Platform Architectuur: gescheiden apps, gedeelde kern

Implementatieplan voor integratie tussen hugoherbots.ai (AI chat/roleplay) en hugoherbots.com (video, webinars, live coaching, admin).

Versie: v1.2
Datum: 20 januari 2026
Scope: Auth & entitlement, navigatie, data/config sharing, RAG corpus, analytics, operational governance.

Executive recommendation
Behoud twee aparte codebases en deployments (hugoherbots.ai en hugoherbots.com), maar ontwerp ze als 1 product-ecosysteem met: (1) 1 gedeelde identity + subscription/entitlement laag, (2) 1 gedeelde data- en configuratiebron (Supabase + SSOT), (3) naadloze cross-app navigatie met context (deep links), (4) expliciete security- en versie-afspraken om schema- en config drift te vermijden.

Waarom dit plan nu?

Je hebt vandaag twee Replits: een AI chat/coach engine en een video/live coaching platform met verschillende pakketten en prijsniveaus. Een volledige code merge is duur en risicovol. Het alternatief is: gescheiden verder bouwen, maar technisch en UX-matig 1 ecosysteem maken.

Wat je krijgt in dit document

- Besliskader: merge vs split (en een derde optie: monorepo zonder merge).
- Doelarchitectuur: gedeelde Supabase kern (data, entitlements, RAG corpus) + twee frontends.
- SSO / login pattern tussen .com en .ai (inclusief risico's en mitigaties).
- Implementatie in fases met Definition of Done en teststrategie per fase.
- Risicoanalyse (security, UX, onderhoud, toekomstbestendigheid).

Inhoud

1. Doel en uitgangspunten

Doel

Een architectuur en implementatiepad definiëren waarmee hugoherbots.ai en hugoherbots.com apart kunnen blijven deployen, maar voor de gebruiker als 1 coherent platform aanvoelen, met 1 account, 1 set rechten (pakketten), en gedeelde data/config.

Niet-doel (bewust)

- Geen volledige code merge of 'big bang' herbouw.
- Geen perfecte 'single-page-app' ervaring over twee domeinen; wel een pragmatische, snelle en veilige hand-off.
- Geen UI pixel-perfect redesign; wel consistente navigatie en branding.

Designprincipes

- Separation of concerns: AI workloads en content/admin workloads schalen anders; hou de deployments onafhankelijk.
- Single source of truth: users, subscriptions, entitlements, configs en RAG corpus leven centraal (niet dupliceren per app).
- Explicit contracts: versieer API's en schema's, zodat updates van app A app B niet stil breken.
- Security by default: least privilege, RLS, korte TTL voor cross-domain tokens, audit logging.
- UX boven interne grenzen: cross-links zijn taakgericht (deep links) en behouden context (session_id, technique_id, etc.).

2. Opties: mergen, gescheiden houden, of monorepo

Er zijn drie realistische strategieën. De juiste keuze hangt af van (a) hoeveel gedeelde code je echt nodig hebt, (b) hoe belangrijk 'login-once' is tussen twee top-level domeinen, en (c) hoeveel operationele overhead je wil dragen.

Optie	Kernidee	Voordelen	Nadelen / risico's	Wanneer kiezen?
A. Volledige merge	1 codebase + 1 deployment	Echte 1-app UX, 1 auth context, 1 repo	Hoge migratiekost; regressierisico; complexiteit explodeert; vertraagt iteratie	Als je team groter is, refactorbudget hebt, en 'alles in 1' cruciaal is
B. Gescheiden apps + gedeelde kern (aanbevolen)	2 codebases + 2 deployments, gedeelde Supabase + gedeelde config/RAG	Snelle iteratie; isolatie van bugs; schaalbaar per workload; productdifferentiatie (basic vs premium)	Cross-domain SSO is niet gratis; governance nodig tegen schema/config drift	Als je snel wil blijven shippen en de apps functioneel duidelijk verschillen
C. Monorepo / shared packages (hybride)	2 deployments, gedeelde packages (UI, SSOT types, client SDK)	Minder duplicatie; consistente UI; betere type safety; toch isolatie in runtime	Setup/CI complexer; discipline nodig rond package versioning	Als duplicatie nu pijn doet maar je niet wil mergen

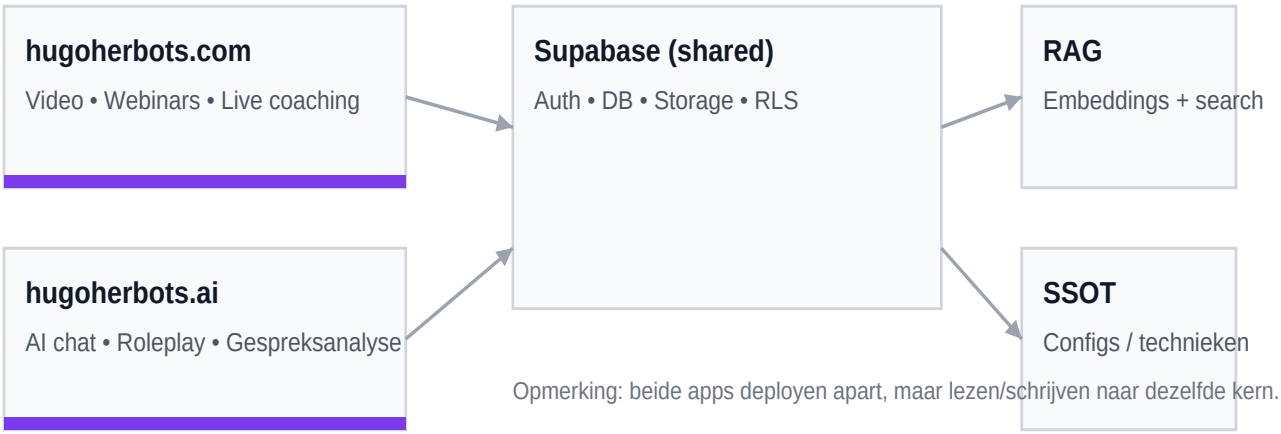
Conclusie optie-keuze

Op basis van je productstrategie (goedkoop/internationaal AI chat versus duur/exclusief live coaching + video) is optie B logisch. Je vermijdt een merge-nachtmerrie, en je behoudt de vrijheid om AI sneller te itereren en anders te schalen.

3. Aanbevolen doelarchitectuur

De kern: laat beide apps autonoom blijven (eigen repo, eigen deployment, eigen runtime), maar maak de kern gedeeld. De kern bevat identity, entitlements, data, configuratie en RAG-documenten. Beide apps zijn clients van die kern.

Target architecture (hoog niveau)



Verdeling van verantwoordelijkheden

Domein	hugoherbots.com	hugoherbots.ai	Gedeelde kern (Supabase + services)
Identity & login	Login UI (optioneel)	Login UI (optioneel)	User identity, auth events, MFA policies
Subscriptions & pakketten	Checkout/upgrade flows	Upsell/upgrade triggers	Entitlements, plan mapping, RLS policies
Content	Video/webinar catalogus, live coaching	Link naar content	Metadata, transcript opslag, storage
AI coaching / roleplay	Link of embed	Primair	Prompt assets, RAG docs, evaluator outputs (waar relevant)
Analytics	Dashboards, admin	Sessiestatistieken	Event schema + centrale event store

4. Identity & entitlement (betalingspakketten)

Omdat je verschillende pakketten hebt (AI-only versus premium met live coaching + video/webinars), moet entitlement de centrale schakel worden. Niet 'welke app', maar 'welke rechten' bepalen toegang.

Minimale entitlements die je expliciet wil modelleren

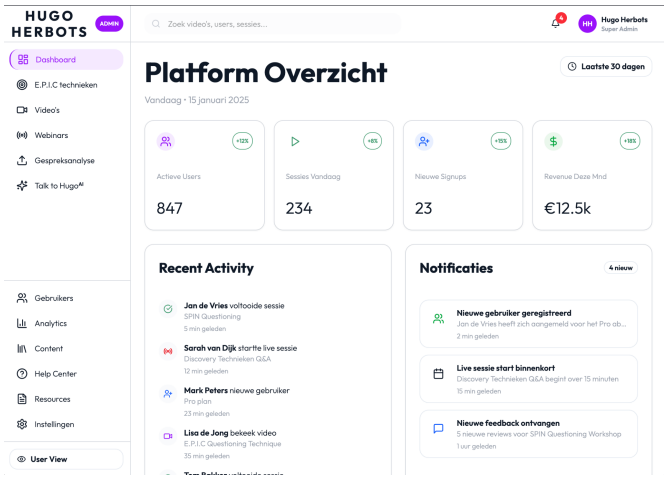
- core.account: basis login + profiel
- ai.chat: coaching chat (meertalig)
- ai.roleplay: roleplay engine / practice flows
- ai.conversation_analysis: gespreksanalyse (uploads, scoring, feedback)
- content.videos: video bibliotheek
- content.webinars: webinars + replays
- coaching.live: live coaching sessies / booking
- admin.platform: admin dashboard, users, analytics, config editor

Technische implementatie (aanbevolen)

- 1 Supabase project voor beide apps; subscription state wordt vanuit Stripe gesynchroniseerd naar Supabase (webhook).
- Entitlements worden afgeleid uit subscription status (plan -> entitlement mapping). Bewaar het resultaat in een tabel (bv. user_entitlements) voor snelle checks.
- Gebruik Postgres functies voor entitlement checks (bv. has_entitlement(user_id, 'ai.roleplay')).
- Enforce op 2 niveaus: (1) UI gating, (2) database/RLS of server API gating (nooit alleen UI).

Belangrijke nuance: cross-domain login is geen 'gratis' feature

Supabase Auth kan perfect door beide domeinen gebruikt worden, maar een browser sessie (tokens) wordt typisch per domein opgeslagen. Dat betekent: zonder extra mechanisme kan een gebruiker bij het wisselen tussen .com en .ai opnieuw moeten inloggen. In sectie 10 (appendix) staat een pragmatisch handoff patroon waarmee je dit bijna onzichtbaar kan maken.



Voorbeeld: hugoherbots.com dashboard met links naar AI en gespreksanalyse (UI moet app-switch consistent maken).

5. Cross-app navigatie en UX: 'bijna onvoelbaar'

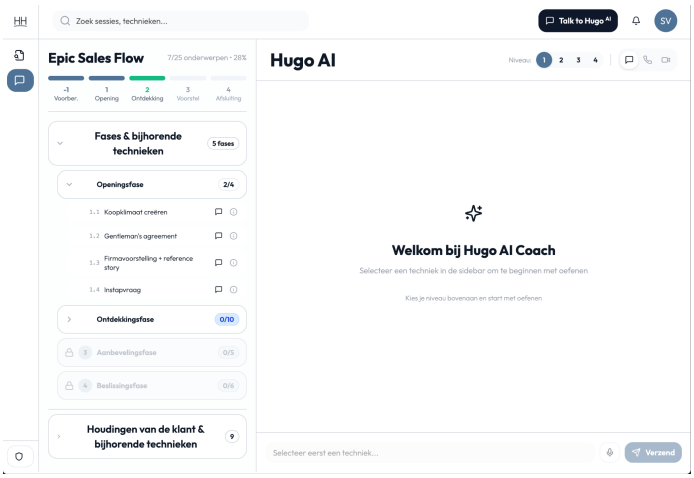
Het is mogelijk om de ervaring bijna onvoelbaar te maken, maar dat vraagt een paar UX-keuzes die consequent doorgevoerd worden.

UX patronen die wél werken

- App switcher: een duidelijke, altijd zichtbare switch (bijv. 'Platform' vs 'AI Coach') in de header.
- Deep links met context: elke navigatie naar de andere app bevat een concrete target (session_id, video_id, technique_id).
- Seamless hand-off: bij klik naar ander domein eerst een korte handoff call om auth/context veilig over te dragen.
- Consistente navigatie: dezelfde sidebar structuur en benamingen in beide apps (ook als items naar andere domein leiden).
- Plan-aware UI: toon items grijs/locked als pakket dit niet bevat (met upgrade CTA).

UX anti-patterns (vermijden)

- Volledige navigatie verbergen in de ene app: dat voelt als 'andere wereld'.
- Links die altijd naar de homepage van de andere app sturen (contextverlies).
- Stille failures: als auth transfer faalt, toon een duidelijke 'Log opnieuw in' en behoud de oorspronkelijke link-target.
- Inconsistent terminology: dezelfde functie met andere naam per app.



Voorbeeld: hugoherbots.ai coaching omgeving. Plaats hier ook links naar video/webinars/live coaching (met deep links).

Praktisch: mapping van navigatie-items

Navigatie-item	Primaire home	Secundaire deep link
Dashboard	.com	.ai kan 'mini dashboard' tonen of doorlinken naar .com/dashboard
Video's	.com	.ai -> .com/videos?filter=...
Webinars	.com	.ai -> .com/webinars/:id
Gespreksanalyse	.ai	.com -> .ai/analysis?session_id=...
Talk to Hugo AI	.ai	.com -> .ai/chat?technique_id=...
Analytics/Admin	.com	.ai -> .com/admin (role gated)

6. Data, SSOT-config en RAG corpus

Je noemt expliciet gedeelde assets zoals technieken_index.json en de RAG corpus, die door beide platformen aangevuld worden (nieuwe video's/webinars enerzijds, config reviews en golden standard chats anderzijds). Dit is precies waar 'split apps' mislukt als er geen single source of truth is.

6.1 SSOT (technieken, overlays, prompts)

Doel: 1 canonieke versie van technieken + overlays + prompt-assets, met versioning en audit trail. Beide apps consumeren dezelfde gepubliceerde versie.

- Optie 1 (snel): maak een aparte 'shared-config' repo of git submodule en importeer in beide Replits. Publiceer versies (tags).
- Optie 2 (aanbevolen): bewaar SSOT in Supabase (jsonb) met 'draft' en 'published' status, inclusief review workflow.
- Cache-strategie: apps fetchen SSOT bij start + on demand; gebruik ETag/version om enkel te verversen bij wijzigingen.
- Compatibiliteit: voeg een 'schema_version' toe zodat oudere clients weten of ze de config kunnen parsen.

6.2 RAG corpus (training context)

Doel: 1 centrale verzameling van documenten met embeddings, metadata en toegangslabels. Beide apps kunnen documenten toevoegen, maar alleen via gecontroleerde ingest (niet rechtstreeks vanuit client).

- Tabel rag_documents: (id, doc_type, title, content, language, source_ref, visibility, embedding, created_at, updated_at).
- Ingest pipeline: transcript -> chunking -> embedding job -> index update.
- Visibility/entitlement: markeer premium-only content; enforce via RLS of via server-side search endpoints.
- Observability: log welke RAG docs gebruikt worden in antwoorden (debug + analytics).

6.3 Statistieken en rapportering

Als zowel AI sessies als video/webinar activiteit events naar dezelfde event store schrijven, kan je analytics in 1 plek bouwen en per product/pakket slicen.

- Gebruik een centrale events tabel (append-only) met event_type, user_id, session_id, payload jsonb, created_at.
- Maak views/materialized views voor dashboards (per dag, per techniek, per kwaliteitsscore, per cohort).
- Cross-domain tracking: client-side analytics is lastiger; server-side events via Supabase/edge functions is stabieler.

7. Implementatieplan in fases

Aanpak: incrementeel en met expliciete 'Definition of Done'. Elke fase levert een bruikbare verbetering op zonder big bang.

Fase	Doel	Belangrijkste deliverables	Risico
0. Alignment & contracten	Beslis en formaliseer shared contracts	Entitlement matrix • Deep link map • Event schema • Security baseline (RLS rules) • Service boundaries	Laag
1. Shared identity + entitlements	1 account en 1 bron van waarheid voor pakketten	Stripe webhook -> subscriptions tabel • user_entitlements tabel • RLS policies • UI gating	Medium
2. Cross-domain hand-off (SSO) + navigatie	Switch tussen apps zonder frictie	Handoff endpoint + one-time code • deep link router • failure UX • audit logging	Hoog
3. SSOT/config centralisatie	Geen config drift tussen apps	SSOT store (Supabase of shared repo) • versioning • publish workflow • cache/ETag	Medium
4. RAG corpus pipeline centralisatie	Beide apps vullen dezelfde kennisbasis	rag_documents schema • ingest jobs • visibility labels • usage logging	Medium-hoog
5. Analytics consolidatie	1 zicht op gebruik en kwaliteit	events store • dashboards/views • cross-product reporting • alerting	Medium
6. (Optioneel) Domein-consolidatie	SSO structureel simpel maken	ai.hugoherbots.com of app.hugoherbots.com • redirects • cookie strategy • HSTS	Medium (strategisch)

Gate per fase

Een fase is pas 'done' als zowel functionele acceptatie als security/observability rond zijn. Dit voorkomt dat de integratie na enkele iteraties alsnog fragiel wordt.

8. Teststrategie

Omdat je twee apps runt die één kern delen, moet je testen vooral focussen op: (a) identity/entitlement, (b) cross-app navigatie, (c) schema/config drift, en (d) security. Onderstaande testset is minimalistisch maar dekt de grootste faalmodi.

8.1 Unit tests (backend)

- Entitlement mapping: plan_id -> entitlements (active, past_due, canceled).
- RLS helper functions: has_entitlement(), is_admin(), visibility rules.
- SSO handoff: one-time code generatie, TTL, replay protection, binding aan user_id + redirect target.
- Deep link router: validatie van session_id/video_id/technique_id; no open redirects.

8.2 Integratietests (tussen apps en Supabase)

- Login op .com -> klik 'Talk to Hugo AI' -> arriveert op .ai ingelogd en op correcte target route.
- Login op .ai -> klik 'Webinars' -> arriveert op .com ingelogd en opent juiste webinar detail.
- Downgrade scenario: premium -> basic; premium features worden op beide apps correct geblokkeerd.
- RAG ingest: nieuwe webinar transcript -> embedding job -> binnen 5 minuten zoekbaar in .ai coaching context.
- SSOT publish: wijzig techniekomschrijving -> beide apps zien nieuwe versie na cache invalidatie.

8.3 End-to-end tests (kritische user journeys)

- AI-only user: kan chatten/roleplay, maar ziet video/webinars als locked met upgrade CTA.
- Premium user: kan na video direct naar gespreksanalyse met dezelfde sessie/context.
- Admin user: kan config aanpassen en effect zien in AI coach zonder handmatige deploy.
- Failure mode: Supabase down of handoff endpoint down -> duidelijke fout, fallback naar login, geen data verlies.

8.4 Regression tests

- Elke app kan nog zelfstandig functioneren als de andere tijdelijk down is.
- Schema migrations zijn backwards compatible (apps kunnen 1 release achter lopen).
- Geen breaking changes in SSOT parsing zonder schema_version bump.

9. Risico's en mitigaties

Gescheiden apps is geen slecht idee, maar je ruilt 'merge-complexiteit' voor 'integratie-contracten'. Onderstaande risico's zijn de belangrijkste om expliciet te managen.

Risico	Waarom dit gebeurt	Mitigatie (concreet)
Cross-domain login frictie	Tokens worden per domein opgeslagen; .com en .ai delen geen cookies/localStorage	Implementeer handoff (one-time code) of migreer naar subdomeinen onder 1 apex domain (fase 6)
Data exposure (premium content)	Beide apps hebben toegang tot dezelfde DB; verkeerde RLS kan premium data lekken	RLS policies per tabel op visibility + entitlement; geen service-role key in client; audit queries
Schema/config drift	2 apps deployen onafhankelijk; schema of SSOT parsing kan verschillen	Versioning (schema_version), compatibele migrations, contract tests, publish workflow met rollback
Coupling door directe app-to-app API calls	.com roept rechtstreeks .ai endpoints aan en creëert runtime dependencies	Centraliseer shared services (Supabase functions/edge) of maak .ai API highly available + versioned
Inconsistente UX/branding	Twee codebases evolueren visueel uit elkaar	Shared design tokens/component library (monorepo package) + visuele regression tests
Complexe debugging	Fouten kunnen in app A, app B of in de gedeelde kern zitten	Structured logging met correlation_id, centrale error tracking, health checks + synthetic handoff tests

10. Appendix: SSO handoff en API-schets

Doel: gebruiker klikt op een link naar het andere domein en blijft ingelogd, zonder dat je cookies tussen .com en .ai moet delen.

10.1 Handoff flow (conceptueel)

1) User is ingelogd op bron-app (heeft Supabase access_token + refresh_token).
2) Bron-app vraagt aan eigen backend: *create_handoff(target_path)*.
3) Backend valideert access_token (user_id) en bewaart tokens **encrypted** met korte TTL in tabel handoff_tokens.
4) Backend geeft one-time code terug (bijv. 256-bit random).
5) Frontend redirect naar doeldomein: https://hugoherbots.ai/handoff#code=...
6) Doel-app leest fragment, post naar eigen backend: *consume_handoff(code)*.
7) Backend haalt tokens op, markeert used_at, verwijdert record, en retourneert tokens.
8) Frontend doet supabase.auth.setSession(tokens) en navigeert naar target_path.

10.2 Minimale tabel (voorbeeld)

```
CREATE TABLE handoff_tokens (
code text PRIMARY KEY,
user_id uuid NOT NULL,
target_path text NOT NULL,
access_token text NOT NULL, -- encrypted at rest
refresh_token text NOT NULL, -- encrypted at rest
expires_at timestamptz NOT NULL,
used_at timestamptz
);
```

10.3 Security checklist

- Gebruik URL fragment (#code=) i.p.v. query param om referer leakage te beperken.
- TTL kort (30-120 sec) + one-time use + delete-on-consume.
- Bind code aan user_id en (optioneel) user agent fingerprint om replay te beperken.
- Valideer target_path strikt (alleen interne routes, geen volledige URL) om open redirects te vermijden.
- Encrypt tokens at rest (bijv. libsodium) en log nooit refresh_token.
- Voer global signout uit bij logout voor consistentie over apps.

10.4 Alternatief (strategisch): subdomeinen onder 1 apex

Als je op termijn de beste UX en de laagste security-complexiteit wil, is migreren naar subdomeinen (bijv. ai.hugoherbots.com en app.hugoherbots.com) de cleanste oplossing. Dan kunnen cookies gedeeld worden op .hugoherbots.com en wordt SSO veel eenvoudiger. Je kan hugoherbots.ai behouden als marketing alias (redirect).

Einde document