Firmy już zapisane na koncie są pomijane (unikalność `userId + placeId`), więc wyszukiwanie można bezpiecznie powtarzać.

1.5 Generowanie stron demo (Claude AI)

- 1 Na karcie firmy kliknij **Generuj stronę**.
- 2 Wybierz styl (nowoczesny, elegancki, wyrazisty, minimalny), opcjonalnie dodaj wskazówki (np. „podkreśl parking”) i cenę — domyślna pochodzi z ustawień.
- 3 Claude tworzy strukturę i treść: SEO, sekcję hero, O nas, usługi z ikonami, Dlaczego my, FAQ i kontakt oraz dobiera kolory do branży. Dane faktów (telefon, adres, godziny, ocena) pochodzą wyłącznie z Google — model nie wymyśla opinii ani cen.
- 4 Po ok. 20–40 s zostaniesz przeniesiony na stronę szczegółów. Publiczny link demo ma postać `/demo/nazwa-firmy` i zawiera pasek z ceną oraz przyciskiem „Kup tę stronę”.

1.6 Podgląd, publikacja i pobieranie kodu

Przy każdej stronie znajdują się trzy główne przyciski:

Przycisk	Działanie
[Podgląd]	Interaktywny podgląd w ramce desktop / tablet / smartfon (skalowany iframe) z możliwością otwarcia w nowej karcie.
[Opublikuj]	Jeśli skonfigurowano Vercel — tworzy deployment produkcyjny (osobny projekt <code>wd-<slug></code>) i zwraca adres <code>https://wd-<slug>.vercel.app</code> . Bez Vercel strona jest hostowana w web dev pod <code>https://<slug>.<SITES_ROOT_DOMAIN></code> .
[Pobierz kod]	Archiwum ZIP generowane w przeglądarce (JSZip + file-saver): HTML/CSS/JS gotowe na dowolny hosting lub projekt Next.js (App Router, treść w <code>content/site.json</code>).

1.7 Własne domeny klientów

- 1 Opublikuj stronę, a następnie w sekcji **Własna domena klienta** wpisz domenę (np. `www.salon-ania.pl`) i wybierz dostawcę (Vercel lub Cloudflare).
- 2 System wyświetli rekordy DNS do ustawienia u rejestratora: `CNAME www → cname.vercel-dns.com` dla subdomeny albo `A @ → 76.76.21.21` dla domeny głównej, ewentualnie rekord TXT weryfikacji.
- 3 Kliknij **Sprawdź**. Po propagacji DNS (zwykle 5–60 minut) status zmieni się na *aktywna*, certyfikat SSL zostanie wystawiony automatycznie, a adres produkcyjny strony zaktualizuje się.

1.8 Outreach — wysyłka ofert e-mail i SMS

- 1 W zakładce **Outreach** wybierz firmę (stronę), kanał (E-mail / SMS) i ton (przyjazny, profesjonalny, konkretny).
- 2 Kliknij **Napisz wiadomość z Claude**. System automatycznie tworzy płatność z kodem `WD-XXXXXX` i — gdy Stripe jest skonfigurowany — link do płatności. Wiadomość zawiera link do demo i link do płatności.
- 3 Popraw treść według uznania. Szablon z ustawień (zmienne `{{firma}}`, `{{link_demo}}`, `{{link_platnosci}}`, `{{cena}}`) służy Claude jako wzór stylu.
- 4 Wyślij od razu albo zaznacz **Zaplanuj wysyłkę** i wybierz termin — zadanie cron wysyła zaplanowane wiadomości co 10 minut, a termin pojawia się w kalendarzu.
- 5 W historii wysyłek oznaczaj odpowiedzi klientów — status leada zmieni się na *Odpowiedział*.

Kontaktujesz firmy na dane publiczne (B2B). Zadbaj o podstawę prawną, pełne dane nadawcy i obsługę rezygnacji (np. „odpisz STOP”). W razie wątpliwości skonsultuj się z prawnikiem.

1.9 Śledzenie przychodów i finanse

Moduł **Finanse** automatycznie zlicza wpłaty z trzech źródeł i odświeża **łącną sumę zarobków w czasie rzeczywistym** (Server-Sent Events).

Stripe

- 1 Wklej klucz `sk_live_...` w Integracjach.
- 2 W Stripe Dashboard → Developers → Webhooks dodaj endpoint pokazany w Integracjach: `/api/webhooks/stripe/<Twoje ID>` i zdarzenia `checkout.session.completed`, `checkout.session.async_payment_succeeded`, `checkout.session.async_payment_failed`, `charge.refunded`.
- 3 Skopiuj *Signing secret* (`whsec_...`) do Integracji. Od teraz każda opłacona strona zmienia status na *opłacona*, lead na *Klient*, a Ty dostajesz powiadomienie.

Przelewy — Open Banking (automatycznie)

- 1 Wprowadź klucze GoCardless Bank Account Data w Integracjach.
- 2 W Finansach kliknij **Połącz bank**, wybierz bank i zaloguj się w nim, aby udzielić zgody (tylko odczyt, 90 dni).
- 3 Transakcje pobierane są od razu i co 6 godzin (cron). Przelewy z kodem `WD-XXXXXX` w tytule są dopasowywane z pewnością; pozostałe — po kwocie i nazwie firmy.

Przelewy — import CSV (półautomatycznie)

- 1 Pobierz wyciąg CSV z bankowości internetowej (PKO BP, mBank, ING, Santander, Pekao, Millennium lub dowolny).
- 2 Kliknij **Import CSV** i wybierz plik. Kodowanie (UTF-8 / Windows-1250), separator i kolumny wykrywane są automatycznie; w razie potrzeby wskaż kolumny daty i kwoty ręcznie.
- 3 Duplikaty są pomijane, a dopasowane wpłaty oznaczane jako opłacone.

Dashboard finansowy

Karta „Łączne zarobki” (na żywo), wpływy w tym miesiącu, kwoty oczekujące i po terminie, wykres słupkowy miesięczny (12 mies.) i tygodniowy (12 tyg.) z porównaniem *zrealizowane* vs *oczekujące* (z widokiem tabeli), lista płatności z akcjami „oznacz jako opłaconą” i „anuluj” oraz ręczne dodawanie płatności.

1.10 Kalendarz sprzedaży

Interaktywny kalendarz miesięczny automatycznie nanosi: **zaksięgowane wpłaty** (Stripe i bank), **terminy płatności** (pomarańczowe; po terminie — czerwone), **zaplanowane i wysłane oferty** oraz własne wydarzenia (follow-up, spotkanie). Kliknij dzień, aby zobaczyć szczegóły w panelu bocznym; filtry pozwalają ukryć wybrane typy. Nagłówek pokazuje sumę wpływów i oczekiwanych płatności w danym miesiącu.

1.11 Pomoc i wsparcie

Zakładka **Pomoc** zawiera wyszukiwarkę bazy wiedzy, samouczki krok po kroku (szukanie firm, konfiguracja e-mail, podpinanie domen, płatności), FAQ w formie rozwijanych akordeonów oraz formularz zgłoszeniowy. Zgłoszenie jest zapisywane w bazie i wysyłane e-mailem na adres `ADMIN_EMAIL` z ustawionym Reply-To na adres użytkownika.

1.12 Ustawienia profilu

Zakładka	Zawartość
Profil	Imię, nazwa agencji, avatar (zmniejszany w przeglądarce do 256×256), zmiana e-maila z potwierdzeniem kodem OTP na nowy adres.
Hasło	Zmiana hasła z weryfikacją obecnego.
Powiadomienia	Nowa wpłata, odpowiedź klienta, raport tygodniowy.
Integracje	Klucze API: Anthropic, Google Places, Resend, Twilio, Stripe, GoCardless, Vercel, Cloudflare + URL webhooka Stripe.
Domeny	Lista wszystkich domen klientów i ich statusów.
Szablony	Nadawca, Reply-To, domyślna cena, IBAN, szablony e-mail/SMS, stopka e-mail.

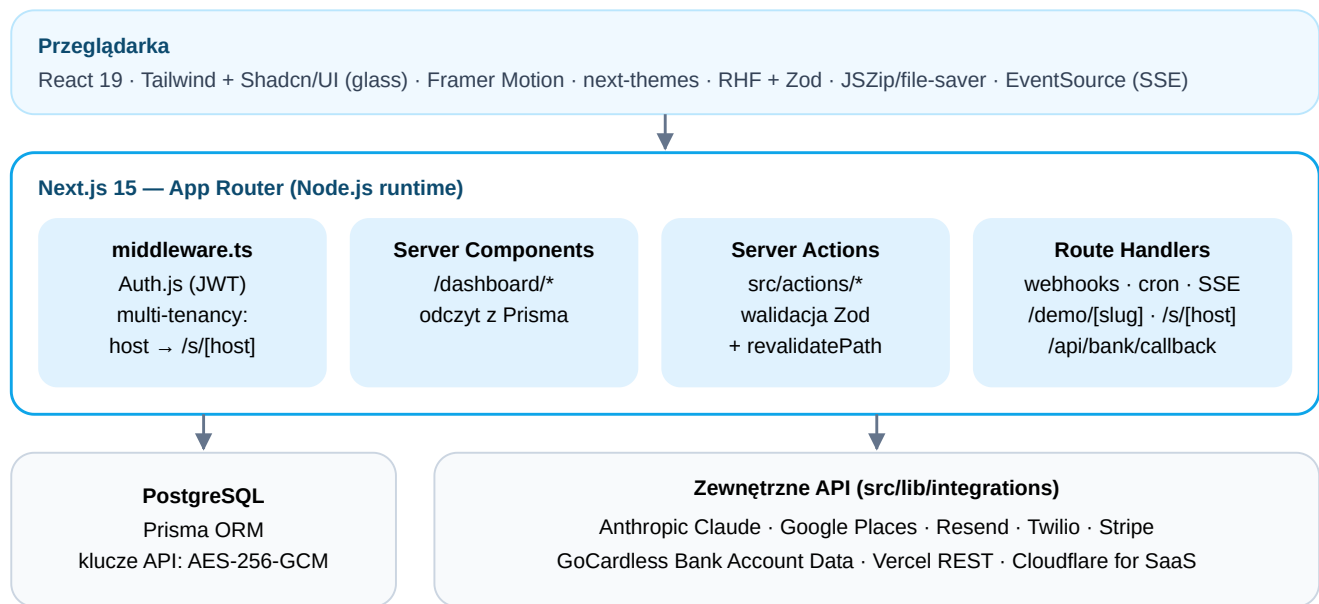
Część II

Dokumentacja techniczna

2.1 Stos technologiczny

Warstwa	Technologia
Framework	Next.js 15 (App Router, Server Components, Server Actions), React 19, TypeScript (strict)
UI	Tailwind CSS 3, komponenty w stylu Shadcn/UI (Radix) z wariantami glass, lucide-react, sonner
Animacje i motyw	Framer Motion (przejścia stron, modale, sprzężujące przyciski, animowane tło), next-themes
Formularze	React Hook Form + Zod (te same schematy na kliencie i serwerze)
Dane	PostgreSQL + Prisma ORM 6
Auth	Auth.js v5 (Credentials + Magic Link przez Resend), sesje JWT, bcrypt
AI	Anthropic Claude (@anthropic-ai/sdk) — tool use z wymuszonym schematem JSON
Płatności	Stripe SDK (Payment Links, Webhooks), GoCardless Bank Account Data, PapaParse (CSV)
Komunikacja	Resend (e-mail), Twilio (SMS)
Hosting stron	Vercel REST API (deploymenty, domeny), Cloudflare for SaaS (opcjonalnie), wbudowany multi-tenant
Eksport	JSZip + file-saver (ZIP w przeglądarce)

2.2 Architektura



Zasady: odczyty wykonują Server Components bezpośrednio przez Prisma; wszystkie mutacje przechodzą przez Server Actions (`src/actions/`), które walidują dane Zod-em, sprawdzają właściciela rekordu (`where: { id, userId }`) i wywołują `revalidatePath` . Integracje zewnętrzne są odizolowane w `src/lib/integrations` i pobierają klucze przez `getCredential()` — najpierw klucz użytkownika, potem fallback z `.env` .

Multi-tenancy: `middleware.ts` rozpoznaje host. Żądania do domen innych niż aplikacja (domena klienta lub `<slug>.SITES_ROOT_DOMAIN`) są przepisywane na `/s/[host]` , który renderuje opublikowaną stronę z bazy. **Czas rzeczywisty:** `/api/finance/stream` (SSE) co 4 s porównuje sumy i wysyła zdarzenie tylko przy zmianie.

Renderowanie stron: Claude zwraca wyłącznie strukturę `SiteContent` (JSON, walidowany Zod). HTML generuje deterministyczny renderer `src/lib/site/render.ts` — to samo źródło służy do podglądu, demo, hostingu, deploymentu Vercel i eksportu ZIP. Dzięki temu model nie może wstrzyknąć skryptów, a wszystkie teksty są escapowane.

2.3 Struktura plików

```
web-dev/
├─ prisma/
│  └─ schema.prisma          # model danych
│  └─ seed.ts                # dane demo (demo@webdev.app / Demo1234)
├─ public/ logo.svg · logo-mark.svg
├─ docs/  web-dev-dokumentacja.pdf · build_docs.py
├─ vercel.json                # zadania cron
└─ src/
   └─ middleware.ts          # auth + multi-tenancy (własne domeny)
   └─ auth.ts · auth.config.ts
   └─ actions/               # Server Actions (auth, leads, sites, outreach,
                              #   finance, calendar, settings, support)
```

```

├─ app/
│  ├─ page.tsx          # landing page
│  ├─ (auth)/           # login · register · verify (OTP)
│  ├─ dashboard/        # pulpit, leads, sites/[id], outreach,
│  │                       #   finance, calendar, help, settings
│  ├─ demo/[slug]/      # publiczne demo strony
│  ├─ s/[host]/          # hosting stron pod domeną klienta
│  └─ api/               # auth, webhooks/stripe/[userId],
│                        #   bank/callback, finance/stream, cron/*
├─ components/          # ui/ (glass Shadcn), landing/, dashboard/,
│                        #   sites/, finance/, calendar/, help/, settings/
├─ hooks/               # use-live-revenue, use-email-check, ...
└─ lib/
   ├─ integrations/     # claude, places, resend, twilio, stripe,
   │                       #   gocardless, vercel, cloudflare
   ├─ site/              # types, json-schema, render, export
   ├─ validations.ts     # schematy Zod (klient + serwer)
   ├─ email-verify.ts    # MX lookup (dns/promises)
   ├─ otp.ts · crypto.ts · credentials.ts
   ├─ payments.ts        # dopasowanie przelewów
   └─ finance.ts · calendar.ts · csv.ts

```

2.4 Schemat bazy danych (Prisma)

Pełny plik `schema.prisma` znajduje się w załączniku A. Najważniejsze modele:

Model	Opis	Kluczowe relacje / indeksy
User, Account, Session, VerificationToken	Konta i sesje Auth.js	email unique
OtpCode	Hash kodu OTP, licznik prób, wygaśnięcie	(email, purpose)
UserSettings	Powiadomienia, nadawca, szablony, cena, IBAN	1-1 z User
ApiCredential	Zaszyfrowany JSON z kluczami integracji	unique (userId, provider)
SearchQuery, Lead	Wyszukiwania i firmy z Google Places	unique (userId, placelId), slug unique
Site, Domain	Strony (treść JSON, status, hosting) i domeny klientów	slug unique, hostname unique
OutreachMessage	E-mail/SMS, status, termin wysyłki	(userId, status, scheduledAt)
Payment	Płatność z kodem WD-XXXXXX, źródło, status, Stripe ID	reference unique, (userId, paidAt)
BankConnection, BankTransaction	Zgody Open Banking i transakcje (API/CSV)	unique (userId, externalId)
StripeEvent	Idempotencja webhooków	id = evt_...
CalendarEvent, SupportTicket	Własne wydarzenia i zgłoszenia	(userId, date)

Kwoty są przechowywane w **groszach** (Int), co eliminuje błędy zaokrągleń.

2.5 Zmienne środowiskowe (.env)

Skopiuj `.env.example` do `.env`. Klucze oznaczone „per-user” mogą być też podane przez każdą agencję w Ustawieniach — wartości z `.env` pełnią rolę globalnego fallbacku.

Zmienna	Opis	Wymagana
<code>NEXT_PUBLIC_APP_URL</code>	Publiczny adres aplikacji (linki demo, webhooki, redirecty).	tak
<code>NEXT_PUBLIC_SITES_ROOT_DOMAIN</code>	Domena dla stron hostowanych wewnętrznie: <code><slug></code> . <code><domena></code> (wildcard DNS).	zalecane
<code>ADMIN_EMAIL</code>	Adres, na który trafiają zgłoszenia z Centrum Pomocy.	zalecane
<code>DATABASE_URL</code>	Connection string PostgreSQL.	tak
<code>AUTH_SECRET</code>	Sekret Auth.js do podpisywania sesji JWT.	tak
<code>AUTH_TRUST_HOST</code>	Wymagane za proxy / na Vercel.	zalecane
<code>ENCRYPTION_KEY</code>	32-bajtowy klucz (base64) do szyfrowania kluczy API użytkowników (AES-256-GCM).	tak
<code>CRON_SECRET</code>	Token autoryzujący zadania cron (nagłówek Authorization: Bearer).	zalecane
<code>ANTHROPIC_API_KEY</code>	Globalny klucz Claude (fallback, gdy użytkownik nie podał własnego).	per-user
<code>ANTHROPIC_MODEL</code>	Model Claude używany do generowania.	per-user
<code>GOOGLE_PLACES_API_KEY</code>	Klucz Google Places API (New).	per-user
<code>RESEND_API_KEY</code>	Klucz Resend dla poczty systemowej (OTP, Magic Link, zgłoszenia).	zalecane
<code>EMAIL_FROM</code>	Nadawca poczty systemowej (zweryfikowana domena w Resend).	zalecane
<code>TWILIO_ACCOUNT_SID</code>	Twilio — identyfikator konta.	per-user
<code>TWILIO_AUTH_TOKEN</code>	Twilio — token.	per-user
<code>TWILIO_FROM_NUMBER</code>	Numer nadawcy SMS w formacie E.164.	per-user
<code>STRIPE_SECRET_KEY</code>	Klucz tajny Stripe.	per-user
<code>STRIPE_WEBHOOK_SECRET</code>	Sekret podpisu webhooka (whsec_...).	per-user
<code>GOCARDLESS_SECRET_ID</code>	GoCardless Bank Account Data — secret ID.	per-user
<code>GOCARDLESS_SECRET_KEY</code>	GoCardless Bank Account Data — secret key.	per-user
<code>VERCEL_TOKEN</code>	Token Vercel REST API (deploymenty, domeny).	per-user
<code>VERCEL_TEAM_ID</code>	ID zespołu Vercel (opcjonalnie).	per-user
<code>VERCEL_PROJECT_ID</code>	ID projektu tej aplikacji — do niego dodawane są domeny klientów przy hostingu wewnętrznym.	per-user

CLOUDFLARE_API_TOKEN	Token Cloudflare (Custom Hostnames).	per-user
CLOUDFLARE_ZONE_ID	ID strefy Cloudflare for SaaS.	per-user

```
openssl rand -base64 32 # AUTH_SECRET
openssl rand -base64 32 # ENCRYPTION_KEY (dokładnie 32 bajty)
```

2.6 Uruchomienie lokalne

```
git clone <repo> web-dev && cd web-dev
npm install # postinstall: prisma generate
cp .env.example .env # uzupełnij DATABASE_URL, AUTH_SECRET, ENCRYPTION_KEY
docker run -d --name pg -e POSTGRES_PASSWORD=postgres -p 5432:5432 postgres:16
npm run db:push # lub: npm run db:migrate
npm run db:seed # konto demo@webdev.app / Demo1234
npm run dev # http://localhost:3000
```

Bez `RESEND_API_KEY` kody OTP są wypisywane w konsoli serwera — wygodne w trybie deweloperskim.
 Hosting multi-tenant lokalnie: dodaj do `/etc/hosts` wpis `127.0.0.1 moja-firma.sites.localhost`.

2.7 Deployment (Vercel)

- 1 Utwórz bazę PostgreSQL (np. Neon, Supabase, Vercel Postgres) i ustaw `DATABASE_URL`.
- 2 Zaimportuj repozytorium w Vercel. Build command: `npm run build` (zawiera `prisma generate`). Uruchom migracje: `npx prisma migrate deploy`.
- 3 Ustaw zmienne środowiskowe z sekcji `.env`, w tym `NEXT_PUBLIC_APP_URL`, `CRON_SECRET` i `VERCEL_PROJECT_ID` (ID tego projektu).
- 4 Zadania cron z `vercel.json` uruchamiają się automatycznie: `/api/cron/outreach` (co 10 min), `/api/cron/bank-sync` (co 6 h), `/api/cron/overdue` (codziennie).
- 5 Hosting wbudowany: dodaj do projektu domenę wildcard `*.sites.twojadomena.pl` (wymaga nameserverów Vercel) i ustaw `NEXT_PUBLIC_SITES_ROOT_DOMAIN`.
- 6 Zweryfikuj domenę nadawcy w Resend (`EMAIL_FROM`) i skonfiguruj webhooki Stripe per użytkownik.

2.8 Integracje z API zewnętrznymi

API	Plik	Wykorzystanie
Anthropic Claude	lib/integrations/claude.ts	<code>messages.create</code> z narzędziem <code>build_site</code> / <code>compose_message</code> i <code>tool_choice</code> wymuszającym narzędzie; wynik walidowany Zod.
Google Places (New)	places.ts	<code>POST places:searchText</code> z <code>X-Goog-FieldMask</code> , do 3 stron x 20 wyników; odrzucane firmy nieczynne; wykrywanie e-maila ze strony.

Resend	resend.ts	Poczta systemowa (klucz globalny) i outreach (klucz agencji), tagi <code>message_id</code> .
Twilio	twilio.ts	<code>messages.create</code> ; normalizacja numerów do E.164 (+48).
Stripe	stripe.ts, api/webhooks/stripe/[userId]	Product + Price + Payment Link z metadanymi <code>paymentId</code> ; weryfikacja podpisu <code>constructEvent</code> ; idempotencja przez <code>StripeEvent</code> .
GoCardless BAD	gocardless.ts, api/bank/callback	token → end-user agreement → requisition → konta → <code>/transactions</code> ; interfejs <code>BankProvider</code> umożliwia podmiannę dostawcy (np. Enable Banking, Salt Edge).
Vercel REST	vercel.ts	<code>POST /v13/deployments</code> (pliki inline), <code>/v10/projects/:id/domains</code> , <code>/v6/domains/:d/config</code> , weryfikacja.
Cloudflare	cloudflare.ts	Custom Hostnames (Cloudflare for SaaS) jako alternatywa dla Vercel Domains.

Dostępność i warunki dostawców Open Banking zmieniają się — przed wdrożeniem produkcyjnym sprawdź, czy GoCardless Bank Account Data przyjmuje nowych klientów w Twoim regionie; w razie potrzeby zaimplementuj interfejs `BankProvider` dla innego AISP.

2.9 Bezpieczeństwo

- Klucze API użytkowników szyfrowane **AES-256-GCM** (`ENCRYPTION_KEY`), nigdy nie są zwracane do przeglądarki.
- Hasła: **bcrypt** (cost 12). OTP przechowywane jako hash SHA-256, porównanie `timingSafeEqual`, limit prób i cooldown.
- Walidacja Zod na kliencie i serwerze; każde zapytanie mutujące filtruje po `userId` (izolacja danych agencji).
- Webhooki Stripe weryfikowane podpisem, cron chroniony `CRON_SECRET`.
- Generowane strony: treść modelu jest danymi, nie kodem — renderer escapuje wszystkie teksty; demo ma `noindex`.
- Weryfikacja e-maili: składnia → lista domen tymczasowych → rekordy MX → OTP.

2.10 Plan wdrożenia krok po kroku

Etap	Zakres	Rezultat
1. Fundament	Next.js + Tailwind + motyw glass, Prisma + PostgreSQL, Auth.js, rejestracja z MX + OTP	Logowanie, pusty panel
2. Ustawienia	Profil, hasło, powiadomienia, szyfrowane integracje, szablony	Agencja podpiną własne klucze
3. Lead generation	Google Places, filtr braku strony, wykrywanie e-maila, zarządzanie leadami	Lista firm do kontaktu

4. AI builder	Schemat SiteContent, Claude tool use, renderer HTML, /demo/[slug], podgląd urządzeń	Strona demo w 30 s
5. Publikacja	Vercel Deployments, hosting multi-tenant, domeny (Vercel/Cloudflare), eksport ZIP	Strona online pod domeną klienta
6. Outreach	Claude messaging, Resend, Twilio, planowanie + cron	Wysłane oferty
7. Finanse	Stripe Payment Links + webhooki, Open Banking, import CSV, dopasowanie, SSE, wykresy	Automatyczne śledzenie przychodów
8. Kalendarz i pomoc	Kalendarz sprzedaży, Help Center, zgłoszenia	Pełny proces sprzedaży
9. Produkcja	Migracje, cron, monitoring, testy E2E (Playwright), limity planów	Wersja 1.0

2.11 Rozwiązywanie problemów

Objaw	Przyczyna / rozwiązanie
„Brak skonfigurowanej integracji”	Dodaj klucz w Ustawienia → Integracje lub zmienną w <code>.env</code> .
Google Places: <i>API not enabled</i>	W Google Cloud włącz Places API (New) i podepnij płatności.
E-maile trafiają do spamu	Sprawdź SPF/DKIM/DMARC w Resend, używaj subdomeny, unikaj linków skracających.
Webhook Stripe zwraca 400	Nieprawidłowy <code>whsec_</code> albo endpoint innego użytkownika — skopiuj URL z Integracji.
Domena „oczekuje na DNS”	Sprawdź rekordy u rejestratora; propagacja może trwać do 48 h. Usuń stare rekordy A/AAAA.
Import CSV nie rozpoznaje kolumn	Wskaż kolumny ręcznie w oknie importu; upewnij się, że plik zawiera nagłówki.
Suma zarobków „offline”	Połączenie SSE przerwane przez proxy — sprawdź, czy hosting nie buforuje <code>text/event-stream</code> .

Załącznik A

prisma/schema.prisma

```
// -----  
// web dev – Prisma schema (PostgreSQL)  
// Money is always stored in minor units (grosze / cents) as Int.  
// -----  
  
generator client {  
  provider = "prisma-client-js"  
}  
  
datasource db {  
  provider = "postgresql"  
  url      = env("DATABASE_URL")  
}  
  
// ----- Enums -----  
  
enum Role {  
  OWNER  
  MEMBER  
  ADMIN  
}  
  
enum Provider {  
  ANTHROPIC  
  GOOGLE_PLACES  
  RESEND  
  TWILIO  
  STRIPE  
  GOCARDLESS  
  VERCEL  
  CLOUDFLARE  
}  
  
enum LeadStatus {  
  NEW  
  SITE_GENERATED  
  CONTACTED  
  REPLIED  
  WON  
  LOST  
}  
  
enum SiteStatus {  
  GENERATING  
  DEMO  
  PUBLISHED  
  ARCHIVED  
  FAILED  
}  
  
enum HostingProvider {  
  INTERNAL  
  VERCEL  
}  
  
enum DomainProvider {  
  VERCEL  
  CLOUDFLARE  
}  
  
enum DomainStatus {  
  PENDING  
  VERIFYING  
  ACTIVE
```

```

    FAILED
  }

  enum Channel {
    EMAIL
    SMS
  }

  enum MessageStatus {
    DRAFT
    SCHEDULED
    SENT
    DELIVERED
    REPLIED
    FAILED
  }

  enum PaymentStatus {
    PENDING
    PAID
    OVERDUE
    REFUNDED
    CANCELED
  }

  enum PaymentSource {
    STRIPE
    BANK_API
    BANK_CSV
    MANUAL
  }

  enum TransactionSource {
    API
    CSV
  }

  enum BankConnectionStatus {
    PENDING
    LINKED
    EXPIRED
    ERROR
  }

  enum CalendarEventType {
    CUSTOM
    FOLLOW_UP
    MEETING
  }

  enum TicketStatus {
    OPEN
    IN_PROGRESS
    RESOLVED
  }

  enum TicketPriority {
    LOW
    NORMAL
    HIGH
  }

  // ----- Auth.js -----

  model User {
    id          String    @id @default(cuid())
    name        String?
    email       String    @unique
    emailVerified DateTime?
    image       String?   @db.Text
    passwordHash String?
    role        Role      @default(OWNER)
    agencyName  String?
    createdAt   DateTime  @default(now())
    updatedAt   DateTime  @updatedAt
  }

```

```

accounts      Account[]
sessions      Session[]
settings      UserSettings?
credentials[] ApiCredential[]
searches      SearchQuery[]
leads         Lead[]
sites         Site[]
domains       Domain[]
messages      OutreachMessage[]
payments      Payment[]
bankConnections BankConnection[]
transactions  BankTransaction[]
events        CalendarEvent[]
tickets       SupportTicket[]
}

model Account {
  id          String @id @default(cuid())
  userId      String
  type        String
  provider     String
  providerAccountId String
  refresh_token String? @db.Text
  access_token String? @db.Text
  expires_at  Int?
  token_type  String?
  scope       String?
  id_token    String? @db.Text
  session_state String?

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([provider, providerAccountId])
}

model Session {
  id          String @id @default(cuid())
  sessionToken String @unique
  userId      String
  expires     DateTime
  user        User @relation(fields: [userId], references: [id], onDelete: Cascade)
}

/// Magic-link tokens (Auth.js Resend provider)
model VerificationToken {
  identifier String
  token      String @unique
  expires    DateTime

  @@unique([identifier, token])
}

/// 6-digit one-time codes for account activation / e-mail change
model OtpCode {
  id          String @id @default(cuid())
  email       String
  purpose     String @default("register") // register | email-change
  codeHash    String
  attempts    Int @default(0)
  expiresAt   DateTime
  createdAt   DateTime @default(now())

  @@index([email, purpose])
}

// ----- Settings -----

model UserSettings {
  id          String @id @default(cuid())
  userId      String @unique
  notifyPayments Boolean @default(true)
  notifyReplies Boolean @default(true)
  notifyWeeklyReport Boolean @default(false)
  senderName  String?

```

```

senderEmail      String?
replyTo          String?
emailSignature   String? @db.Text
defaultEmailSubject String?
defaultEmailTemplate String? @db.Text
defaultSmsTemplate String? @db.Text
defaultPrice     Int       @default(149900) // 1 499,00 PLN
currency         String   @default("PLN")
bankAccountIban  String?
language         String   @default("pl")

user User @relation(fields: [userId], references: [id], onDelete: Cascade)
}

/// Per-user integration keys. `encrypted` holds AES-256-GCM encrypted JSON.
model ApiCredential {
  id      String   @id @default(cuid())
  userId  String
  provider Provider
  encrypted String   @db.Text
  hint    String? // e.g. "sk-ant-...alb2" – never the full key
  verified Boolean   @default(false)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([userId, provider])
}

// ----- Lead generation -----

model SearchQuery {
  id      String   @id @default(cuid())
  userId  String
  category String
  location String
  onlyNoSite Boolean @default(true)
  resultsCount Int   @default(0)
  savedCount  Int   @default(0)
  createdAt  DateTime @default(now())

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
  leads Lead[]
}

model Lead {
  id      String   @id @default(cuid())
  userId  String
  searchId String?
  placeId String
  name    String
  slug    String   @unique
  category String?
  address String?
  city    String?
  phone   String?
  email   String?
  emailMxValid Boolean?
  website String?
  hasWebsite Boolean @default(false)
  rating   Float?
  reviewsCount Int?
  openingHours Json? // string[] e.g. ["poniedziałek: 08:00–16:00", ...]
  latitude Float?
  longitude Float?
  googleMapsUrl String?
  status   LeadStatus @default(NEW)
  notes    String?    @db.Text
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  user      User @relation(fields: [userId], references: [id], onDelete: Cascade)
  search    SearchQuery? @relation(fields: [searchId], references: [id], onDelete: SetNull)
  sites     Site[]

```

```

messages OutreachMessage[]
payments Payment[]
events CalendarEvent[]

@@unique([userId, placeId])
@@index([userId, status])
}

// ----- Generated websites -----

model Site {
  id String @id @default(cuid())
  userId String
  leadId String
  slug String @unique // public demo: /demo/<slug>
  title String
  content Json? // SiteContent (see src/lib/site/types.ts)
  model String?
  status SiteStatus @default(GENERATING)
  error String?
  demoViews Int @default(0)
  price Int // minor units
  currency String @default("PLN")
  hosting HostingProvider @default(INTERNAL)
  vercelProjectId String?
  vercelDeploymentId String?
  productionUrl String?
  publishedAt DateTime?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
  lead Lead @relation(fields: [leadId], references: [id], onDelete: Cascade)
  domains Domain[]
  messages OutreachMessage[]
  payments Payment[]

  @@index([userId, status])
}

model Domain {
  id String @id @default(cuid())
  userId String
  siteId String
  hostname String @unique
  provider DomainProvider @default(VERCEL)
  status DomainStatus @default(PENDING)
  verification Json? // DNS records the client must add
  externalId String? // Cloudflare custom-hostname id
  lastCheckAt DateTime?
  createdAt DateTime @default(now())

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
  site Site @relation(fields: [siteId], references: [id], onDelete: Cascade)
}

// ----- Outreach -----

model OutreachMessage {
  id String @id @default(cuid())
  userId String
  leadId String
  siteId String?
  paymentId String?
  channel Channel
  recipient String
  subject String?
  body String @db.Text
  status MessageStatus @default(DRAFT)
  providerId String? // Resend e-mail id / Twilio SID
  error String?
  scheduledAt DateTime?
  sentAt DateTime?
  createdAt DateTime @default(now())
}

```

```

user      User      @relation(fields: [userId], references: [id], onDelete: Cascade)
lead      Lead      @relation(fields: [leadId], references: [id], onDelete: Cascade)
site      Site?     @relation(fields: [siteId], references: [id], onDelete: SetNull)
payment   Payment?  @relation(fields: [paymentId], references: [id], onDelete: SetNull)

@@index([userId, status, scheduledAt])
}

// ----- Finance -----

model Payment {
  id          String          @id @default(cuid())
  userId      String
  leadId      String?
  siteId      String?
  reference    String          @unique // "WD-7F3K2Q" – put in bank transfer title
  description  String?
  amount      Int // minor units
  currency     String          @default("PLN")
  status      PaymentStatus @default(PENDING)
  source       PaymentSource @default(MANUAL)
  dueDate     DateTime?
  paidAt      DateTime?
  stripePaymentLinkId String?   @unique
  stripePaymentLinkUrl String?
  stripeSessionId String?      @unique
  stripePaymentIntentId String? @unique
  createdAt   DateTime         @default(now())
  updatedAt   DateTime         @updatedAt

  user      User      @relation(fields: [userId], references: [id], onDelete: Cascade)
  lead      Lead?     @relation(fields: [leadId], references: [id], onDelete: SetNull)
  site      Site?     @relation(fields: [siteId], references: [id], onDelete: SetNull)
  transactions BankTransaction[]
  messages   OutreachMessage[]

  @@index([userId, status])
  @@index([userId, paidAt])
}

model BankConnection {
  id          String          @id @default(cuid())
  userId      String
  institutionId String
  institution  String?
  requisitionId String          @unique
  accountIds  String[]
  status      BankConnectionStatus @default(PENDING)
  validUntil  DateTime?
  lastSyncAt  DateTime?
  createdAt   DateTime         @default(now())

  user      User      @relation(fields: [userId], references: [id], onDelete: Cascade)
  transactions BankTransaction[]
}

model BankTransaction {
  id          String          @id @default(cuid())
  userId      String
  connectionId String?
  externalId  String // bank transactionId or hash of CSV row
  source      TransactionSource
  bookingDate DateTime
  amount      Int // minor units, positive = incoming
  currency     String          @default("PLN")
  counterparty String?
  title       String?          @db.Text
  paymentId   String? // matched payment
  raw         Json?
  createdAt   DateTime         @default(now())

  user      User      @relation(fields: [userId], references: [id], onDelete: Cascade)
  connection BankConnection? @relation(fields: [connectionId], references: [id], onDelete: SetNull)
  payment    Payment?   @relation(fields: [paymentId], references: [id], onDelete: SetNull)
}

```

```

    @@unique([userId, externalId])
    @@index([userId, bookingDate])
}

/// Idempotency log for Stripe webhooks
model StripeEvent {
  id      String   @id // evt_...
  type    String
  createdAt DateTime @default(now())
}

// ----- Calendar & support -----

/// Manual events. Payments and scheduled outreach are projected onto the
/// calendar automatically (see src/lib/calendar.ts) – no duplication.
model CalendarEvent {
  id      String           @id @default(cuid())
  userId  String
  leadId  String?
  type    CalendarEventType @default(CUSTOM)
  title   String
  notes   String?
  date    DateTime
  createdAt DateTime       @default(now())

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
  lead Lead? @relation(fields: [leadId], references: [id], onDelete: SetNull)

  @@index([userId, date])
}

model SupportTicket {
  id      String           @id @default(cuid())
  userId  String
  subject  String
  category String
  message  String           @db.Text
  priority TicketPriority @default(NORMAL)
  status   TicketStatus    @default(OPEN)
  createdAt DateTime       @default(now())

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
}

```