

DENAMICO

HubSpot Diamond Solutions Partner · Franchise Practice

HUBSPOT AI AGENT CONTEST · SUBMISSION DOSSIER

Franchise Performance Intelligence

A multi-agent operating layer for franchise networks, built as an installable HubSpot public app.

BUILT, DEPLOYED, AND RUNNING

28 HubSpot app components 4 app objects 9 published agent tools 9 native workflows

Submitted by	Denamico
Product	Franchise Performance Intelligence (FPI)
HubSpot app ID	46836361 · developer platform 2026.03
Live in	Portal 51781063 (fpi-demo), installed since 24 July 2026
Document date	5 August 2026
Status of every figure	Verified against the repository, the AWS account, and HubSpot build #15. Work merged after that build is marked as pending deploy wherever it appears

Confidential. Prepared for contest evaluation.

CONTENTS

0	For the judges	4
0.1	The one-paragraph version	4
0.2	Where each criterion is answered	4
0.3	Three things worth knowing before you read further	5
1	The opportunity	7
1.1	Revenue fog	7
1.2	Who this is for, and what each of them is afraid of	7
1.3	The governing principle: influence, not control	8
1.4	What it is not	8
2	The product	9
2.1	One lap, crossing every surface once	9
2.2	The exchange that makes the data arrive	10
2.3	The play library is the memory and the moat	10
3	Depth of HubSpot tooling	12
3.1	The developer platform and the app	12
3.2	Three betas, three states	14
3.3	Hub-by-hub feature inventory	18
3.4	The data model	22
3.5	The 28 app components	26
3.6	Automation: nine workflows, three custom actions	28
3.7	The member portal: a Content Hub theme built to depth	31
3.8	HubSpot APIs consumed	33
3.9	OAuth scopes: 44 requested	34
3.10	The five classes of build-time-baked URL	35
4	The agent layer	36
4.1	Nine agent tools	36
4.2	The Breeze agent roster, and a finding that reshaped it	37
4.3	Four authorization planes	41
4.4	What binds every tool, enforced in the payload rather than the prompt	41
4.5	The refusal contract	42
4.6	The second door: native webhooks	43
4.7	Tools yes, agents no: a platform finding with a cost attached	44
4.8	What turned out not to be an agent	46
5	Architecture and innovation	48
5.1	Three runtimes, one product	48
5.2	Why the whole backend is in AWS, and why that was not a preference	48
5.3	Built once, installed many times	49
5.4	The AWS backend	50

6	Governance, security, and compliance	53
6.1	Four gates	53
6.2	The two approval gates never cross	53
6.3	The isolation guard: one module where a bug is a data breach	55
6.4	The member session chain	56
6.5	Five caller classes, five proofs	56
6.6	The benchmark anonymity floor	58
6.7	Franchisee-facing responses omit rather than null	58
6.8	Consent compliance	58
6.9	Secrets discipline	59
7	The design system and the brand posture	60
7.1	The brand is a chassis, not a badge	60
7.2	The three-layer colour model, made literal in code	60
7.3	Number presentation is a brand commitment	61
7.4	Thirty-one components, five of which are non-negotiable	61
8	Evidence: what is built, and what is verified	63
8.1	Deployed and verified	63
8.2	Exercised against a live portal	63
8.3	Quality gates	64
8.4	Not yet exercised	65
8.5	Known-open defects and gaps, stated plainly	65
9	Field report: where the platform and its documentation disagree	67
9.1	App and platform level	67
9.2	Associations and CRM writes	68
9.3	HubDB, CMS, and pages	68
9.4	HubL and Jinjava: the most expensive category	69
9.5	Automation	70
9.6	What we would ask HubSpot for	71
10	Scale, impact, and the road to September	72
10.1	Why this scales, structurally	72
10.2	Customer impact	73
10.3	The road to September, and past it	73
10.4	Decisions that belong to Denamico, stated openly	74
Appendix A	Numbers at a glance	75
Appendix B	Glossary	77

0 For the judges

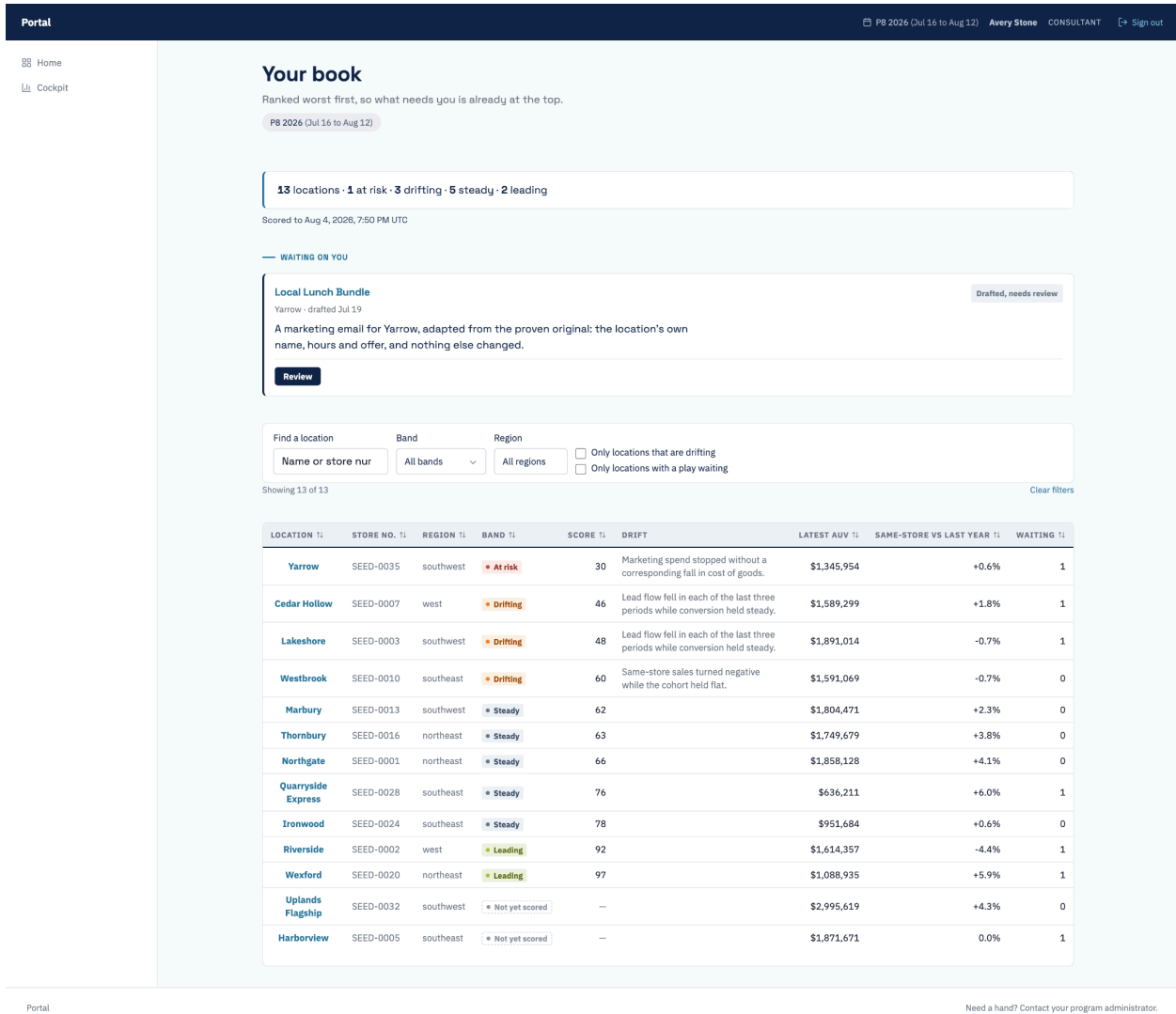
This document exists to let you evaluate Franchise Performance Intelligence without having to reconstruct anything. It is organised so that the five judging criteria map onto named sections, every claim carries the evidence that supports it, and the things that are not finished are stated as plainly as the things that are.

0.1 The one-paragraph version

WHAT FPI IS

A franchisee hands over their monthly profit and loss statement and gets something genuinely useful back **in the same interaction**. That exchange is the whole product. A deterministic scoring pass grades every location each period and flags the ones deteriorating against their own trend and their cohort. HubSpot Breeze agents then draft a specific intervention grounded in a library of interventions that have actually worked elsewhere, and route it through two independent human approval gates. The approved asset deploys through a native HubSpot workflow, and a writeback measures whether it worked. The network gets smarter every lap.

It is built as an installable HubSpot public app carrying its own data model, so the second franchise brand gets it the moment they install. It is live in a HubSpot portal today.



Criterion	Where	The strongest single piece of evidence
Innovation	Part 4, Part 5	Four distinct authorization planes resolving the same question differently; a refusal contract engineered so that a large language model handed a denial cannot turn it into a membership oracle; a learning loop where confidence is a record of history rather than a score; runtime schema resolution so nothing hardcodes a wire name.
Scale	Part 1, Part 10	Roughly 830,000 franchise establishments in the Locationed States alone. The architecture is explicitly built once and installed many times: no per-portal provisioning, no hardcoded object type ids, per-portal OAuth token custody, configuration in HubDB rather than code.
Customer impact	Part 1, Part 10	The consultant persona carries a book of 30 to 150 locations and currently finds out about a problem when the lagging metric arrives. FPI moves that discovery forward by a full reporting period. The franchisee gets a benchmark instead of a compliance demand.
Story and stage	Part 8, Part 9	A working system in a live portal with 24 of 24 isolation tests and 18 of 18 governance-gate tests passing, plus a field report of more than forty documented platform behaviours where the published documentation and the platform disagree. Part 9 is written as feedback to HubSpot, not as a complaint.

0.3 Three things worth knowing before you read further

This is not a prototype

The app is installed in a live HubSpot portal and has been since 24 July 2026. A real OAuth round trip has completed. Real records have been written across all four objects. The rate limiter has survived a burst of roughly 530 writes including a 429 back-off. Two Breeze agents have run end to end against live tool output. Every number in this document was read from the repository, the AWS account, or the HubSpot build API on the document date rather than copied from an older document.

The intelligence is the engine, not the pitch

FPI is not sold as an AI product. It is sold as a way for a franchise consultant to know which of their locations needs them before that location calls. The agents are how it works, not what it is. This distinction shapes the entire design, most visibly in the rule that no agent is ever given a name or a personality on a franchisee-facing surface.

The open items are listed

Part 8.5 lists ten known-open defects and gaps by name, with severity. One of them is a fail-open path in the module where a bug would be a data breach rather than a defect. It is disclosed because a judge evaluating an operations product should be able to see whether the team knows where its own weak points are. A system this size with nothing open would be a system nobody had looked at closely.

1 The opportunity

1.1 Revenue fog

A franchisor sells a system. A franchisee buys the right to run it. What sits between them is a relationship with almost no direct authority in it, because the franchisee is an independent business owner who has put their own capital at risk. The franchisor cannot instruct. It can only influence.

The instrument of that influence is the franchise business consultant, who carries a book of somewhere between 30 and 150 locations. Their job is to notice which locations are struggling and help them. In practice, the noticing happens through lagging indicators that arrive late: royalty reports, quarterly reviews, a location that goes quiet. By the time a consultant knows a location is in trouble, the trouble has been running for a reporting period or more.

Underneath that is a data problem. The franchisor sees royalty revenue because royalty revenue is what gets reported. It does not reliably see cost of goods, labour percentage, occupancy, or marketing spend, because collecting those from an independent owner is an extraction problem: corporate demands compliance data, the owner complies late and incompletely, and nobody gets a benchmark out of it. The result is a network where the corporate view of location economics is fog.

And underneath that is a knowledge problem, which is the interesting one. In any network of a hundred locations, several have already solved the problem a struggling location is facing. That solution is sitting in one operator's head, or in a consultant's notes, or in a marketing asset that ran once in one market and worked. There is no mechanism that moves it. **Every franchise network is quietly re-solving problems it has already solved.**

1.2 Who this is for, and what each of them is afraid of

Three people touch this system, and the design of every surface follows from what each of them is afraid of. Getting the fear right matters more than getting the feature right, because a franchisee who feels graded stops submitting data and the whole loop starves.

	The fear	The promise
Franchisee	Being graded. Being compared to a named peer. Handing over financial data and getting nothing back for it.	Your numbers stay yours. What you get back is about your business, not your ranking. No peer is ever named, and no other location's figures ever reach your screen.
Consultant	Drowning in a book of 30 to 150 locations and missing the one that needed them this week.	You will know who needs you before they call, and you will get a drafted intervention with its track record attached rather than a blank page.

	The fear	The promise
Executive	Not knowing whether any of this is working, and not being able to prove the programme caused anything.	You can see whether the network is moving, and the measurement writeback tells you whether the programme moved it.

1.3 The governing principle: influence, not control

The obvious version of this product autonomously optimises franchisees. That version is wrong, and it is wrong in a way that would destroy the product commercially rather than merely making it slightly worse.

Franchisees are independent owners. A system that acts on their business without their consent is not a growth tool, it is a liability. So the design constraint is absolute: **the system recommends and equips; a human approves.** Cross-network plays are opt-in. The autonomy setting exists in the codebase and in September it always resolves to gated, because shipping the branch and never enabling it is honest, and shipping a dial that does something is not what this stage of the product should do.

WHY THIS MAKES THE PRODUCT BETTER RATHER THAN WEAKER

The hero of this system is the consultant, not the algorithm. A consultant covering 130 locations does not want a machine that acts for them. They want a machine that reads all 130 P&Ls every period, remembers every intervention the network has ever tried, and hands them a short list with a drafted starting point.

That is a materially easier product to sell into a franchise organisation, and it is also a materially better product, because the consultant's judgement about their own book is the most valuable signal in the system and an autonomous design throws it away.

1.4 What it is not

Stating the negative space precisely is part of the design, because each of these is a plausible adjacent product that FPI deliberately is not.

Not this	Because
A compliance portal	Compliance portals extract. A franchisee fills them in because they have to, which produces late and low-quality data. FPI inverts the exchange: you submit a P&L because of what comes back.
A leaderboard	Ranking independent operators against each other by name is exactly the fear that stops submission. The benchmark is a mirror against an anonymised cohort, never a league table.
A dashboard product	A dashboard tells you a number is bad. FPI tells you what to do about it, hands you a draft, and then checks whether it worked.
An AI product	The agents are the mechanism. Nothing on a franchisee-facing surface names an agent, gives it a personality, or writes in the first person. The word for this in the design system is attribution without anthropomorphism.

2 The product

2.1 One lap, crossing every surface once

The product is a single loop. It touches the CRM, the Content Hub portal, the Breeze agent layer, the automation engine, and the marketing tools, once each per lap, and then it starts again with better information than it had last time.

Stage	What happens	Where it runs
Sense	A point-of-sale feed and the portal's P&L form write Location and Financial Period records. Two idempotency keys make both paths safe to re-run.	AWS <code>feedIngest</code> , <code>pn1Submit</code> ; HubSpot CRM app objects
Recommend	A deterministic scoring action grades every location against its own trend and its cohort, and flags deterioration. A Prescribe agent then drafts a Play Assignment grounded in the knowledge vault, with citations.	HubSpot workflow W7 → AWS <code>scoreLocation</code> ; then Breeze agents calling FPI agent tools
Govern	A consultant approves in the agent inbox. Where the play touches customers, the franchisee approves in the portal. Two distinct gates, one record, and they never cross.	AWS <code>playDecision</code> ; HubSpot CRM cards; the member portal
Deploy	The Approved stage fires a native HubSpot workflow whose custom action clones and adapts the proven asset into the content approval gate.	HubSpot workflow W4 → AWS <code>workflowDeploy</code> → <code>workerAdapt</code>
Measure	The writeback records the delta against baseline and rolls confidence up to the parent Play, which is what makes the next lap sharper.	AWS <code>measure</code> ; HubSpot CRM

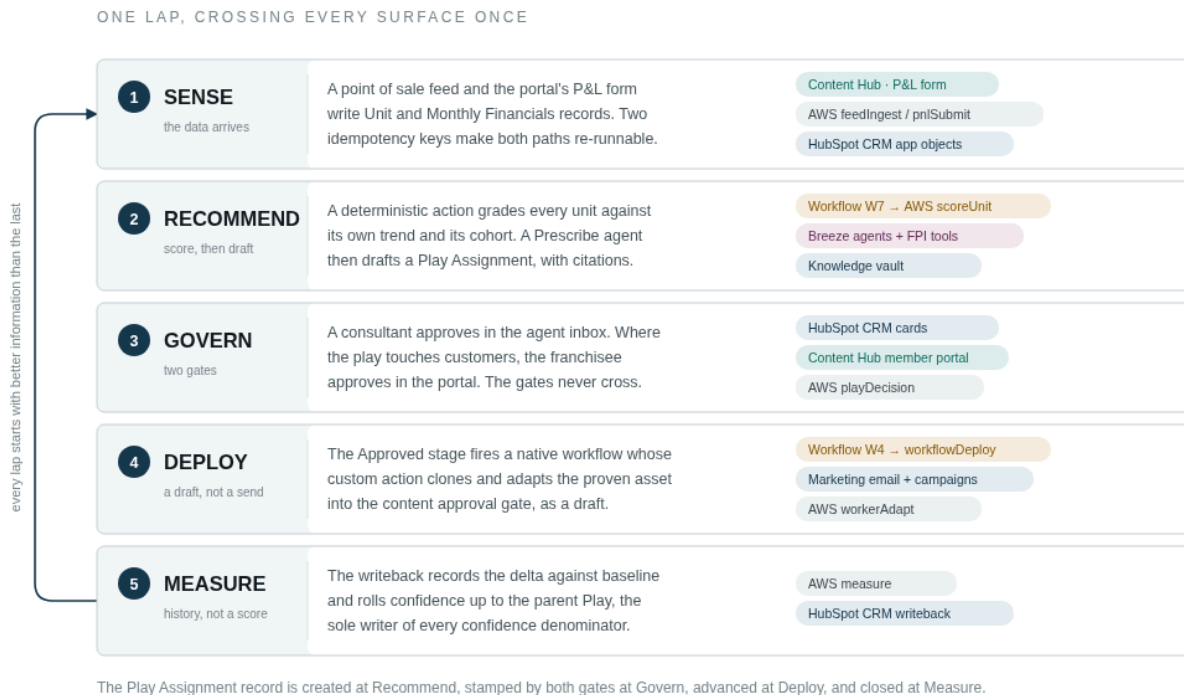


Figure 2. One lap of the loop. Sense and Measure write to the CRM, Recommend runs in Breeze, Govern spans the agent inbox and the member portal, and Deploy fires a native workflow.

THE SPINE

The Play Assignment record is the spine of the entire lap. It is created by the Recommend stage, stamped by both governance gates, advanced by the deploy workflow, and closed by the measurement writeback. Its five-stage lifecycle is what buys stage timestamps, board views, stage-triggered workflows, and funnel reporting from HubSpot without writing any of them.

2.2 The exchange that makes the data arrive

The hardest input in this entire system is franchisee-level profit and loss data. It is also the most valuable, because it is the only thing that turns a revenue number into a margin story. Every attempt to collect it as a compliance obligation produces late, partial, unreliable data.

So FPI does not collect it as an obligation. The franchisee submits their P&L through the member portal, and what comes back immediately is a benchmark of their own numbers against an anonymised cohort of comparable locations, plus a short list of interventions that have moved that specific metric for locations like theirs. The submission is the price of admission to something the franchisee actually wants.

This creates a second loop

The outcomes loop makes the play library smarter. The submission loop makes the benchmarks sharper. They compound against each other: more submissions produce denser cohorts, denser cohorts produce

better benchmarks, better benchmarks make submission more worthwhile, and every play that gets tried and measured makes the recommendation on the next submission more specific.

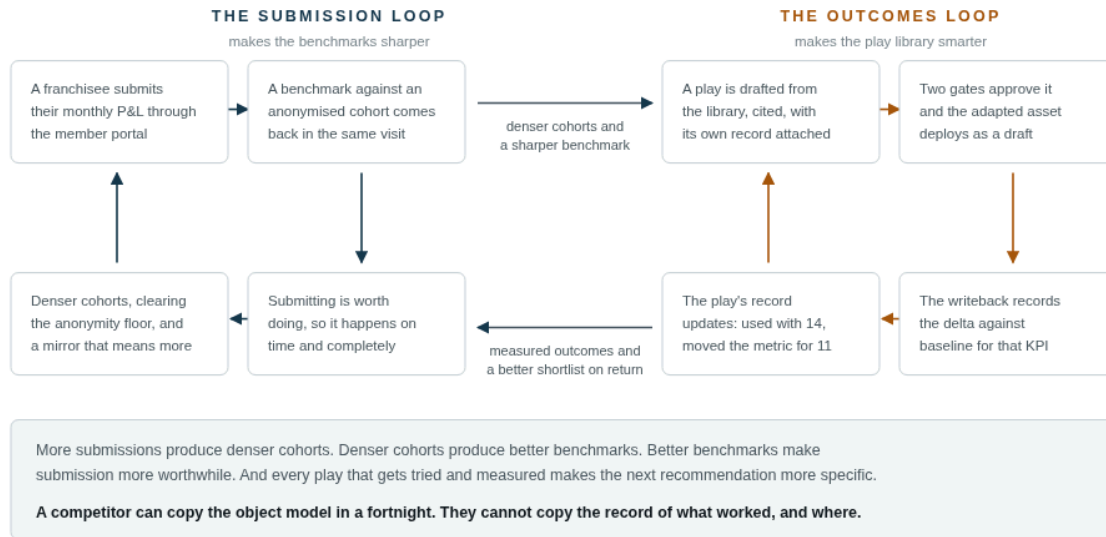


Figure 3. The two loops that compound. The submission loop makes the benchmarks sharper; the outcomes loop makes the play library smarter. Each one raises the value of the other.

This is the part of the design that is genuinely defensible. A competitor can copy the object model in a fortnight. They cannot copy an accumulated record of what was tried, where, for which cohort, and whether it worked, because that record only exists after the network has run laps.

2.3 The play library is the memory and the moat

A Play is a reusable intervention: a localised campaign, an incentive structure, an operational change, a coaching brief. A Play Assignment is one attempt to apply one Play to one location, carrying its own outcome. The library accumulates.

Two design decisions about this library matter more than anything else in the product, and both of them are about honesty:

- 1. Confidence is history, never a score.** The tools return `times_used` and `times_it_worked`. A `confidence_score` property does exist on the Play object and it is deliberately absent from every agent tool response, because the moment an agent can see a score it will quote it, and a quoted score invites a false precision the underlying sample does not support. What a consultant reads is "used with 14 locations, moved the metric for 11."
- 2. A percentile never appears without its cohort size.** A location told it sits in the 40th percentile has been told nothing until it also knows the comparison was against five locations rather than fifty. The two figures are returned in the same object so they cannot be separated by accident, and the anonymity floor guarantees the cohort is never small enough for the percentile to identify a peer.

A PLAY AN AGENT WROTE CANNOT BE RECOMMENDED BY AN AGENT

The Learn stage lets an agent draft a new Play candidate from a location that outperformed. That creates an obvious hazard: an agent drafts a play at location A, another agent picks it up and recommends it to a franchisee at location B, and no human has read a word of it.

The guard is structural rather than procedural. `draft-play-candidate` writes `play_status: candidate`. The loader that feeds the recommendation path filters to `active`. A human promotion is the only path between the two states.

3 Depth of HubSpot tooling

This is the section the contest asks about most directly, so it is written to be exhaustive rather than representative. Every HubSpot capability, hub, beta, API, and platform feature the build touches is listed, with what FPI does with it and what it cost to learn.

THE SHORT VERSION

FPI is a **marketplace-distributed OAuth public app on developer platform 2026.03**, carrying 28 components including four app objects with 84 properties and eight labelled associations. It engages **three separate HubSpot betas at three different states**. It uses the CRM as its object model, Content Hub for the member experience, Marketing Hub for asset deployment, the automation engine for orchestration, and Breeze for the intelligence layer. It consumes roughly sixty distinct HubSpot API endpoints across six product areas and requests 44 OAuth scopes.

It deliberately does **not** use Data Hub custom coded actions, because they require a tier every installing franchisor would then have to buy.

3.1 The developer platform and the app

Item	Value
Developer platform version	2026.03
Project name / source directory	denamico-fpi / src
App ID	46836361
App name	Franchise Performance Intelligence
Distribution	marketplace, unlisted (install links; up to 25 accounts before a listing is required)
Auth type	OAuth 2.0
Developer test account	fpi-demo, portal 51781063 (DEVELOPER_TEST)
Latest build	#15, SUCCESS, 4 August 2026, 28 components green
Tooling	@hubspot/cli 8.9.1+ as a local devDependency, driven through the HubSpot Developer MCP server

Distribution was a forced choice, and it is worth understanding why

The technical specification originally chose `distribution: private`, specifically to stay out of the marketplace. That turned out to be impossible. HubSpot's own feature matrix marks app objects as unavailable for both private distribution modes and available only for OAuth marketplace apps. The data model the product needs therefore forced marketplace distribution.

Marketplace distribution does not force a public listing, because 25 installs are permitted before listing is required. What it does add is HubSpot approval, a HubSpot-assigned object naming scheme, and an acceptable use policy signature. This is a real constraint on a partner building a vertical app, and it is the first of several places where the platform's capability matrix, rather than a design preference, determined the architecture.

3.2 Three betas, three states

FPI engages three distinct HubSpot beta or entitlement gates. One is shipped and in production use, one is authored in full and deliberately held out of the build, and one opened during the build and forced a permanent decision within hours. Each is described with what it bought, what it cost, and what was learned about the platform.

BETA 1 · APP OBJECTS

APPROVED, SHIPPED, IN PRODUCTION USE

The FPI data model deploys **as code inside the app** rather than being provisioned per portal. Four app objects, 84 properties, and eight labelled associations are declared in `*-hsmeta.json` files and created by HubSpot on `hs project upload`.

What it buys. The second franchise client gets the entire data model the moment they install. No per-portal object creation, no custom-object provisioning script, no drift between installs. For a product whose governing rule is "built once, installed many times," this single platform capability removes what would otherwise be the largest source of per-install work and per-install divergence.

FIGURE 4

Screenshot: the four FPI app objects in the portal CRM navigation, with record counts

place image here

Figure 4. The data model as an installing customer sees it. These four objects shipped inside the app and were present the moment it was installed, with no provisioning step.

Current status. Live. All four objects resolve against portal 51781063 with real object type ids. Reads and writes have run across all four. The rate limiter survived a burst of roughly 530 writes including a 429 back-off.

PLATFORM BEHAVIOURS DISCOVERED WHILE SHIPPING ON THIS BETA

- Properties are auto-prefixed `a46836361_` at runtime, which contradicts the specification and is handled by runtime prefix resolution rather than by hardcoding.
- Scope strings use **lowercase** object names while the fully-qualified name is upper snake case.
- `GET /crm/v3/schemas`, the list route, requires a scope this app does not hold, so objects are resolved one at a time by fully-qualified name.

- User-reference properties cannot be declared in hsmeta at all on 2026.03: the app-object config schema is `additionalProperties: false` over a field list that contains neither `externalOptions` nor `referencedObjectType`. The one property that needs it is created by the setup runner through the CRM properties API instead.
- App objects are **absent from the CMS GraphQL association schema** while present as top-level collections, which retired five queries. This is documented in full in Part 9.
- App objects **can** be enrolled as a workflow's object type. This was previously undocumented and was confirmed by successfully creating workflow W5 against `a46836361_UNIT`.

BETA 2 · APP EVENTS

OPT-IN GRANTED 4 AUGUST 2026. EIGHT EVENTS PUBLISHED.

THE EIGHT EVENT TYPES, AS AUTHORED

uid	Object	Properties carried
fpi-location-scored	Location	trajectory_score, trajectory_band, drift_flag, drift_reason, scored_at
fpi-financials-submitted	Location	period_label, period_key, submission_source, amended, pos_variance_flag
fpi-benchmark-returned	Location	period_label, cohort_size, revenue_percentile, net_margin_percentile, below_floor
fpi-play-recommended	Play Assignment	play_name, kpi_name, assignments_count, assignments_worked, citation_count, drafted_by
fpi-play-decided	Play Assignment	decision, gate, decided_by, decided_by_email, decided_at, channel, rejected_reason
fpi-play-deployed	Play Assignment	asset_type, deployed_asset_id, source_asset_id, campaign_id, deployed_at
fpi-play-measured	Play Assignment	kpi_name, outcome, baseline_value, result_value, kpi_delta, measured_at
fpi-assignment-stage-changed	Play Assignment	stage, change_source, source_id, changed_at

Two design decisions inside the event work

Every event sets `propertyWriteMode: NEVER_WRITE` and `onMiss: DISCARD`. The documented property-stamping feature lets an occurrence overwrite a CRM property, which is a path straight through the guard that protects agent-owned properties: an event could raise a trajectory score nobody scored. Closing that by construction rather than by discipline costs nothing. `DISCARD` rather than `CREATE` stops an unresolvable occurrence conjuring a phantom Location record.

Occurrence ids are global, not per-portal. HubSpot requires the occurrence id to be unique *across all accounts*, and a duplicate inside a year is silently discarded, keeping the first. So an id like `assignment-4021-decided` would collide between two installed portals, and the second brand's audit trail would quietly stop appearing. The events module composes every id from the portal id outwards and exports no way to build one without a portal. This is the governing rule's defect class exactly: it works for one portal and fails for the second.

BETA 3 · AGENT TOOLS (BREEZE)

OPT-IN GRANTED 4 AUGUST 2026. NINE TOOLS UPLOADED AND PUBLISHED.

The developer test portal was opted into the agent-tools beta on the document date, closing a tracking item that had been open for weeks. Five tools uploaded in build #13, four more joined in #14, and all nine were published in #15.

How the gate was confirmed without spending anything. Before the opt-in landed, `npx hs project add` printed a local refusal for the agent-tool component type. The CLI answers the entitlement question locally, which is how the gate was confirmed without uploading speculative components and spending permanent uids to find out.

One permanent decision was forced by a builder rule. The builder demands "Input field definition must have exactly one supportedValueType," while the config schema types `supportedValueTypes` as an array with no maximum. `STATIC_VALUE` was chosen, permanently, because an agent constructs its own inputs. The cost is written down because it cannot be undone: **these nine tools can never be driven from a workflow enrolled on Location records.**

A second, independent route was built while the beta was shut, and is kept. HubSpot's stock "Send a webhook" action in the agent builder needs no beta. Nine `/v1/native/*` routes reach the same handlers through a different identity source. It is retained not as dead weight but because it solves a different problem, and it is the fallback if beta access lapses before the September deadline. It is described in full in Part 4.6.

3.3 Hub-by-hub feature inventory

What follows is every HubSpot feature the build touches, grouped by hub, with what FPI does with each.

CRM · the object model

Feature	How FPI uses it
App objects	Four objects shipped inside the app: Location, Financial Period, Play, Play Assignment
App-object associations	Eight labelled associations, including two to standard objects (Contact and Company). All <code>INTEGRATOR_DEFINED</code>
Calculated properties	<code>labor_pct</code> and <code>net_margin_pct</code> on Financial Period, using the <code>calculation_equation</code> field type, platform-computed and confirmed live
Unique-value properties	<code>external_location_id</code> on Location and <code>period_key</code> on Financial Period. These are the two idempotency keys the entire write path rests on
Enumerations	13 enumeration properties across the four objects, including the five-stage lifecycle
CRM record cards (UI extensions)	Two React and TypeScript cards at <code>crm.record.tab</code> , on Location and on Play Assignment
Standard-object properties	Four created by the setup runner: Contact <code>fpi_role</code> and <code>fpi_registered_at</code> ; Company <code>fpi_is_franchisee</code> and <code>fpi_locations_owned</code>
CRM Lists	Three access lists (FPI Consultants, FPI Franchisees, FPI Executives), created via <code>POST /crm/v3/lists</code> , dynamic on <code>fpi_role</code>
Owners	The <code>hubspot_owner_id</code> cascade for owned-only visibility, stamped by workflows W1a, W1b, and W2
Batch APIs	<code>batch/read</code> , <code>batch/upsert</code> , and <code>batch/update</code> on all four objects; <code>associations/batch/read</code> for every isolation walk
Association labels API	<code>GET /crm/v4/associations/{from}/{to}/labels</code> , resolved per portal and cached
CRM search	Used for bounded reads. Explicitly not used for the isolation walk, because HubSpot does not support filtering on associations in the search API

Content Hub · the member portal

Feature	How FPI uses it
Themes	<code>franwise</code> , version 1.8.0, a flat self-contained fork with no <code>extends</code> key
Templates	8 portal page templates, 1 portal layout, 11 membership and system templates, plus partials
Custom modules	30 modules, 21 of which carry their own JavaScript

Feature	How FPI uses it
HubL	56 macros across 6 files covering all 31 design-system components; <code>crm_objects()</code> , <code>crm_associations()</code> , <code>hubdb_table_rows()</code> , <code>request_contact</code> , <code>is_in_page_editor</code>
Memberships / private content	The whole franchisee experience. Contacts with membership passwords. Franchisees hold no HubSpot seats , which is what makes opt-in possible at network scale
<code>publicAccessRules</code>	Page gating by access-group membership. A write-only field: a read after write returns an empty array, so verification fails closed rather than reporting green
CMS serverless functions	Three endpoints: the member-auth broker, an allowlisted member-route proxy, and an identity echo
HubDB	<code>fpi_config</code> , one table per portal and one row per brand. Chosen over an external store precisely because it is natively readable by CMS templates at render time
Source Code API	<code>/cms/source-code/2026-03</code> . The setup runner installs the theme itself; also used as the pre-upload validator
Pages API	<code>/cms/pages/2026-03/site-pages</code> . The runner creates and publishes the eight portal pages
Domains API	GET <code>/cms/v3/domains</code> for domain detection during setup
CMS GraphQL	Two query files remain. Five association-traversing queries were retired: see Part 9
<code>`fields.json`</code> theming	Identity layer only: logo, three colours, typeface. Nothing semantic, by design
Site content search	Deliberately avoided, because dynamic pages would index record data

Marketing Hub · deployment of the approved asset

Feature	How FPI uses it
Marketing email	POST <code>/marketing/v3/emails/clone</code> performs the clone-and-adapt of the proven asset
Campaigns	<code>/marketing/v3/campaigns</code> for asset association on deploy, so attribution survives
Subscription types	"Franchise Performance Updates", looked up by name. No public create API exists , so the setup runner halts with an instruction and must never substitute another type, because substituting would mail under a consent that was never given
Content approvals	Cloned assets are created as drafts and never published . This is enforceable rather than merely intended, because push-live cannot publish a never-published draft

Automation · orchestration

Feature	How FPI uses it
Native workflows (flows v4)	Nine committed definitions, imported by the setup runner via POST /automation/v4/flows. One is schedule-enrolled
Custom workflow actions	Three: <code>fpi-deploy-approved-play</code> , <code>fpi-stamp-access</code> , and <code>fpi-score-location</code>
The Run agent action	HubSpot's built-in action for invoking a published Breeze agent from a flow. This is what makes the executive briefing a scheduled job rather than something a person must remember
Schedule enrolment	<code>enrollmentSchedule</code> on a list-criteria flow. Documented only against a beta namespace, but present in the stable v4 flow representation
Action callbacks	POST /automation/v4/actions/callbacks/{id}/complete. The deploy action always answers BLOCK and completes asynchronously
Action-type catalogue	GET /automation/2026-09-beta/action-types is documented but not deployed on this portal. Handled by a fallback rather than by failing closed: see Part 9.5
Custom code actions	Deliberately not used. They require Data Hub Professional, which every installing franchisor would then have to buy

Breeze and Agent Hub · the intelligence layer

Feature	How FPI uses it
Agent tools	Nine <code>workflow-action</code> components with <code>supportedClients: [{client: "AGENTS"}]</code> . Seven GET_DATA, two TAKE_ACTION
llmConfig.actionDescription	Where every brand rule and compliance prohibition is duplicated per tool, so a tool bound to a <i>*different*</i> agent still carries them
executionRules	The refusal channel: an array of conditions matching on <code>errorCode</code> , mapped to member-visible copy per locale
hs_execution_state: FAIL_CONTINUE	Refusals return HTTP 200 carrying this state, so a refusal does not dead-end the agent's turn
Native "Send a webhook" action	The second door: nine <code>/v1/native/*</code> routes, no beta required
Agent Hub / agent builder	Agents themselves are not programmable . Tools yes, agents no. See Part 4.7 for the evidence
Run agent action	HubSpot's built-in flow action for invoking a published agent. This is what turns the executive briefing from something a person must remember into a scheduled job
Developer Tool Testing Agent	The only test path available here. HubSpot documents two-phase tool testing that begins in workflows, but that assumes a tool supported in both clients; these are agent-only, so phase one does not exist for this app

Feature	How FPI uses it
Knowledge vaults	The grounding layer for the Prescribe agent, fed by a segment of Play records plus narrative playbooks, with responses cited to document and section

Data Hub · what is deliberately not used

The absence is a design decision and worth an explicit note, because a judge could reasonably wonder why a data-heavy product avoids the data tooling.

- **Custom coded workflow actions are deliberately avoided.** The owner cascade in workflows W1a, W1b, and W2 would be natural in a code action. Instead a second app-defined action ships, so that no installing franchisor is forced to buy a tier to make the product work.
- **Datasets and data-quality tooling are not used.**
- The data-layer work that would sit in Data Hub is carried by **HubDB** for configuration that templates must read at render time, **DynamoDB** for token custody, idempotency and job state, and **the CRM object model itself** for everything a human or a report ever needs to see.

3.4 The data model

Object	uid	Object type ID (fpi-demo)	Properties
Location	fpi-location-object	1-13082241	16
Financial Period	fpi-financial-period-object	1-13082239	25
Play	fpi-play-object	1-13082238	13
Play Assignment	fpi-play-assignment-object	1-13082240	30
Total			84

Fully-qualified names are a46836361_UNIT, a46836361_MONTHLY_FINANCIALS, a46836361_PLAY, and a46836361_PLAY_ASSIGNMENT. The runtime property prefix is a46836361_. Property names in the repository are written un-prefixed and resolved at runtime, which is what makes the granted object naming a cache invalidation rather than a code change.

Location · a franchise location tracked for performance and coaching

Property	Type	Notes
location_name	string	
external_location_id	string	hasUniqueValue: true. The point-of-sale feed's idempotency key
region_market	enumeration	Placeholder option set pending the brand
cohort_tags	enumeration (checkbox)	Size, maturity, and format tags. The benchmark cohort key
location_status	enumeration	building, open, closed
open_date	date	
latest_auv	number	Average location volume
same_store_sales_pct	number	
lead_flow_30d	number	Demand signal. A scalar the feed overwrites, carrying no history , which is why drift cannot be defined against it. See below
lead_conversion_pct_30d	number	Conversion signal, with the same caveat
trajectory_score	number	agent-owned
trajectory_band	enumeration	agent-owned. leading, steady, drifting, at_risk
drift_flag	bool	agent-owned

Property	Type	Notes
drift_reason	string	agent-owned
drift_since	datetime	agent-owned. Answers "what is new since Friday"
last_scored_at	datetime	agent-owned. Every score carries its date

Financial Period · one row per location per period

Seven properties carry the P&L line items: revenue, cost of goods, labour cost, occupancy cost, marketing spend, other expenses, and net profit. Two more are platform-calculated. The rest carry the benchmark snapshot and the submission provenance.

Property	Type	Notes
period_key	string	hasUniqueValue: true , shaped "STORE-19::P7 2026". HubSpot has no composite unique constraint, so "one row per location per period" is materialised into a single property
period_start, period_end	date	A period is never a bare month
the seven line items	number	revenue, cogs, labor_cost, occupancy_cost, marketing_spend, other_expenses, net_profit
labor_pct	calculation_equation	Platform-computed. Confirmed live
net_margin_pct	calculation_equation	Platform-computed. Confirmed live
source	enumeration	form, agent_assisted, imported
pos_reported_revenue	number	Point-of-sale feed only
pos_variance_pct	number	Plain number, not calculated. The reason is a platform defect documented in Part 9.1
financials_status	enumeration	submitted, validated, flagged
benchmark snapshot	number / datetime	benchmark_cohort_size, revenue_percentile, net_margin_percentile, benchmarked_at

Play · the library of what has worked

Thirteen properties: play_name, description, category, kpi_moved, autonomy_level, play_status, playbook_url, best_fit_cohort, source_asset_type, source_asset_id, and three agent-owned counters. confidence_score, assignments_count, and assignments_worked are the counters, and the first of those is deliberately absent from every agent tool response.

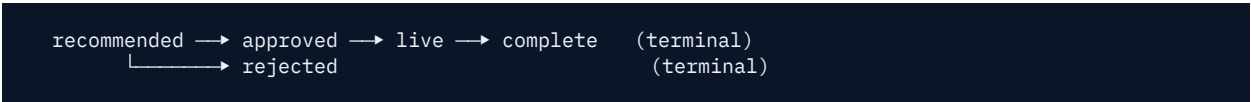
Play Assignment · the spine

Thirty properties across six groups. Identity, lifecycle (stage, stage_changed_at, recommended_at), authorship (recommended_by_agent, recommendation_rationale, draft_summary, draft_asset_id), **the consent terms** (audience_description, from_brand, send_window), **the two gates**, deployment (deployed_asset_id, campaign_id, deployed_at), and measurement (kpi_name, kpi_baseline_value, kpi_result_value, kpi_delta, outcome, measured_at).

The two gates are kept structurally apart: consultant_reviewed_by and consultant_reviewed_at for review, against approved_by_name, approved_by_email, approved_at, and approval_channel for consent. Why that separation is load-bearing is Part 6.2, and it is one of the more instructive parts of this build.

The stage machine

App objects have no pipeline, so stage is a plain enumeration acting as a pipeline substitute.



A schema helper returns the native hs_pipeline_stage when the resolved object reports one and the plain stage enumeration when it does not. Note that hs_pipeline_stage exists on every app object, so its presence proves nothing: the app's own stage enumeration wins.

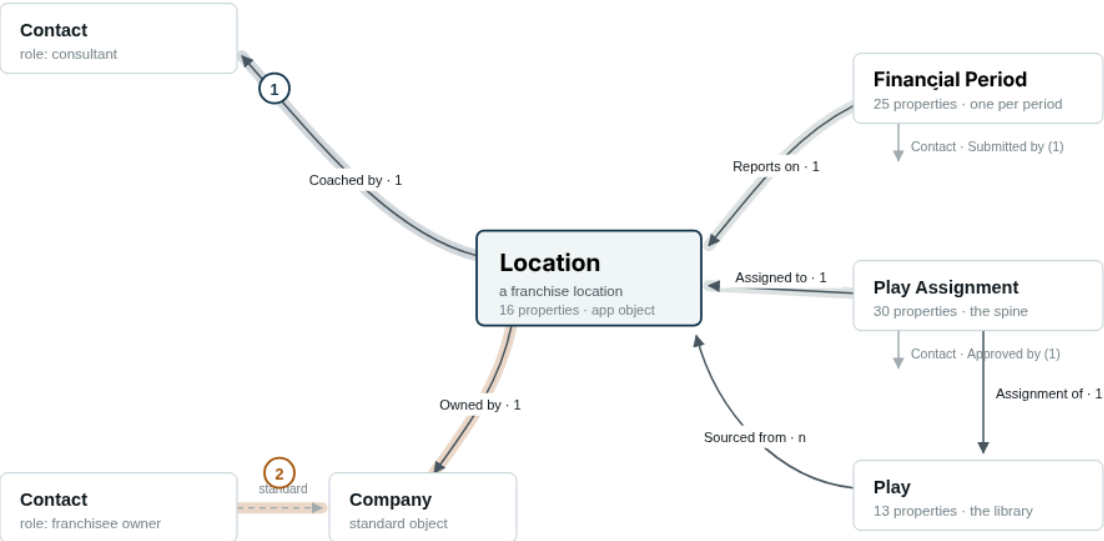
The eight associations, which are also the security model

All eight are INTEGRATOR_DEFINED, declared in the app and created by HubSpot on upload rather than by the setup runner. Both label directions are declared on every edge.

uid	From → To	Label	Limit
fpi- assoc-location-company	Location → Company	Owned by	1
fpi- assoc-location-contact-coach	Location → Contact	Coached by	1
fpi- assoc-financials-location	Financial Period → Location	Reports on	1
fpi- assoc-financials-contact-submitter	Financial Period → Contact	Submitted by	1
fpi- assoc-assignment-play	Play Assignment → Play	Assignment of	1
fpi- assoc-assignment-location	Play Assignment → Location	Assigned to	1
fpi- assoc-assignment-contact-approver	Play Assignment → Contact	Approved by	1

uid	From → To	Label	Limit
fpi- assoc-play-location-source	Play → Location	Sourced from	n

THE ASSOCIATION GRAPH IS THE SECURITY MODEL



- Walk 1, consultant. Contact, Coached by, Unit. Four consultants' books resolve live and are pairwise disjoint.
 - Walk 2, franchisee. Contact, their Company, Owned by, Unit. An owner and a general manager inherit identical reach.
 - Downstream, both. Once a Unit is on the walk, its financials and assignments follow. Nothing else is reachable.
- Login proves identity. The walk proves reach. Identity never implies scope, so every request re-derives it.

Figure 5. The eight associations, and the only two paths through them. Consultant reaches a location by the Coached by edge; franchisee reaches it through their company by the Owned by edge. Nothing else is reachable.

THE ASSOCIATION GRAPH IS THE SECURITY MODEL

There are exactly two isolation walks in the entire product:

Consultant: Contact —*Coached by*→ Location → its financials and its play assignments.

Franchisee: Contact → Company —*Owned by*→ Location → down.

An owner and a general manager associated to the same franchisee company inherit identical reach with no per-person setup. **Login proves identity; the walk proves reach; identity never implies scope**, so every request re-derives it. This is described in full in Part 6.3.

3.5 The 28 app components

Type	Count	Components
APPLICATION	1	fpi-app
APP_OBJECT	4	location, financial-period, play, play-assignment
APP_OBJECT_ASSOCIATION	8	as listed above
CARD	2	fpi-location-intelligence-card, fpi-play-assignment-card
SETTINGS	1	fpi-settings
WEBHOOKS	1	fpi-webhooks
WORKFLOW_ACTION	11	2 workflow actions plus 9 agent tools. A twelfth, fpi-score-location, is merged and pending upload
(held back)	*8*	*APP_EVENT, in app-pending/ with a documented restore procedure*

CRM record cards

Two React and TypeScript UI extensions, both at location: `crm.record.tab`, built on `@hubspot/ui-extensions` and React 18.

- **FPI Location Intelligence**, on Location. Trajectory score, band, drift reason, the last three periods, point-of-sale variance, and open play assignments.
- **FPI Play Assignment**, on Play Assignment. The drafted intervention, the Prescribe rationale with its vault citations, source-play history, and the full approval trail.

The card client calls the Lambda service through `hubspot.fetch()`. Because HubSpot does not return response headers to an extension, everything the caller needs has to be in the body, so the client synthesises a result envelope from the service's bare-body-plus-status convention. **Identity never travels in the body.** HubSpot appends `portalId`, `userId`, `userEmail`, and `appId` to the URL and signs it, and the service reads identity only from the signed query string.

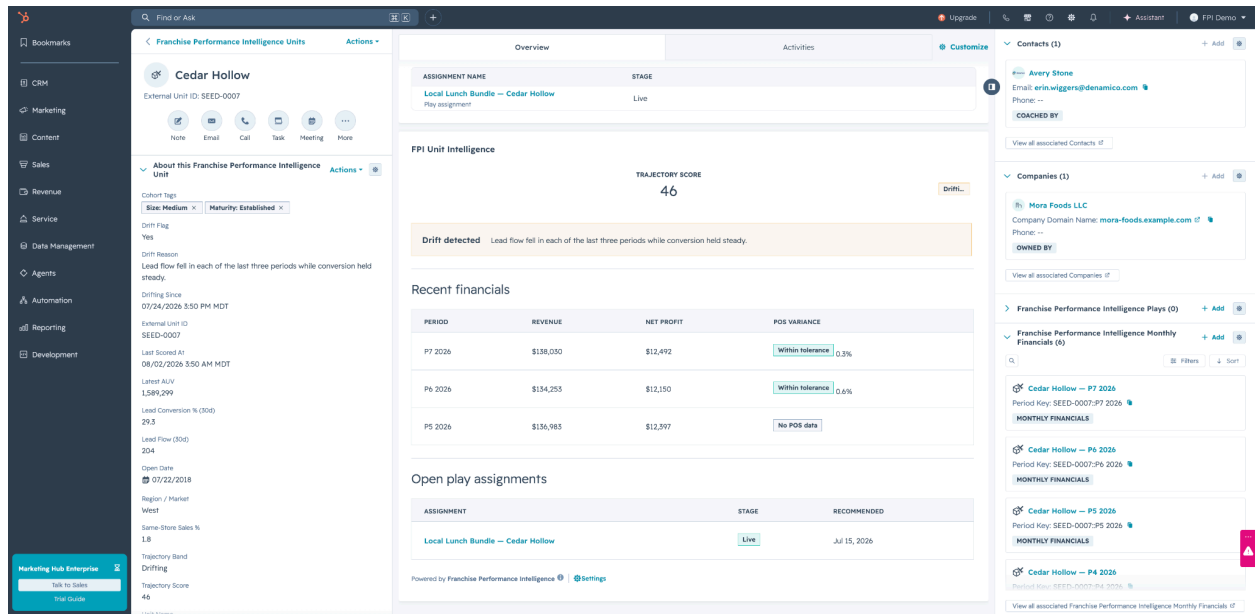


Figure 6. A React UI extension at `crm.record.tab`. It holds no database and no credentials; it calls the Lambda service through `hubspot.fetch()` and reads identity only from the signed query string.

The settings page

Panel	What it does
Brand configuration	Reads and writes the <code>fpi_config</code> HubDB row through the app config endpoints, with client-side validation mirroring the server's
Autonomy	The autonomy dial. In September this is configuration rather than approval: the branch exists and always resolves to gated
Setup and health	The 12-item install checklist: a read-only verify, an async run, and per-item polled status
Agents	Where an administrator records the portal's agent instance ids. Agents cannot be created, listed, or read by any API, so there is nothing for the app to discover and the ids have to be entered

Franchise Performance Intelligence

Account: Erin Wiggers

Overview Insights **Settings**

General Brand configuration Autonomy **Setup & health**

App cards

The install checklist for this portal. Verify is read-only and safe to run any time. Run setup is idempotent and additive-only: it creates whatever verify reports missing, in dependency order, and never deletes or overwrites admin-edited values.

Run verify Run setup

Setup status

ITEM	STATUS	DETAILS
Standard-object properties (four)	OK	All four properties exist on contacts and companies.
App-object property (additional unit access)	OK	additional_unit_access exists on Unit.
Association labels (eight)	OK	All 8 association labels resolve in this portal.
Config table and first row	OK	Table 'fpi_config' (id 372967817) is published and not publicly readable.
Domain detection	OK	Primary domain www.franswistech matches configuration.
Franchise Performance Updates subscription type	OK	Subscription type "Franchise Performance Updates" found (id 3217020839).
Membership access lists (three)	OK	All 3 access lists exist.
Portal theme	OK	franswistech is published, all 5 sentinel files present.
Portal pages (eight)	OK	All 8 portal pages exist, are published, and render from franswistech. Gating is not verifiable here — HubSpot omits publicAccessRules from GET responses. Request a portal URL signed out and confirm it bounces to the membership login.
Automation preflight	OK	All object types resolved and all FPI action ids known. The action catalogue is not readable here (unsupported), so field names are checked by the flow read-back instead.
Workflow import (seven)	Action needed	All 7 are present, but 3 are switched off: W2 - Consultant turnover cascade, W4 - Play lifecycle orchestration, W5 - Drift alerting. Action: Open Automation → Workflows, read W2 - Consultant turnover cascade, W4 - Play lifecycle orchestration, W5 - Drift alerting, and switch each on, they stamp record ownership across every period and assignment, so setup installs them rather than starting them.
Final verify	Action needed	1 check(s) need manual action: workflows. Action: Open Automation → Workflows, read W2 - Consultant turnover cascade, W4 - Play lifecycle orchestration, W5 - Drift alerting, and switch each on, they stamp record ownership across every period and assignment, so setup installs them rather than starting them.

Powered by Franchise Performance Intelligence

Figure 7. What an installing administrator sees. Per-item status rather than one pass or fail, because a portal without Content Hub Enterprise fails the access-list step while everything else succeeds.

Webhooks

Target POST `/v1/webhooks/inbox` with `maxConcurrentRequests: 10`. Four subscriptions: object creation on Financial Period, property change on Location `drift_flag`, property change on Location `last_scored_at`, and property change on Play Assignment stage.

`objectType` accepts app-object component uids, **but only after the objects exist in the portal**, so webhooks deploy after the objects' first deploy. The webhook target host must also resolve in DNS at deploy time, which is the gate that makes the custom-domain script a prerequisite for the first non-demo portal.

3.6 Automation: nine workflows, three custom actions

Key	Name	Enrols	Trigger	Installed
w1a	Child record access stamp (Financials)	Financial Period	object created	disabled
w1b	Child record access stamp (Assignments)	Play Assignment	object created	disabled
w2	Consultant turnover cascade	Location	owner changed	disabled
w3	Financials intake routing	Financial Period	object created	enabled
w4	Play lifecycle orchestration	Play Assignment	stage changed	enabled
w5	Drift alerting	Location	drift flag changed	enabled
w6	Membership provisioning	Contact	fpi_role known	enabled
w7	Location rescore	Financial Period	financials_status changed	*merged, pending deploy*
w8	Executive briefing	Contact (list)	monthly schedule	*merged, installs off*

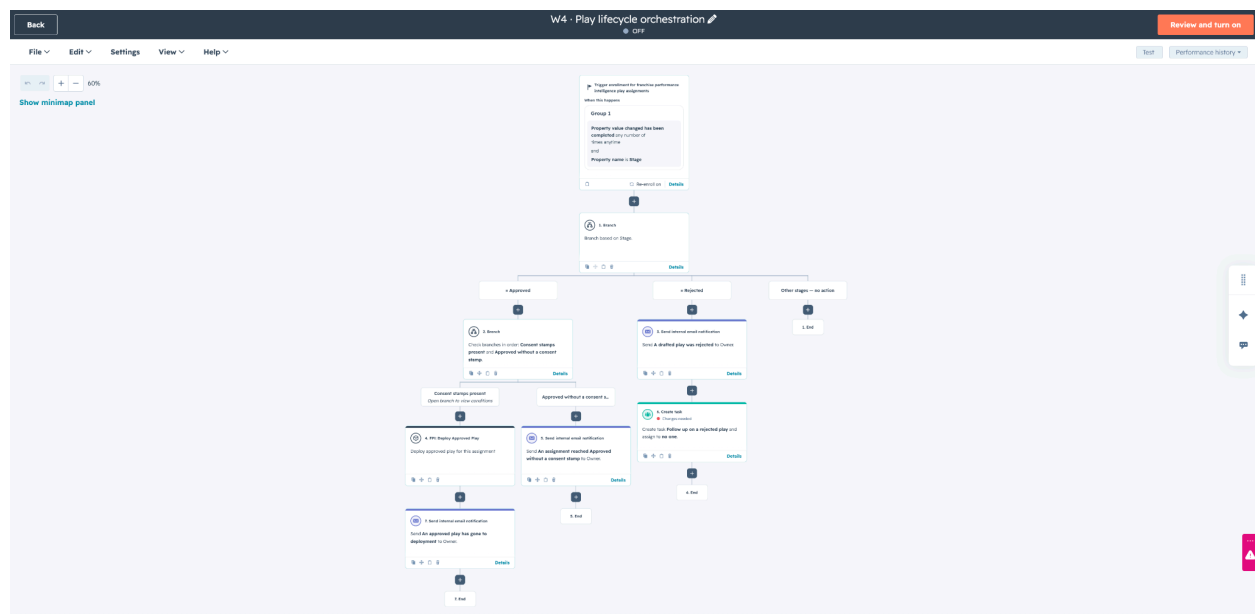


Figure 8. A custom step inside a native workflow. Of the nine flows, three call an app-defined action and everything else is built-in HubSpot functionality.

W1 is two flows because a flow enrolls exactly one object type and the specification lists W1 against two.

The install posture is deliberate. Every flow is created disabled, read back, and only then enabled. The read-back is the only validation the API offers. W1a, W1b, and W2 are left off, because they restamp `hubspot_owner_id` and additional access across every child row a location owns, and ownership is what owned-only visibility is built on. A mis-specified enrolment there changes who can see what. The verify step reports them present-but-off and does not go green until a human has enabled them.

W7 AND W8 EXIST BECAUSE AN AGENT RUN IS A JOB, NOT A STANDING BEHAVIOUR

A Breeze agent cannot be relied on to run itself. It runs when a person picks it in the agent inbox and supplies its declared inputs, when a builder tests or runs it, when a workflow invokes it, or from a record. **Nothing continuous can depend on a consultant thinking to ask**, so the fleet's standing work needed a home that was not a person's memory.

W7 · Location rescore carries the deterministic scoring. It enrolls on financials status changing, deliberately sequenced behind W3, so a point-of-sale feed stub never triggers a score against incomplete data. It calls a third custom action, `fpi-score-location`.

W8 · Executive briefing runs the executive agent monthly through HubSpot's built-in Run agent action. It is the one agent job that automates cleanly, because an executive scope carries no location ids by contract and therefore has no per-user book to get wrong. **It installs switched off on purpose:** it spends the customer's AI credits and it reaches their executives, and neither should begin without a decision.

Why the scoring pass is a workflow action and not an agent tool

Six Location properties carry the scoring output, and until this work none of them had a writer anywhere in the system. Every surface reading them had been finished for months and was quietly rendering its empty state: cockpit ordering fell back to arbitrary order, all four band tiles read zero, the executive distribution was entirely unbanded, both Monitor-role tools answered that nothing had been scored, and **W5 was a drift-alerting workflow with nothing in the system able to fire it.**

It could not be an agent tool, and the reason is structural rather than preferential. The agent plane resolves identity down to one Contact holding an FPI role and re-derives that person's walk. **A portal-wide scoring pass has no consultant to be.** So it is a deterministic app workflow action, which is also the honest description of the work: scoring is arithmetic, not judgement. The arithmetic is cohort-relative where a cohort exists and trend-based where one does not, and a missing component is dropped rather than zeroed, so a location is not penalised for a gap in its own data.

A DEFINITION THE BUILD HAD TO CORRECT, AND IT IS WORTH BEING PLAIN ABOUT

The original brief defined drift as **a leading indicator moving before the lagging metrics do**. That turned out not to be computable. The two lead properties are scalars the point-of-sale feed overwrites in place, carrying no history, so there is nothing for a movement to be measured against.

Drift is therefore defined as **measured deterioration in per-period financials and in cohort standing**. That is a weaker claim than the brief made, and it is the one the system can actually

support. The product still moves discovery forward by a reporting period, because a location is rescored the moment its financials land rather than when the royalty view catches up. It simply does not do it the way the brief said it would.

Why an app action rather than the native Edit records action

The owner cascade copies a property **across an association to an app object**, and the turnover direction is one-to-many. No documented flow construct expresses either: the token format naming a property fetched from an associated record appears in no HubSpot document, specification example, or public repository, and nothing suggests the native action fans out. Custom code actions were not an option, because they require Data Hub Professional.

So a second app-defined action ships: `fpi-stamp-access`. The flow supplies the trigger, which the API does document, and the handler walks the association and writes idempotently, skipping records that already carry the value, capped per run with the cap reported rather than hidden.

WHAT THE FLOWS API WOULD NOT TELL US

The flows API types an action's `fields` as an unconstrained object. **The v4 flows API publishes no contract for the one part of a workflow that decides what it does.** HubSpot's own guidance for learning an action's field names is to build it in the UI and read the flow back. There is no dry run and no validate-only parameter, and a definition wired to a field name the action does not recognise is accepted, runs, and does nothing, permanently and silently.

Three compounding unknowns followed from that, none resolvable from documentation, and **all three were answered by the first live install on 25 July**: an app object can be a workflow's enrolled object type; the `objectTypeId` takes the numeric form; and an app's own action type id belongs to the **app** rather than the portal, which is why resolution moved to deploy time.

3.7 The member portal: a Content Hub theme built to depth

The portal is where a franchisee does the only work the system asks of them, which is why it carries more design weight than any other surface.

Layer	Count
Theme	franwise 1.8.0, flat self-contained fork, no <code>extends</code> key
Portal page templates	8
Membership and system templates	11 (login, register, reset, password prompt, subscription preferences, unsubscribe, confirmation, search, 404, 500)
Custom modules	30 (21 carrying their own JavaScript)
HubL macros	56 across 6 files, covering all 31 design-system components
CMS serverless endpoints	3
Browser layer	1,399 lines of progressive enhancement over server-rendered HTML
CSS	A token file scoped under a single class, and a component file containing no colour literal , checked by the inspection suite
Theme HubL, JS and CSS	~14,950 lines
Structural inspection checks	52 passing, 0 failing

Page	Slug	Audience
portal-home	/portal	consultant, franchisee, executive
cockpit	/portal/dashboard	consultant only
location-detail	/portal/location?id=	consultant, franchisee
my-business	/portal/my-business	franchisee only
pnl-submit	/portal/submit-pnl	franchisee only
play-review	/portal/play?id=	consultant, franchisee
exec-dashboard	/portal/executive	executive only
access-denied	/portal/access-denied	all

FIGURE 9

Screenshots, side by side: the P&L submission form and the benchmark mirror a franchisee receives back

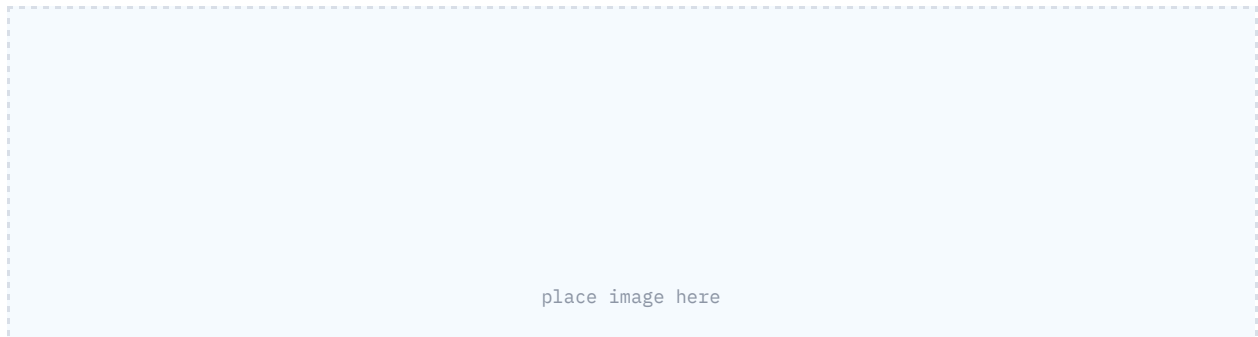


Figure 9. The exchange that makes the data arrive. The franchisee submits on the left and gets the right back in the same interaction, benchmarked against an anonymised cohort that always carries its size.

The four rules the theme cannot violate

- 1. Every template that renders record data runs the isolation guard, before render.** A denial renders the access-denied page with a not-found posture: byte-identical output for a nonexistent record and an out-of-walk record, with no record identifier in the page, the title, or the URL.
- 2. `fields.json` exposes identity, never semantics.** Logo, brand primary and secondary, ink-on-primary, typeface. The four status colours, the chassis neutrals, the type scale and the density are never exposed, and the setup runner must not expose them either. If a franchisor could configure the at-risk red to their brand red, the consultant cockpit stops working.
- 3. No franchisee-facing surface renders a trajectory band, a drift flag, or another named location's numbers.** Bands are consultant coaching signals. Benchmarks do the franchisee-facing work.
- 4. Consent copy sits adjacent to the control,** in the same visual block and the same viewport at every supported width, naming what is sent, to whom, from which brand, over what period, and what happens on decline. The stamped record is shown back to the member after they approve.

3.8 HubSpot APIs consumed

Extracted from the service's HubSpot client modules and handlers. This is the complete list.

CRM

```
GET    /crm/v3/schemas/{objectType}
GET    /crm/v3/objects/{objectType}/{id}
POST   /crm/v3/objects/{objectType}
PATCH /crm/v3/objects/{objectType}/{id}
POST   /crm/v3/objects/{objectType}/batch/read
POST   /crm/v3/objects/{objectType}/batch/upsert    (idProperty per input)
POST   /crm/v3/objects/{objectType}/batch/update
POST   /crm/v3/objects/{objectType}/search
GET    /crm/v3/properties/{objectType}/{name}
POST   /crm/v3/properties/{objectType}
GET    /crm/v3/owners
POST   /crm/v4/associations/{from}/{to}/batch/read
GET    /crm/v4/associations/{from}/{to}/labels
PUT    /crm/v4/objects/{type}/{id}/associations/{toType}/{toId}
POST   /crm/v3/lists                · POST /crm/v3/lists/search
PUT    /crm/v3/lists/{id}/memberships/add
```

CMS AND CONTENT

```
GET/POST/PATCH /cms/v3/hubdb/tables      · /{t}/draft · /{t}/draft/publish
GET/POST/PATCH /cms/v3/hubdb/tables/{t}/rows · /rows/draft · /rows/{id}/draft
GET            /cms/v3/domains
PUT            /cms/source-code/2026-03/{env}/content/{path}
GET            /cms/source-code/2026-03/{env}/metadata/{path}
POST           /cms/source-code/2026-03/{env}/validate/{path}    (compile-only)
POST           /cms/source-code/2026-03/extract/async · /extract/{path}
GET            /cms/source-code/2026-03/extract/async/tasks/{id}/status
GET/POST/PATCH /cms/pages/2026-03/site-pages/{id}
PATCH         /cms/pages/2026-03/site-pages/{id}/draft
POST           /cms/pages/2026-03/site-pages/{id}/draft/push-live
POST           /cms/pages/2026-03/site-pages/schedule
GET/PATCH     /cms/v3/pages/landing-pages/{id}    (+ /draft)
```

MARKETING AND PREFERENCES

```
POST    /marketing/v3/emails/clone
GET/PATCH /marketing/v3/emails/{id}
PUT      /marketing/v3/campaigns/{id}/assets/{type}/{assetId}
GET      /communication-preferences/v3/definitions
```

AUTOMATION

```
POST    /automation/v4/flows                · GET/PUT /automation/v4/flows/{id}
POST    /automation/v4/actions/callbacks/{callbackId}/complete
GET      /automation/v4/actions/{appId}      (developer-scoped)
GET      /automation/actions/2026-03/{appId} (developer-scoped)
GET      /automation/2026-09-beta/action-types (documented, not deployed)
```

AUTH AND EVENTS

```

POST /oauth/v3/token
POST /integrators/timeline/v4/events (app events, suppressed)
POST /integrators/timeline/v4/events/batch

```

Client behaviour

The HubSpot client carries a per-portal OAuth bearer with automatic token refresh, and handles 429 responses by **branching on the policy name**, so that a daily-limit breach stops rather than backing off into a wall. The `retry-after` header is honoured with a ten-second floor, and there is no retry on any 4xx other than 429 and 423.

3.9 OAuth scopes: 44 requested

```

oauth
crm.objects.contacts.read / .write
crm.objects.companies.read / .write
crm.objects.owners.read
crm.schemas.contacts.write      crm.schemas.companies.write
automation      content      marketing-email
marketing.campaigns.read / .write
hubdb      cms.membership.access_groups.write
communication_preferences.read_write      cms.domains.read
crm.lists.read / .write
crm.app.objects.a46836361_{location|financial_period|play|play_assignment}
      .{view,create,edit,merge,delete}
crm.app.schemas.a46836361_{...}.read
crm.app.schemas.a46836361_{...}.properties.write

```

Three scope facts worth documenting

- App-object scopes use **lowercase** object names while the fully-qualified name is upper snake case, and they are **generated at first upload**. Adding an object is therefore a two-phase upload: upload, copy the generated scopes out of the CLI log, paste them into the required scopes, upload again.
- `timeline` is **absent on purpose**. It was removed along with the app events, because a scope that forces every installed portal to re-authorise should not be requested for a feature that is not there.
- ``cms.source_code.read`` and ``write`` **do not exist on 2026.03**. The string appears zero times in the scopes reference. `content` is the only scope that reaches the Source Code API, which is why theme installation needs no scope change and no reinstall. An unknown scope string passes local validation and fails server-side at upload, which is a sharp edge worth knowing about.

3.10 The five classes of build-time-baked URL

These live in HubSpot app configuration, so changing them means an app redeploy. All five currently carry the raw API gateway host across twelve files.

#	Where	Path
1	<code>app.config.auth.redirectUrls[]</code>	<code>/v1/oauth/callback</code>
2	<code>app.config.permittedUrls.fetch[]</code>	<code>/v1/</code> — a prefix allowlist, so one entry covers every route
3	<code>webhooks.config.settings.targetUrl</code>	<code>/v1/webhooks/inbox</code>
4	<code>workflow-action.config.actionUrl</code> ×2	<code>/v1/workflow/deploy,</code> <code>/v1/workflow/stamp-access</code>
5	<code>actionUrl</code> on the nine agent tools	<code>/v1/agent/*</code>

4 The agent layer

This is where the contest's subject matter lives, so it is documented at the level of the individual tool, the identity plane each tool trusts, and the refusal contract that governs what happens when a tool says no.

4.1 Nine agent tools

#	Shown to the agent as	Type	Loop role	Endpoint
1	Find locations that need attention	GET_DATA	Monitor	/v1/agent/locations-needing-attention
2	Get location performance summary	GET_DATA	Monitor	/v1/agent/location-summary
3	Find plays matching a location	GET_DATA	Router	/v1/agent/matching-plays
4	Compare a location to its cohort	GET_DATA	Benchmark	/v1/agent/cohort-comparison
5	Draft a play recommendation	TAKE_ACTION	Prescribe	/v1/agent/draft-recommendation
6	Find outcomes across a book	GET_DATA	Loop	/v1/agent/book-outcomes
7	Check filing status across a book	GET_DATA	Monitor	/v1/agent/book-filing-status
8	Draft a play candidate	TAKE_ACTION	Learn	/v1/agent/draft-play-candidate
9	Brief the network	GET_DATA	Revenue Clarity	/v1/agent/network-briefing

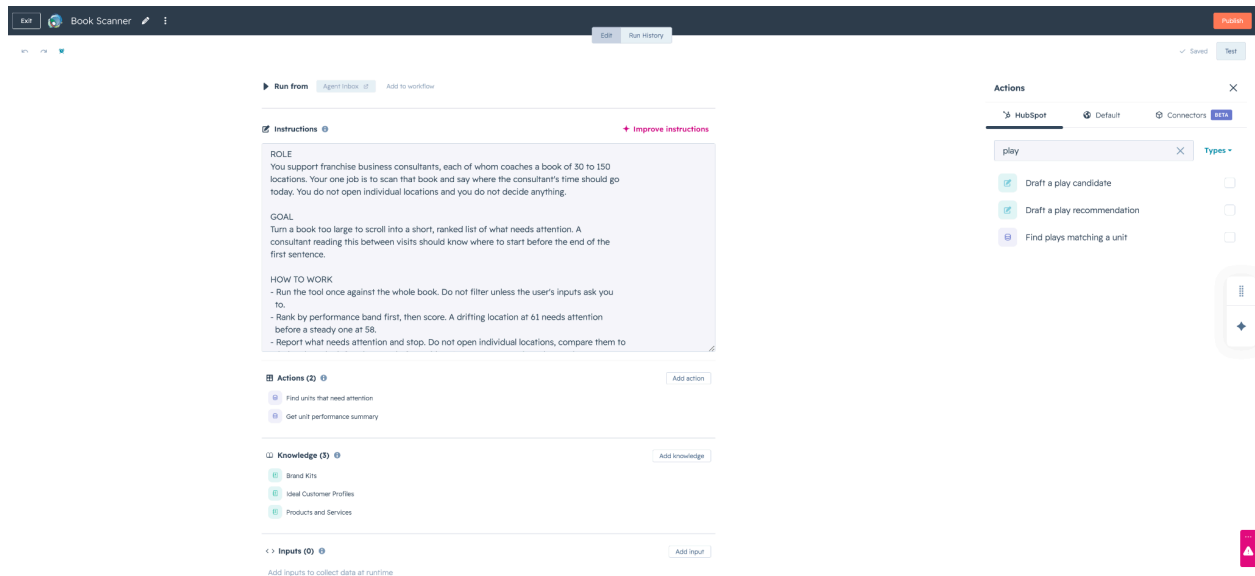


Figure 10. All nine published and visible to an agent builder. They sit under Franchise Performance Intelligence rather than Denamico, because nothing a franchise customer sees carries the partner brand.

All nine sit under the display name "Franchise Performance Intelligence", which is the programme name rather than the partner name, because nothing a franchise customer sees should carry Denamico's branding. All are published, all declare no object types, and all use a single supported value type.

NOTHING IN THESE TOOLS IS NEW BUSINESS LOGIC

Every one of the nine reuses the same walk resolver, location loader, active-play loader, cockpit comparator, and benchmark snapshot that the member portal and the CRM cards already read. A new caller class got a new **consumer**, not a new branch inside the modules that decide who may see what. That is the single most important architectural decision in the agent layer and it is what makes the security argument in Part 6 hold across all four planes at once.

Ordering is not cosmetic

Tool 5 depends on ids produced by tools 1 or 2, and on a rationale informed by tools 3 and 4. Tool 8 depends on a location id from tools 1 or 2 in the same way. HubSpot's own guidance names tool sequencing as the thing agents get wrong unaided, which is why the reference agent instructions state the order explicitly rather than trusting the tool descriptions to imply it.

Tool 9 belongs on a different agent

Network briefing is refused to consultants server-side. Binding it to the consultant agent would put a tool on the picker that always refuses, which reads as a broken install rather than as a boundary. It goes on a separate executive agent.

4.2 The Breeze agent roster, and a finding that reshaped it

The original design called for one conversational agent that sequenced tools across a conversation. Building against the actual Agent Hub surface reversed that decision.

HUBSPOT AGENT HUB AGENTS ARE SINGLE-SHOT TASK RUNNERS, NOT CONVERSATIONAL ASSISTANTS

A run is instructions plus typed inputs, executed once. There is no user turn to read a subject from and no second hop. The "one agent, not five" decision rested on sequencing across a conversation, and with no conversation, **agents are tasks rather than personas.**

HubSpot offers no general "chat with my custom agent" surface at all. A custom agent runs in exactly four ways: **a person picks it in the agent inbox and supplies its declared inputs; a builder tests or runs it in the agent builder; a workflow invokes it through the Run agent action, including on a schedule; or it runs from a record.** Any description of this product in which a consultant converses with an agent is wrong.

This is also strictly better for the contest's per-agent activation mathematics than one agent was, which is a happy accident rather than the reason for the change.

#	Agent	Serves	Inputs	Tools it holds	Status
1	Book Scan	consultant	none	locations-needing-attention	Built, verified
2	Location Briefing	consultant	<code>location_id</code>	location-summary, cohort-comparison, matching-plays	Built, verified
3	Draft Recommendation	consultant	<code>location_id</code> , <code>play_id</code>	the three above plus draft-recommendation	Specced, in build
4	Play Outcome Review	consultant	<code>kpi</code> (optional)	book-outcomes, location-summary	Designed
5	Draft a Play Candidate	consultant	source location, name, what was done, evidence, KPI moved	draft-play-candidate, book-outcomes, location-summary	Designed
6	Period Filing Check	consultant	<code>period_label</code> (optional)	book-filing-status, locations-needing-attention	Designed
7	Network Performance Briefing	executive	none	network-briefing	Designed

SEVEN AGENTS, NINE PUBLISHED TOOLS

	units-needing-attention	unit-summary	matching-plays	cohort-comparison	draft-recommendation	book-outcomes	book-filing status	draft-play-candidate	network-briefing	
TOOL TYPE	READ	READ	READ	READ	ACTION	READ	READ	ACTION	READ	STATUS
1 · Book Scan consultant · no inputs	●									■ Built and verified
2 · Location Briefing consultant · unit_id		●	●	●						■ Built and verified
3 · Draft Recommendation consultant · unit_id, play_id		●	●	●	◆					■ In build
4 · Play Outcome Review consultant · kpi, optional		●				●				□ Designed
5 · Draft a Play Candidate consultant · source unit, name, evidence		●				●		◆		□ Designed
6 · Period Filing Check consultant · period_label, optional	●						●			□ Designed
7 · Network Performance Briefing executive · no inputs									●	□ Designed

● read tool ◆ action tool All nine tools are deployed and published. That half is code, and it is done.
 Binding a tool to an agent is portal configuration, which HubSpot exposes no API for: 10 to 15 selections per portal.
 Ordering is not cosmetic: draft-recommendation depends on ids from tools 1 or 2, and on a rationale informed by 3 and 4.

Figure 11. The fleet against the tool surface. All nine tools are deployed and published; the agents that hold them are portal configuration, which is why their statuses differ.

All nine tools are deployed and published. Agents 1 and 2 are built and verified against live tool output, agent 3 is in build, and agents 4 through 7 are designed with their instruction blocks written. The distinction matters: **the tools are the platform work and they are done; the agents are configuration work that HubSpot does not expose an API for**, which is Part 4.7.

AGENTS 4 AND 5 ARE DELIBERATELY TWO AGENTS RATHER THAN ONE

Agent 4 produces the observation across a consultant's book: which plays came back worked, which did not, which are still open, and which were deployed and never measured. A human then decides which of those patterns is worth turning into a reusable play. Agent 5 drafts it.

The constraint behind the split is general and worth stating: **a job that needs a human decision partway through is two agents with a person between them, not one agent with a confirmation step.** Collapsing them would put the judgement inside the model.

Agent 7 is what makes an executive agent possible at all. All the earlier tools are consultant-scoped by construction: the scope resolver returns an empty scope for an executive, so the book-scan tool hits its executive branch and every other tool refuses every id. An executive agent built on those tools would answer nothing. Network briefing is the route that changes that, and it is the one agent whose native-webhook twin is worth building unconditionally, because an executive scope carries no location ids by contract and therefore needs no per-person configuration.

What the two live agents actually did

Agent 1 returned 13 coached locations for a seeded consultant, correctly ranked by band and then by score, and quoted a two-day-old trajectory score **as of its date** rather than as a current figure. Agent 2 ran all three of its tools against one location, reported a suppressed benchmark as insufficient reporting rather than as poor performance, listed three plays as two-count records with proven-asset flags, and reasoned from "lead flow weak, conversion steady" to a demand problem rather than a conversion problem, then picked the play whose target metric matched.

The screenshot displays the Denamico 'Location Briefing' interface. On the left, a 'Test runs' sidebar shows a list of saved tests with status indicators (Started, Done, Error, Action needed). The main content area is titled 'Location Briefing - Simulated test' and shows a 'Content' analysis of a test run. The analysis includes a list of findings, such as 'Where it stands - Drifting, score 46 as of Aug 2, 2026' and 'Against comparable locations - For P7 2026 (Jun 18, 2026-Jul 15, 2026), too few comparable locations had reported to produce a benchmark'. Below the content, 'Execution outputs' are shown, including a detailed analysis of the 'Where it stands' finding, noting that the drift is being driven by lead flow falling in each of the last three periods while conversion has held steady.

Figure 12. Agent 2 against a live location. Note the suppressed benchmark reported as insufficient reporting rather than poor performance, and the play record quoted as a two-count rather than a percentage.

That last step is real diagnostic work, and it is the thing the consultant-as-hero narrative is actually selling.

Two platform findings from building the agents

- **App objects are not available as agent inputs.** The input picker offers Text, Number, Checkbox, and CRM Object limited to the standard objects. So a Location cannot be a record-reference input, and the consultant pastes a record id that Agent 1 handed them. This is a platform gap worth reporting.
- **Agent tools restore the invoking user's identity.** The invoking user's email is in the agent-tool payload, so the scope resolver resolves the actual person and **one agent serves every consultant**. This retired an entire branch of deployment design that would otherwise have required cloning an agent per consultant.

Instruction patterns that measurably worked

Each of these was written in response to an observed failure against real tool output, and each one fixed it. They are included because agent instruction design is the least documented part of building on this platform.

Instruction	The failure it fixed
"Run all three even if the first looks unremarkable."	The agent short-circuited on a steady summary and skipped the cohort check, which is exactly where a comfortable-looking location turns out to be sliding.
"Locations with no score at all are a separate state from a low score. Never drop them."	The agent silently dropped 9 of 13 locations by deciding "needs attention" meant at-risk plus drifting. Not-yet-scored is the most neglected state and it had become invisible.
"Do not cite a bare figure as support for a play unless something in the data establishes that the figure is a problem."	The agent cited a 9.1% net margin as support for a margin play, with no benchmark to say whether 9.1% was good.
"Relay the refusal as given and stop. Do not retry with different inputs."	An agent that hits a permission refusal and rephrases is defeating the guard.
"Report what needs attention and stop."	Once agents became single-purpose, Agent 1 wandered into Agent 2's job without this.
"Round currency to whole locations and percentages to one decimal."	Platform-calculated properties return unrounded, and 7.7289% reads as a spreadsheet leaking through.

4.3 Four authorization planes

The same question, "may this caller see this record," is answered four different ways depending on who is asking and what proves they are who they say they are. Setting these out explicitly is the clearest way to show why an agent tool is not simply another API endpoint.

Plane	Caller	Identity comes from	Walk?
Internal CRM	A card via <code>hubspot.fetch()</code>	signed query string	No. The card only renders on a record the user already opened, so HubSpot's permission sets governed access first
Member	A portal contact	member JWT	Yes , re-derived per request
Agent	A HubSpot user, via a language model	signed body	Yes , re-derived per request
Native webhook	The same, via a stock webhook action	signed query string	Yes , re-derived per request

MAY THIS CALLER SEE THIS RECORD? FOUR PLANES, FOUR ANSWERS

	INTERNAL CRM	MEMBER	AGENT	NATIVE WEBHOOK
CALLER	A React card in the CRM, via <code>hubspot.fetch()</code>	A portal contact, logged in with a password	A HubSpot user, via a language model	The same user, via the stock webhook action
IDENTITY COMES FROM	Signed query string on every card fetch	Member JWT HS256, 15 minutes	Signed body carrying the invoking user	Signed query string administrator-typed
WHAT GOVERNED ACCESS FIRST	HubSpot's own permission sets. The card only renders on a record the user had already opened.	Nothing upstream. The broker proves only that someone is logged in, and the customer can edit it.	Nothing. The record id reaches the endpoint because the agent put it there.	Nothing. The URL is administrator-typed configuration the agent cannot influence.
WALK RE-DERIVED, EVERY REQUEST	No HubSpot governed it first	Yes before any read or write	Yes before any read or write	Yes before any read or write

THE CRUX

A CRM card is safe partly because of where it renders. An agent tool has none of that: the endpoint accepts arbitrary input, and the record id arrives because the agent chose it, from a declared input a person typed or from its own reasoning over an earlier result.

The tool is the boundary.

Every handler requires the agent caller proof, resolves scope from the invoking user's email through the Contact record and its role, and asserts the record is in reach before any read. No role or an unrecognised role refuses, and it never falls back to everything.

The body is untrusted for identity and trusted for subject.

Figure 13. The same question, answered four ways. Only the internal CRM plane inherits an upstream permission check; the other three re-derive the walk on every request.

THE CARD'S REASONING DOES NOT TRANSFER TO AN AGENT, AND THIS IS THE CRUX

A CRM card is safe partly because of where it renders: on a record the user already opened, which means HubSpot's own permissions governed access before the card existed. **An agent tool has none of that.** The record id reaches the endpoint because the agent put it there, whether from a declared input a person typed or from its own reasoning across an earlier tool result, and the endpoint accepts arbitrary input.

HubSpot's own documentation is explicit that an agent has no access to CRM data unless explicitly given tools to get it. **The tool is the boundary.** Every tool handler therefore requires the agent caller proof, resolves the scope from the invoking user's email through the Contact record and its role, and asserts the record is in reach **before** any record read. No match, no role, or an unrecognised role refuses. It never falls back to everything.

4.4 What binds every tool, enforced in the payload rather than the prompt

A language model will repeat whatever it is handed, which is why most of these constraints are structural rather than instructional. A rule that lives only in a prompt is a rule that survives until someone edits the prompt.

Rule	How it is enforced
Confidence is history, never a score	Tools return <code>times_used</code> and <code>times_it_worked</code> . <code>confidence_score</code> exists on the Play and is deliberately not in any tool response
A percentile never appears without its cohort size	Both are returned in the same object; the action description instructs the agent to quote them together
No peer is ever named	Aggregate-only by return type. The anonymity floor is an absolute minimum of 3 and a default of 5
Every score carries its date	<code>last_scored_at</code> is on every location in every response
Periods carry their range	<code>period_start</code> and <code>period_end</code> , never a bare month
Nothing is communicated by colour alone	Bands are returned as text values
Agents are attributed, never anthropomorphised	No tool returns a persona, a first-person string, or an agent name for member display
The autonomy dial stays gated	<code>draft-recommendation</code> has no branch producing anything but a recommended state
No agent writes a consent stamp	The drafting tools write none of the approval properties
No agent writes an agent-owned score	Asserted in test against the guard list itself, not against a restatement of it
A play an agent wrote cannot be recommended	<code>draft-play-candidate</code> writes candidate status; the active-play loader filters to active

Rule	How it is enforced
A count an agent repeats has to reconcile	<code>book-outcomes</code> four buckets plus declined sum to the assignment count, asserted in test
A tool says when it could not answer	<code>book-filing-status</code> refuses explicitly where the executive dashboard degrades to empty tiles
Roles reach different things	Explicit refusal codes on both writes, and on the network briefing for non-executives

The reason the arithmetic rule is in that list is worth stating: **an agent handed arithmetic that does not reconcile improvises the difference.** That was observed, not theorised.

4.5 The refusal contract

A tool refuses with **HTTP 200, an execution state of `FAIL\_CONTINUE`, an error message, and an error code**. The reason it is a 200 rather than a 4xx is that HubSpot documents that an agent cannot request information from the user after it has been invoked, so a non-2xx leaves the agent with nothing to work with, and it will improvise.

THE REFUSAL HELPER TAKES A CODE AND NO MESSAGE, AND THAT IS AN INVARIANT RATHER THAN A STYLE CHOICE

A constants table maps each code to its exact words. One code answers both "outside your book" and "no such record". **If the message were an argument, the two call sites could drift apart by a single word, and the tool would become a membership oracle for every id a language model cared to guess.** Looking the message up by code makes byte-identity structural.

One thing was deliberately **not** equalised: timing. The guard refuses before the record is fetched, so an out-of-book id is measurably faster than an in-book id that does not exist. Equalising that would mean fetching records the caller cannot reach, which is exactly what the guard forbids. The bytes are identical, the timing is not, and that trade is the right way round.

A declaration gap was found and closed on this path. All three components that declared execution rules conditioned on the error code, and **none of them declared the error code in its output fields**, which means every refusal message already written would have been dead copy. All eleven workflow actions now declare it, and a 42-assertion test fails if that stops being true.

4.6 The second door: native webhooks

HubSpot's agent builder ships a stock "Send a webhook" action. It is not gated on any beta, an administrator configures it in the agent's actions list, and it can point at endpoints that already exist. So the same nine tools are reachable twice. **Both routes run the same handler**, and that is structural rather

than careful: each tool's work is split into an inner run that never sees the request, so a second envelope could not become a second copy of a tool.

What a captured invocation settled

A real native agent-webhook invocation was captured and both of its signature headers were verified offline against the app client secret. Both matched. Three consequences followed:

- 1. No legacy verifier was written.** The existing v3 verifier handles a native invocation unmodified. The legacy version header describes the companion header, not the request. Holding that line rather than adding a fallback on a guess cost nothing and preserved a replay window a legacy-only path could not offer.
- 2. The replay window survives,** because the modern signature carries a timestamp and the legacy one does not.
- 3. No route was loosened.** A request carrying only the valid legacy headers is refused, and that is pinned by a test.

THE LINE, STATED ONCE

The body is untrusted for identity and trusted for subject.

A location id the agent chose is fine, because the guard authorises it against the walk rather than believing it. A user email the agent chose is not, because nothing downstream re-checks it. An agent tool puts the invoking user's email in the body and the signature covers the body, so the body is trustworthy there. A native webhook body is composed key by key in the agent builder, each key either a static value an administrator typed or one the agent fills at run time, and **the request does not say which**. So on that route, identity travels in the URL, which is administrator-typed configuration the agent cannot influence and which the signature also covers.

The cost is a scaling decision rather than a security one. An administrator-typed URL is static, so one configured webhook action reaches one consultant's book. Correct isolation, coarser granularity. Neither captured invocation carried any identity for the invoking user, and the reference confirms it from the other side: an action's configured values are static or object-property, and there is no current-user token. So the native route **structurally cannot** resolve who invoked it. That leaves exactly three options, and the agent-tool path is the one that scales.

4.7 Tools yes, agents no: a platform finding with a cost attached

Agents themselves cannot be created programmatically. This was researched and then verified adversarially in both directions, and the evidence is affirmative rather than merely absent.

- Requesting the feature config schema for an agent type returns "unable to locate JSON schema", identical to the response for a deliberately bogus type, while the workflow-action and app-event types return real schemas.

- HubSpot's machine-generated specification catalogue lists 183 published API specs with **zero occurrences** of the word agent.
- Sixteen plausible endpoint paths all return 404 Resource not found, while a control endpoint returns 401. **A hidden-but-callable API would return 401.**
- The project components configuration defines exactly 15 component types, and none is an agent.

	Automatable	Manual, UI only
Tool definition, inputs, outputs, action URL	✓ hsmeta + upload	
Tool visibility in an installed portal	✓ published flag	
Tool execution and callbacks	✓ HTTP	
Agent existence		✗
Agent instructions, knowledge, inputs		✗
Binding a tool to an agent		✗
The human-review toggle on an action tool		✗ (and the customer can switch it off)

What was built to work around it

If agents cannot be created, listed, or read by any API, then nothing in the app can discover which agent is which in an installed portal. So the ids became configuration: `agent_id_executive` and `agent_id_consultant` are columns on the brand configuration table, entered by an administrator on a new **Agents** tab in the app settings page, alongside brand configuration, autonomy, and setup.

This is what lets a scheduled workflow invoke an agent at all. The executive briefing flow is held back from naming its agent token until an id is set, so the flow cannot install pointing at nothing. **An agent instance id is per-portal**, and stays per-portal even if agents are one day distributed centrally, because the instance is created inside the account.

The reference agent, and what it is forbidden to say

The instruction text is a reviewed repository artifact, treated as source precisely because it cannot be deployed. Its prohibitions are duplicated into each tool's action description, so that a tool bound to a different agent still carries them. Agent instructions are a second layer of defence, never the only one.

- Never say a play is proven, guaranteed, recommended by the system, or has a success rate. State its record: used with 14 locations, moved the metric for 11.
- Never convert that pair into a percentage or a confidence level.
- Never quote a percentile without saying how many locations it was computed across.

- Never name, describe, or speculate about another franchise location.
- Never say anything has been sent, scheduled, deployed, or approved.
- Never describe a location only by colour.
- Do not refer to yourself as a person, give yourself a name, or use "I think" or "I recommend".

An EU AI Act read, made as a decision rather than found in review

Agent tools must not support use cases the EU AI Act treats as unacceptable or high risk. The listing requirements name social scoring and employment decisions as examples, and ask that tools empower front-office use cases.

FPI scores the performance of independently owned businesses and routes coaching to their owners. That is defensible: it grades **business locations, not people**, and field marketing and account management are front office. The build keeps it that way deliberately. Every tool operates on a Location record, none takes a person as a subject, no tool name or description uses grading language, and nothing returns an individual assessment. It is close enough to the line that it should be a decision made in advance rather than a finding discovered in review, which is why it is written down here.

4.8 What turned out not to be an agent

Designing the rest of the fleet removed two things the original system design had assumed would be agents. Both removals are more informative than the agents that survived, because each one is a case where the obvious agentic framing was wrong for a specific, checkable reason.

P&L intake cannot be an agent

The submitter is the franchisee. Franchisees hold no HubSpot seat and must never reach an agent, and the agent scope resolver refuses anything that is not a consultant or an executive. The submit handler runs the franchisee walk only, and the reason is in the association it writes: the "Submitted by" edge carries the caller. **A consultant filing on a franchisee's behalf would put the wrong name on the owner's numbers**, which the intake routing workflow and the audit trail would both read as the owner asserting those figures.

So the upload-and-pre-fill step is a member-plane endpoint instead: the portal accepts the upload, a Lambda extracts a draft, the form pre-fills, and the franchisee confirms through the existing submit route. **The confirm step does not move.** What is genuinely an agent in this area is the consultant's chase list, which is agent 6.

The router's cohort assignment belongs to measurement

The best-fit-cohort property decides which locations a play is offered to, because the matching tool ranks on it and the suggestion path filters on it. Whatever writes that property steers the entire network. The evidence for it already exists in the system: the cohort tags of locations where measured assignments came back as worked.

So it should be computed in the measurement roll-up alongside the assignment counters and confidence, and added to the agent-owned property guard rather than being written by an agent. A new play carries no cohort, and the matching tool already treats an empty cohort as general-purpose and ranks it last. **A play earns its cohort by working somewhere.** No schema change is needed, and the decision sits with Denamico.

The full exclusion list

Not an agent	Why
Loop measurement	Stays workflow-invoked. It is the sole writer of the denominator of every confidence figure, and a double increment corrupts it permanently with nothing recording the true value
The router's cohort assignment	A roll-up from measured outcomes, as above
P&L intake and pre-fill	A member-plane endpoint, as above
Deployment	No agent client on the deploy action. An agent that can push an asset live is what "influence, not control" forbids
The Monitor scoring pass	Deterministic, network-wide, and it writes what every read surface trusts. Now built as the <code>fpi-score-location</code> workflow action driven by W7. The agent plane resolves identity to one Contact and re-derives that person's walk, so a portal-wide pass has no consultant to be
The batch benchmark pass	Same reasoning, and it is the known blocker
Anything franchisee-facing	No agent, ever, at any tier. This one is absolute

Five new refusal codes come with the new tools

Each needs a constants entry and a matching declaration in the component metadata, or the parity test fails: missing play fields, missing evidence, unknown KPI, already-drafted candidate, and not-an-executive, which is the mirror of the existing not-a-consultant code.

5 Architecture and innovation

5.1 Three runtimes, one product

THREE RUNTIMES, ONE PRODUCT

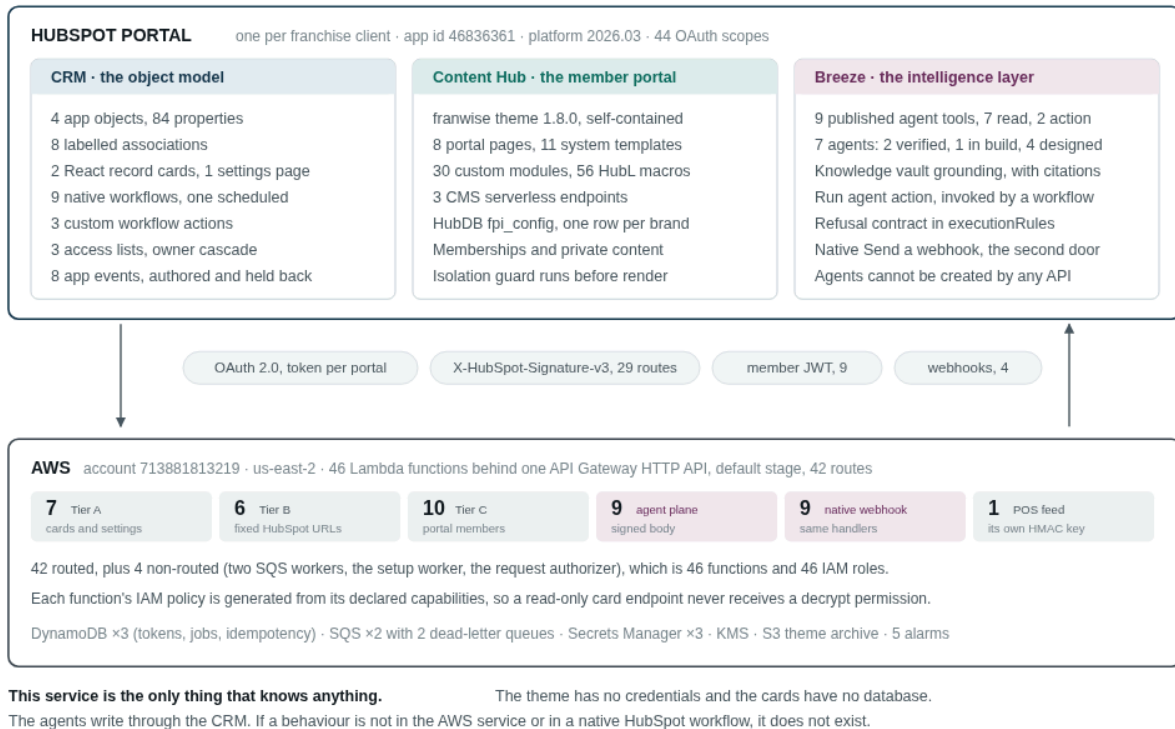


Figure 14. Three runtimes, one product. Every surface a customer touches sits in their own HubSpot portal; everything that knows anything sits in one AWS service behind one API.

5.2 Why the whole backend is in AWS, and why that was not a preference

The technical specification split the backend across three runtimes: app functions inside the HubSpot project, a small AWS service, and CMS serverless functions. **All of it lives in AWS instead**, for two reasons the platform imposed.

- 1. HubSpot excludes OAuth-authenticated apps from project serverless functions on platform 2026.03**, and 2025.2 has none at all. The app-function directory the specification describes was never available to this app. The CRM cards and the settings page call the Lambda equivalents through `hubspot.fetch()` instead.
- 2. HubL provably cannot compute an HMAC.** Its only hash filter is md5, and it cannot make outbound HTTP calls. So the specification's "the page renders a signed context token" is not implementable. A small CMS serverless broker in the theme confirms the visitor is logged in and asks AWS to mint the token, so the signing key never leaves AWS.

THE ONE CONSEQUENCE WORTH INTERNALISING

This service is the only thing that knows anything. The theme has no credentials, the cards have no database, and the agents write through the CRM. If a behaviour is not in the AWS service or in a native HubSpot workflow, it does not exist. That is a deliberately small surface for a system with this many moving parts, and it is what makes the security argument tractable.

5.3 Built once, installed many times

September runs in one portal, but nothing is built in a way that would have to be rebuilt for the second franchise client. This single constraint explains most of the architecture, and it is worth showing the specific engineering it forces rather than asserting it.

The rule forces	Instead of
Runtime schema resolution against the portal	Hardcoded object type ids
An idempotent, additive-only setup runner	Manual provisioning per portal
Per-portal OAuth token custody in DynamoDB	A single shared credential
Configuration in HubDB, one row per brand	Configuration in code or in a redeploy
Globally-unique event occurrence ids composed from the portal id outwards	Ids that would collide between two installed portals and silently drop the second brand's audit trail
A data model shipping as code inside the app	A provisioning script that drifts between installs

Runtime schema resolution: nothing hardcodes a wire name

A constants module names everything **logically**. A schema module asks the portal what it actually has.

```
getPortalSchema(portalId)
  memory (1h) → DynamoDB (1h) → GET /crm/v3/schemas/{fqqn}
```

The matcher tolerates every shape the platform might return: an app-object fully-qualified name, a custom-object one, or a bare name. The prefix deriver reads the property prefix off a property the schema actually reports, rather than trusting a naming convention. **When the granted object name changes, it is a cache invalidation rather than a code change.**

Cache	TTL	Invalidated by
Portal schema (memory and DynamoDB)	1 hour	An explicit invalidation at the end of every setup run
Brand configuration (per warm container)	5 minutes	An explicit invalidation on every config write

Cache	TTL	Invalidated by
Member JWT	15 minutes	Nothing. The browser re-mints a minute before expiry
HubSpot access token	30 min, refreshed 5 min early	The conditional-write lease described below

5.4 The AWS backend

46 Lambda functions behind one API Gateway HTTP API. The stage is the default stage rather than a named one, because a stage segment prefixes every path and the request signature covers the full URI.

The manifest is the source of truth and the shell scripts stay dumb

A single manifest file holds the declarative half: name, route, handler, auth mode, timeout, memory, capabilities, and an optional trigger. **Adding an endpoint is one entry there plus one file in the handlers directory.** The deploy scripts loop over it. No Lambda, role, or route is ever hand-edited.

Each function's IAM policy is generated from its declared capabilities, so a read-only card endpoint never receives a decrypt permission and the OAuth callback does. A capability string the library does not recognise **warns and produces no policy**, so the function deploys with a role that cannot do the thing, which fails loudly at the right time.

Capability	Grants	Held by
secrets:app	Read the app credential secret	45 functions
secrets:signing	Read the signing-key secret	3
tokens:rw	Read and write portal tokens, plus KMS via the service condition	42
idem:rw	Claim and release idempotency keys	15
jobs:rw	Setup job rows	3
sqs:*:send / :consume	Per-queue send or receive, never both	1 each
invoke:setup-worker	Invoke exactly one named function	1
s3:theme-read	Read the theme archive	1

The 46 functions, by tier

- **Tier A, 7 functions.** Replaces the six app functions the platform would not host. Called by the cards and the settings page, all signature-authenticated: location intelligence, assignment detail, config read and write, setup verify, setup run, setup status.
- **Tier B, 6 functions.** Everything HubSpot calls at a fixed URL: install, OAuth callback, the webhook receiver, the deploy workflow action, the access-stamp workflow action, and the measurement writeback.

- **Tier C, 10 functions.** Portal members, all member-JWT, all re-walking before any read or write: token issue, P&L submit, benchmark, play decision, cockpit, location detail, assignment detail, financials, my business, executive dashboard.
- **Agent plane, 9 functions**, plus **native webhook plane, 9 routes** onto the same handlers.
- **Feed, 1 function**, authenticated by its own HMAC key.
- **Non-routed, 4 functions:** two SQS workers, the setup worker invoked asynchronously, and the API Gateway request authorizer.

The webhook receiver is a five-second door

It verifies, enqueues to SQS, and returns. Nothing else. Everything that could be slow happens in a worker with its own concurrency cap and its own dead-letter queue, because HubSpot's webhook timeout is not a suggestion.

Infrastructure, verified against the account

Resource	Detail
HTTP API	One API, default stage, auto-deploy, 42 routes
DynamoDB	Portal tokens (customer-managed KMS key, point-in-time recovery on), jobs, idempotency
SQS	Webhook events and clone jobs, each with a dead-letter queue, batch item failures reported, concurrency capped
Secrets Manager	Three secrets: app credentials, signing keys, developer credentials
S3	The immutable theme archive, keyed by version
IAM	46 roles, one per function, all generated from declared capabilities
CloudWatch	46 log groups, JSON log format, 30-day retention
Alarms	5: errors on the OAuth callback, the webhook receiver, and the deploy action, plus depth on both dead-letter queues

The deploy pipeline, and why the order is load-bearing

One root script is the only supported way to deploy, because four things must happen in one order and getting it wrong produces failures that look like something else entirely.

1. **The HubSpot project uploads first**, because the app objects carry the properties the Lambda code writes. Deploy the service first and every consultant decision fails on a property that does not exist yet.
2. **The schema cache is flushed.** A property added in step one is invisible for up to an hour, and the window looks exactly like step one having failed.

3. **The theme archive goes to S3**, because the function deploy bakes the theme version into all environments and the setup worker fetches that exact key.
4. **Then the service deploys**: bootstrap, build, functions, API, verify.

It refuses to run on a dirty tree, runs the test suite first, and requires the operator to type a confirmation word before the irreversible half. Every individual deploy script is create-or-update, so re-running is the intended way to repair drift rather than something to avoid.

Token custody, and one detail worth showing

HubSpot access tokens live 30 minutes and 46 functions want one. The read path is three layers: module-scope memory in a warm container, which is free; a consistent DynamoDB read, which is single-digit milliseconds; and a refresh guarded by a roughly fifteen-second conditional-write lease.

The lease is the interesting part. The loser's conditional-check failure carries the current item back with it, so if the winner has already written a fresh token the loser uses it immediately **with no extra read**. A returned refresh token is always persisted over the old one, because rotation may or may not be real and being wrong in that direction costs nothing while the reverse loses the install.

6 Governance, security, and compliance

For a product that reads independent operators' financial data and can cause marketing to be sent to their customers, this is not a compliance appendix. It is the part of the design most likely to determine whether a franchise brand can adopt it at all.

6.1 Four gates

- 1. **Consultant review** in the agent inbox. The agent's draft is good enough to show a franchisee.
- 2. **Franchisee opt-in** in the portal. This may go to my customers.
- 3. **Content approval plus subscription enforcement** on every send.
- 4. **The autonomy dial**, which in September is configuration rather than approval. The branch exists and always resolves to gated.

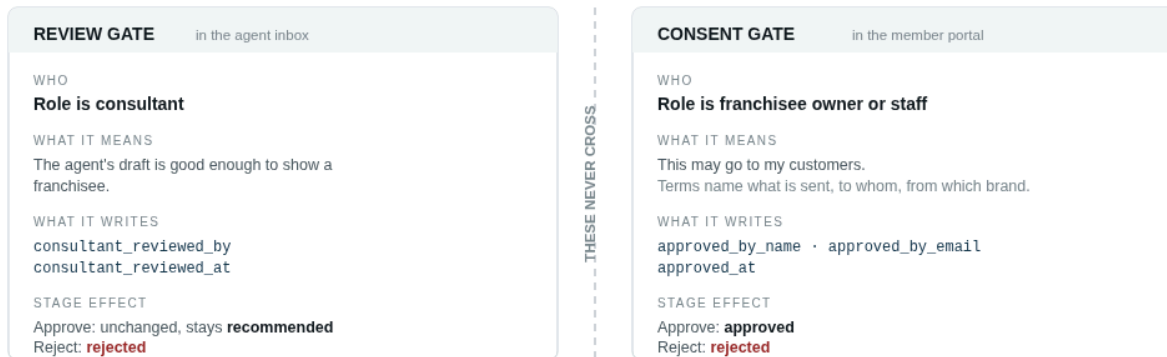
6.2 The two approval gates never cross

A Play Assignment carries two different decisions and they are not interchangeable.

Gate	Who	What it writes	What it means
review	Role is consultant	consultant_reviewed_by, consultant_reviewed_at	The agent's draft is good enough to show a franchisee
consent	Role is franchisee owner or staff	approved_by_name, approved_by_email, approved_at	This may go to my customers

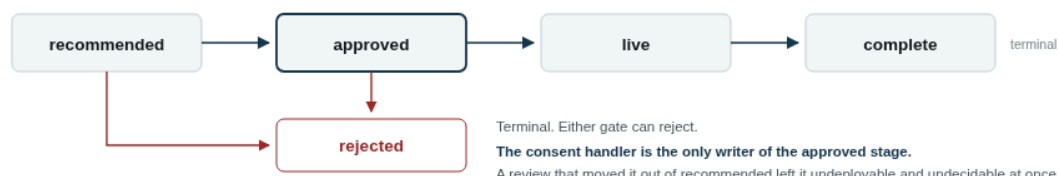
Anything else, whether an executive, an absent role, or an unrecognised one, is **refused rather than defaulted**. There is no safe default here: guessing wrong either fabricates a consent or discards a real one.

TWO GATES ON ONE RECORD, AND THEY NEVER CROSS



Anything else, whether an executive, an absent role, or an unrecognised one, is refused rather than defaulted.
There is no safe default here: guessing wrong either fabricates a consent or discards a real one.
The channel field is written on both paths and records the surface, not the gate. No surface may present a review stamp as an approval.

THE STAGE MACHINE · APP OBJECTS HAVE NO PIPELINE, SO STAGE IS AN ENUMERATION ACTING AS ONE

**The branch is on the role, never on the walk.**

The walk composer tries the franchisee walk first, so a consultant who also owns a location would be classified as a franchisee and stamped as having given customer consent on someone else's behalf. Of the two ways to be wrong, only one of them sends email.

Figure 15. Two decisions on one record. The gates write different properties, mean different things, and only one of them advances the stage.

The branch is on the role, never on the walk

The walk composer tries the franchisee walk first, so a consultant who also happens to own a location would be classified as a franchisee and stamped as having given customer consent on behalf of someone else. Branching on the role read from the contact record fails the other way: the review stamp lands, the deploy action's consent re-check finds no consent, and nothing deploys.

A gate also decides its own stage transition, and getting that wrong was a real defect

Gate	On approve	On reject
review	Stage unchanged. Stays recommended	rejected
consent	approved	rejected

The approval channel field is written on both paths and records the **surface**, not the gate. **No surface may present a review stamp as an approval.** An approval trail, a card, or a projection that falls back to the consultant reviewer for the word "approved" recreates the defect on the way out.

6.3 The isolation guard: one module where a bug is a data breach

The walks module is deliberately small, has no clever paths, and is the only place allowed to decide what a member may see.

Function	What it does
<code>resolveWalk</code>	The walk. Returns the contact, the portal, and the reachable location and company ids
<code>assertLocationInWalk</code>	Throws not-found if the location is not on the walk
<code>assertRecordInWalk</code>	Same posture for a financials row or an assignment, and returns the owning location id on success
<code>financialsForWalk / assignmentsForWalk</code>	Batch association reads keyed off the walk's location ids
<code>resolveExecutiveScope</code>	Not a walk. Executives are network-wide read-only, so their scope comes from the role. A separate function precisely so it cannot be reached by accident from inside the walk resolver

Three rules every caller depends on

- 1. Login proves identity; the walk proves reach.** Every member handler re-derives the walk on **every** request, before any read or write.
- 2. List surfaces build their query set from the walk.** Nothing out of reach is ever fetched and then filtered, because filtering after fetching is how out-of-scope data leaks.
- 3. A denial is indistinguishable from a nonexistent record.** Both throw the same not-found, with the same status and the same body, and never hint that the record exists.

Composition lives **outside** the walks module. Helpers that try one walk and then the other add no reach of their own, which is the point: a change to the walks module triggers the full isolation test plan, and a composition helper should not.

6.4 The member session chain

LOGIN PROVES IDENTITY. THE WALK PROVES REACH.

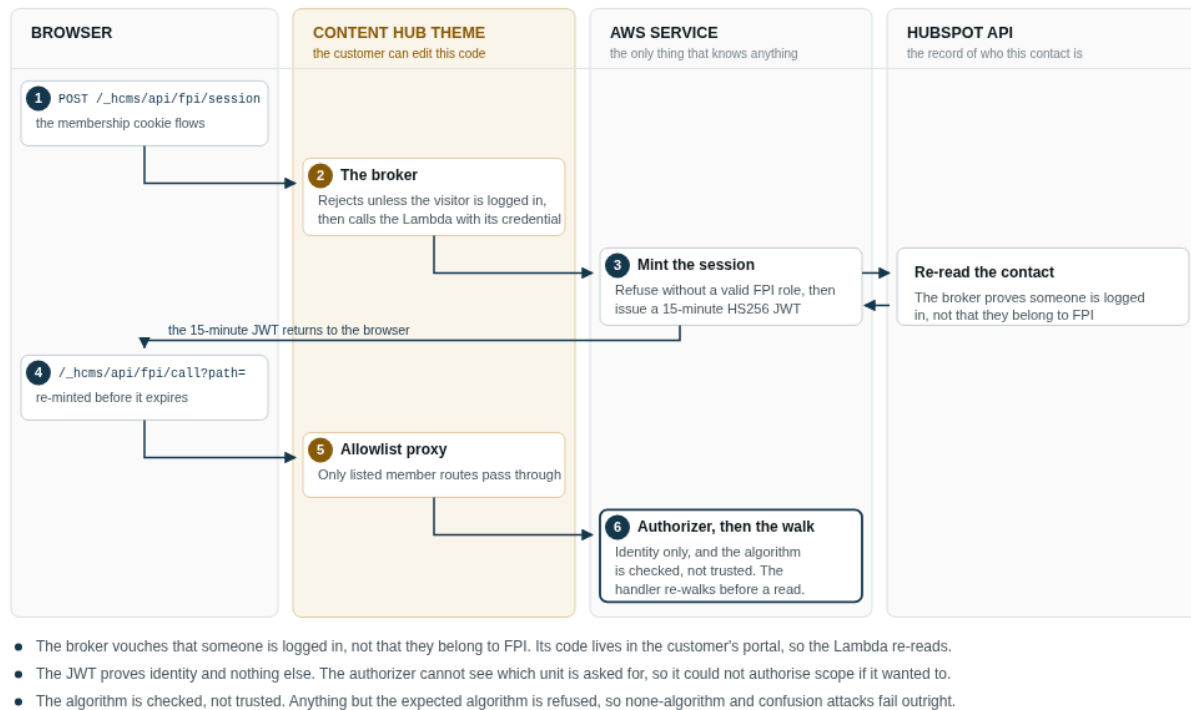


Figure 16. The member session chain. The theme lane runs in the customer's own portal and its code is editable by them, which is why the Lambda re-reads the contact rather than trusting the broker.

Three properties hold this up, and each of them answers an obvious attack:

- **The broker vouches that someone is logged in. It does not vouch that they belong to FPI,** and it runs in the customer's portal where its code is editable by the customer. So the Lambda re-reads the contact from HubSpot and refuses to mint a token without a valid FPI role.
- **The JWT proves identity and nothing else.** The authorizer cannot see which location is being asked for, so it could not authorise scope even if it wanted to.
- **The algorithm is checked, not trusted.** Anything but the expected algorithm is refused, so the none-algorithm and confusion attacks are refused outright.

6.5 Five caller classes, five proofs

Mode	Caller	Proof	Routes
hubspot-signature	HubSpot itself: webhooks, workflow actions, card fetches, agent tools, native webhooks	X-HubSpot-Signature-v3	29

Mode	Caller	Proof	Routes
<code>member-jwt</code>	A logged-in portal member	HS256 JWT via a request authorizer	9
<code>broker</code>	The theme's CMS serverless broker	Shared credential from Secrets Manager	1
<code>feed-hmac</code>	The point-of-sale feed	Its own key, same construction	1
<code>none</code>	Install, OAuth callback, and the workers	Nobody is installed yet, or the trigger is not HTTP	6

Three things that silently break signature verification

- 1. The API stays on the default stage.** A named stage prefixes every path, and the signature covers the full URI including path and query string.
- 2. Never put a redirecting proxy in front of these routes.** A redirect changes the URL and invalidates every signature.
- 3. Verification cannot move into a Lambda authorizer,** because the authorizer payload carries no request body in either format version, and the signature covers the body.

The URI builder is the single most error-prone line in the service, and it was **confirmed empirically** against a captured native invocation: the signature matched only against the URI exactly as that function builds it. A trailing slash, an http scheme, and a bare path all failed. An unsigned request is also rejected **before** Secrets Manager is read, because these routes are on the public internet and will be probed, and loading the secret first would turn every drive-by request into a 500.

6.6 The benchmark anonymity floor

The absolute minimum cohort is 3 and the default is 5, and the effective floor is the larger of the two, **so an administrator cannot configure their way below it**. Below three, a benchmark is a peer's numbers with extra steps. Below the floor, the result carries no numbers at all.

- **No peer's identity or values ever appear in a response.** Peer numbers exist only inside the benchmark computation and never leave it.
- **Only peers who actually reported the same period enter the cohort.** Counting locations that filed nothing would inflate the cohort size, which is precisely the number the floor guards.
- **The two sides of the mirror have different bases**, which is why the cohort is nested rather than flattened. The percentile is a stamped snapshot from whenever the row was last benchmarked; the medians are computed now. A stored snapshot proves only that the cohort cleared the floor **when it was written**, which is why the medians never come from the row.

A GUARDRAIL FIRING ON EMPTY DATA LOOKS IDENTICAL TO A BROKEN PRODUCT

A live finding worth recording: no batch benchmarking pass existed anywhere in the codebase, so no benchmark had ever actually been computed in the demo portal, and every agent briefing returned "too few comparable locations."

That message is the anonymity floor working correctly. It is also what a demo audience reads as failure. **Both statements are true, and the second one is the one that matters on stage.** It is the last blocker before the pitch and it is named in Part 8.5 rather than glossed over, because a system that cannot tell data absence from data failure will diagnose the wrong thing.

6.7 Franchisee-facing responses omit rather than null

Projection	Franchisee audience	Consultant audience
Location	Omits band, score, drift flag, drift reason, drift since	Carries all five
Assignment	Omits the consultant review stamps	Carries both

The fields are omitted, not nulled. An absent key cannot be rendered by a template that forgot the rule, and it is not sitting in the page source for someone to read either. The audience argument is always the walk that actually reached the record, never a request parameter.

6.8 Consent compliance

One brand in this vertical carries a marketing opt-out class action, which is why the consent design is load-bearing rather than boilerplate. Concretely:

- The consent statement names **what** is sent, **to whom**, **from which brand**, and **over what period**. Three dedicated Play Assignment properties were added for exactly this.
- A term that is genuinely absent stays undefined, so the gate renders a visible "not supplied" marker rather than a silently incomplete consent statement.
- The consent component's decline-consequence copy is a **required prop**, because a consent gate with no exit is not consent.
- **Declining is never blocked by the consent checkbox.** Native form validation runs before the submit event, so a required checkbox plus a decline button meant a franchisee **had to consent before they were allowed to refuse**. Fixed, and now asserted by a structural check.
- The setup runner looks the subscription type up by name and **halts with an instruction rather than substituting another one**, because substituting would mail under a consent that was never given.
- Cloned assets are created as drafts and never published, which is enforceable because push-live cannot publish a never-published draft.

Figure 17. The locked consent treatment. The terms sit in the same visual block as the control at every supported width, and declining is never blocked by the consent checkbox.

Secrets live in AWS Secrets Manager, never in the repository, never in HubDB, and never in a log line. The configuration table's public API access stays off: turning it on would publish the install's configuration to the CDN, readable with no authentication. The broker credential is referenced by name in configuration and is never logged, never returned, and never reaches the browser. Git hooks hard-block staged credential files with no override.

7 The design system and the brand posture

The design layer is included in this document because it carries a functional requirement, not an aesthetic one. In a product whose entire premise is that a franchisee will hand over financial data voluntarily, how a number is presented is part of the security model.

7.1 The brand is a chassis, not a badge

The franchisor owns everything a franchisee reads as identity. FPI owns everything read as behaviour. Denamico owns nothing visible on a franchisee-facing page.

The brand posture, stated as a constraint

The franchisor owns the logo, the colour, the programme name, and the typeface. FPI owns the structure, the vocabulary, the density, the way numbers are presented, and a **locked** semantic colour layer that no install may re-skin, because status colour carries meaning that cannot survive theming.

The mitigation that makes the lock safe is simple and absolute: **nothing is communicated by colour alone**. Every state, band, and variance renders its word as well as its colour.

7.2 The three-layer colour model, made literal in code

Layer	Tokens	Themeable
Identity	brand primary, secondary, on-primary, soft, plus hover and active states	Yes , via the theme fields
Chassis	ink, surface, border, link, accent, chart, and panel scales	No
Semantic	the four status ramps, freshness, and consent	Never. Locked.

The accent and chart palette is chassis-owned and deliberately separate from status. The chart green and the system green that means "this location is healthy" are different colours on purpose. Accent colours are legal on reading-surface eyebrows, the benchmark marker, coverage bars, chart fills, and navy panels. They are **illegal on anything carrying a status meaning or readable as a grade**.

Motion tokens are real, running from 80 milliseconds to 240, and drop to zero under a reduced-motion preference. There is no bounce, no spring, no count-up on figures, and **no celebratory animation on submission**, because a P&L is not a form completion.

7.3 Number presentation is a brand commitment

These are enforced in the tool payloads, the projections, and the components, in that order, so that a template which forgot the rule cannot break it.

- A percentile never appears without its cohort size.
- Every score carries its date.
- Aggregates carry their basis.
- Deltas carry direction and comparison, as in "+4.2% versus P6".
- Periods always carry their range, as in "P7 2026 (Jun 30 to Jul 27)", because three fiscal calendars are supported and a bare month name is ambiguous.
- Tabular figures on every number. Currency in whole locations in tables, cents only in the P&L form. Percentages to one decimal.

7.4 Thirty-one components, five of which are non-negotiable

The design system ships 31 component references across core, forms, data, and patterns, with 198 tokens. Five exist because the brand brief makes them non-negotiable, and they are the ones to reach for first:

Component	Why it exists
BenchmarkMirror	The return the franchisee gets. A mirror, not a grade
ConsentBlock	The locked consent treatment. The decline-consequence copy is a required prop
AgentAttribution	Attributed, never anthropomorphised
StatValue / Delta	Numbers that carry their basis
StatusBadge	The locked ramp, and it never renders on a franchisee-facing page

Three items in the system are documented substitutions rather than decisions, and they are flagged as such rather than hardened: the typeface stands in for the franchisor's licensed family and was chosen because it carries real tabular figures, which a 150-row cockpit table needs; the icon set is an open-licensed one, replaceable by swapping a single map; and no franchisor logo exists yet, so the brand lock renders the programme name in plain type.

8 Evidence: what is built, and what is verified

Everything in this section was read from the repository, the AWS account, or the HubSpot build API on 4 August 2026, with the agent-layer work of 5 August incorporated and marked as pending deploy wherever it appears. Where a figure disagrees with an older document, this one is correct.

8.1 Deployed and verified

	State
HubSpot app	Build #15 SUCCESS, 28 components, installed in portal 51781063 since 24 July 2026
AWS service	46 functions, 42 routes, all infrastructure present and matching the manifest
Theme	Built, validated, uploaded. The repository copy is the one fetches back from the portal that renders correctly
Agent tools	9 uploaded, published, endpoints live and routed; reachable through both the agent-tool contract and native webhooks
Breeze agents	2 built and verified end to end against live tool output, a third in build, and four more designed with their instruction blocks written
Secrets	All three populated
Merged since build #15	The <code>fpi-score-location</code> workflow action, W7 · Location rescore, W8 · Executive briefing, and the settings Agents tab. Merged to main, 759 tests passing, typecheck clean, `hs project validate` SUCCESS. Not deployed and not uploaded. The endpoint deploys before the app uploads, or automation preflight blocks in every portal until the action id resolves

8.2 Exercised against a live portal

Between 24 and 28 July, the following ran against a real HubSpot portal rather than against a mock:

- A real OAuth round trip, and per-portal token custody including a live refresh.
- Runtime schema resolution: all four app objects resolve, the property prefix derives correctly, and the stage property round-trips.
- The setup runner, end to end.
- Reads and writes across all four objects, association creation across all nine seeded edges, and **the rate limiter under a burst of roughly 530 writes including a 429 back-off.**
- A validation flag answered: the platform **does** calculate the two calculated properties.
- The isolation walk itself: four consultants' books resolve live and are pairwise disjoint, and the franchisee walk reaches a location through contact, company, and location.
- **The isolation test plan: 24 passed, 0 failed, 0 inconclusive.**

- **The two approval gates: 18 passed, 0 failed, 0 inconclusive**, including the hard case of a consultant who also owns a location, reached by the franchisee walk first and still correctly gated as a review.
- The logged-in portal render, per signed-in member, across all guarded pages.
- A native agent-webhook invocation captured and both of its signatures verified offline.

8.3 Quality gates

Gate	What it is	Result
Backend test suite	759 tests across 37 files (719 at build #15)	All passing
Backend typecheck	Strict, with unchecked index access on	Clean
Theme inspection	52 structural checks over code with comments stripped	52 passed, 0 failed
Theme validation	Compiles every theme file against a real portal	Blocking on failures
Deployed-gateway verify	Eight read-only assertion steps	Counts derived from the manifest
Isolation test plan	Live, read-only	24 passed, 0 failed
Governance gate tests	Live, and they write ; refuses without an explicit confirm flag	18 passed, 0 failed
HubSpot build	The platform's own build	#15 SUCCESS. Post-build work passes <code>hs project validate</code> and is pending upload
Git hooks	Blocks main commits, bulk commits, credential files, build output	Installed per clone

Five test properties worth naming

- **A drift test reads the deployment contract file at test time**, so a divergence between the service constants and what is actually deployed fails in continuous integration. This exists because one constant had already drifted to a property no portal has, silently breaking every stage read and write.
- **A pending-components test asserts both HubSpot builder rules against everything in the app directory**, so re-adding the app events before the platform supports them fails in CI rather than ten minutes into a deploy.
- **The refusal test asserts declaration and emission parity in both directions**, and byte-identity **through the handler rather than through the helper**, with a companion test proving the two answers really did come from two different code paths.

- **One guard test was run against a deliberately broken build** and it failed, alone, as it should. A guard test that has never been watched to fail is a guard test that proves nothing.
- **Both live test scripts report inconclusive rather than pass** for a check they could not evaluate. A test plan that fails open is the same class of defect as the guard it is testing, and the first drafts of both made exactly that mistake.

8.4 Not yet exercised

Item	Note
A real signed webhook from HubSpot	The receiver is deployed and the verifier is proven against captured invocations
Clone-and-adapt	The seeded source asset ids are synthetic strings, so the adapt worker has nothing real to clone
The four newest agent tools	Deployed, routed, and in the build, but nothing has driven one end to end
Whether an execution-rule message reaches the user	All eleven actions declare the error code; answerable in a single invocation that refuses
A browser member session for a seeded contact	There is no membership-registration API and every seeded contact uses a reserved domain that cannot receive registration mail, hence an operator preview path
The network briefing endpoint	Nothing has ever reached it. Testing is blocked on AI credits on the developer account, and it is the one tool the executive agent depends on entirely

8.5 Known-open defects and gaps, stated plainly

This list is here on purpose. A judge evaluating an operations product should be able to see whether the team knows where its own weak points are, and a system this size with nothing open would be a system nobody had looked at closely.

#	Item	Severity and note
1	The walk resolver fails open on an unresolvable association label , widening to the unfiltered contact-to-location set. In the one module where a bug is a data breach	Highest. Latent rather than universal: the trigger is an administrator renaming a label, a partial install, or a labels read that answers without them. The seed writer's preflight refuses to seed such a portal. A branch carries the real fix and it triggers a full isolation re-run
2	The six agent-owned scoring properties have a writer at last, but it is not deployed	Closed in code, open in every portal. The <code>fpi-score-location</code> action and W7 are merged and tested. Until they deploy, cockpit ordering falls back to arbitrary order, all four band tiles read zero, the executive distribution is entirely unbanded, and W5 is a drift-alerting workflow nothing can fire

#	Item	Severity and note
2b	Unknown: what identity a workflow-driven agent run carries into a tool call	Open and untested. The candidates are the enrolled Contact, the seat holder, or nothing at all. Two of those three fail silently , which is the worst possible failure shape inside an agent turn. W8 depends on the answer, and testing is blocked on AI credits on the developer account
3	No batch benchmarking pass exists , so cohort comparison returns below-floor everywhere	The last blocker before the pitch. Revenue Clarity *is* that section. Fix is scoped: audit cohort density, then reuse the existing per-location benchmark writer in a batch pass. The floor is never to be lowered
4	Nothing writes the three consent-term properties. The drafting agent is supposed to	A franchisee's consent block currently renders its terms from three empty properties. Raised rather than touched, because it is a change to an existing tool's permanent input fields
5	The measurement writeback writes notes wholesale while the adapt worker appends its deployment record there	Measuring an assignment while passing a note destroys the deployment history
6	The play projection does not split by audience the way the assignment projection does	Its evidence field is free text a language model composed and can still name a peer
7	A denial flag is read in the layout but no longer set there	Left exactly as the portal runs it, because the two readings disagree and neither is safe to guess. Resolvable by reading a denied page's title as a logged-in member
8	The best-fit-cohort roll-up is not built	A promoted candidate stays general-purpose forever unless someone sets it by hand
9	W5 was created but could not be enabled (a 400 from the platform)	Worth its own investigation
10	The custom-domain script is written and not run	The explicit gate before the first non-demo portal. See Part 3.10

9 Field report: where the platform and its documentation disagree

This section is written as feedback to HubSpot rather than as a complaint. Each entry was found at build time, each is handled in code, and none of them is to be "fixed" back toward the specification. For a partner evaluating whether to build a vertical app on this platform, this is arguably the most useful thing in the document, because it is the cost that does not appear in any estimate.

PRECEDENCE, AS THE REPOSITORY STATES IT

The specifications outrank the repository's prose. **The build-time records outrank the specifications where they disagree.** And where a document and the code disagree, **the code is right.**

9.1 App and platform level

Behaviour	Consequence
Private distribution plus OAuth cannot carry app objects	The app ships marketplace-distributed and unlisted
OAuth apps cannot ship project serverless functions on 2026.03	The six specified app functions moved to Lambda; cards and settings call them via <code>hubspot.fetch()</code>
App-object properties are auto-prefixed with the app id	Contradicts the specification; handled by runtime prefix resolution
Calculated-property formula grammar has no closing keyword	The documented syntax is rejected by the parser
The formula rewriter substring-corrupts property references	One property had to become a plain number written by the submit path. The two that do work are confirmed platform-calculated
Webhook <code>objectType</code> accepts app-object uids, but only after the objects exist in the portal	Webhooks deploy after the objects' first deploy
Webhook target host must resolve in DNS at deploy time	Provision DNS before swapping the domain
<code>hs_pipeline_stage</code> exists on every app object regardless of pipelines	Its presence proves nothing; the app's own stage enumeration wins
GET <code>/crm/v3/schemas</code> (the list route) needs a scope this app does not request	Objects resolve one at a time by fully-qualified name
The app-object config schema admits no user-reference fields	User-reference properties cannot be declared in hsmeta on 2026.03. Created via the CRM properties API instead

Behaviour	Consequence
A required agent-tool input field cannot be removed or updated once uploaded	All eleven actions ship every input as not-required and enforce required-ness in the Lambda
A workflow action's locale labels require input-field label and description maps even when the action declares no inputs	One tool ships both as empty objects
<code>supportedValueTypes</code> is typed as an unbounded array; the builder demands exactly one	A permanent choice on all nine tools
<code>objectHandlingConfig</code> is defined in the app-event schema and refused by the builder	All eight app events held out of the build
An unrecognised enumeration value fails the whole object create rather than being ignored	The candidate-drafting tool validates and drops an unrecognised category rather than refusing the write
No API creates a subscription type; no membership access-groups API; no content-approvals API	The runner halts with instructions; access lists are CRM lists; cloned assets stay drafts

9.2 Associations and CRM writes

Behaviour	Consequence
A v4 association write validates the category and type id as a pair , and every FPI label is integrator-defined	The associate helper hardcoded the wrong category, so no labelled association in the service could be written at all . The error names a type id that plainly exists, so it reads as a bad label
The labels endpoint returns only the string naming the to side	A lookup in the wrong direction returns undefined, and the batch reader treats undefined as no filter . The turnover cascade had been reading every association each Location has
<code>domain</code> is HubSpot's company deduplication key but is not a unique-value property	Batch upsert refuses it as an id property and the equivalent read route is unrouted. Companies are matched by paging the list route, never by search

9.3 HubDB, CMS, and pages

Behaviour	Consequence
A HubDB draft table PATCH deletes any column omitted from the request	Column additions must read the draft, carry every existing column forward with its id, append, then publish
The read and write column schemas are different : the id is a string on read and an integer on write, and read carries six fields write does not declare	Echoing the read shape back produced a HubSpot 500

Behaviour	Consequence
HubDB soft-deletes columns and keeps returning them, and the column count includes deleted ones	A tombstoned column counted as present, was never re-created, and the row write then rejected it by name
HubDB v3 has no per-row publish	Any config write publishes the whole draft table, so it happens only on a genuine period rollover
A theme's <code>fields.json</code> has no free-text field type , and an unsupported type is not rejected , it just renders no input	Sixteen fields shipped pinned to their defaults. All moved to HubDB columns
<code>publicAccessRules</code> is write-only ; the values are intentionally omitted from GET responses	Verification cannot confirm a page is private. Never re-PATCH the field to "ensure" it, because array fields replace rather than merge. Fail closed
PUBLISHED is not a documented writable state on 2026.03, and the legacy publish endpoint does not exist	Create without a state, escalate to schedule with a now timestamp, then push live. A page that stays a draft is reported blocked, never silently accepted
<code>cms.source_code.*</code> scopes do not exist on 2026.03	<code>content</code> is the only scope that reaches the Source Code API. An unknown scope passes local validation and fails server-side at upload

9.4 HubL and Jinjava: the most expensive category

Behaviour	Consequence
HubL cannot compute an HMAC and cannot make outbound calls	The CMS serverless broker replaces the specification's signed context token
<code>`crm_object()`</code> returns an enumeration's LABEL, not its internal value	The role came back capitalised, never lowercase. The REST API returns the value, so every service-side check was right while every template-side one was silently false. It denied every guarded page to every member of every portal , and a false role gate is invisible because the guard's one exit is a denial with no reason code
A block nested inside a conditional in a layout does not gate its child's content	Blocks resolve at composition, not at evaluation
A page's constants are not readable from the layout's own top-level statements	The guard read every page's kind and required role as unset, and the default filter turned that into a silent wrong answer instead of an error
A child block whose name no parent declares is silently dropped	No error at compile, upload, or validate time. The denial surface itself rendered full chrome around an empty canvas
A set inside a conditional does not escape the conditional	The working idiom is a one-element list declared outside and mutated
<code>`include`</code> rebases relative path resolution ; <code>`import`</code> and <code>`from`</code> do not	No relative form was correct for files imported from two places, so nested imports are absolute and must stay so.

Behaviour	Consequence
	A missing template is a warning: the macro silently renders nothing and the page still returns 200
A comma inside a quoted macro-signature default breaks the whole file	The argument list is split before string literals are parsed. It is a whole-file failure: every macro in the file becomes unresolvable, and so does every consumer
CSS comments do not nest and HubL does not strip them	The stylesheet is one include chain, so a stray comment terminator closes a comment early and swallows the next rule. This cost the portal all 198 of its design tokens
The membership list collection is not the documented type, and carries legacy ids	Two independent reasons the list-derived half of the role union was dead
App-object associations are absent from the CMS GraphQL schema while present as top-level collections	The traversal is undefined under every spelling while the control association is accepted, so the walk cannot be a GraphQL traversal. Five queries retired
<code>crm_objects()</code> accepts the upper-snake fully-qualified name or the bare name, never lower snake	And it enforces the private-page rule at compile time

9.5 Automation

Behaviour	Consequence
The native Edit-records action rejects the value shape in its own documented example	The documented shape answers a 500; it needs the grammar from a different action. Confirmed by bisect
The guide's prose writes a flow read path that does not exist	An importer coded from the prose 404s on every read-back and ships a portal full of workflows that were created and never enabled
A flow can be created before its CRM object exists , and the update that switches it on then fails	A creation-status field is the thing that settles, and the runner polls it
The beta action-type namespace answers HTML 404 while the stable flows route answers a JSON scope error on the same account	A scope-gated route answers JSON; a router miss answers HTML. That distinction is the finding. The gate therefore enforces only what the CRM schema can prove, because a gate that read "I could not ask" as "the answer is no" would install nothing, anywhere
An app's action type id belongs to the app , not the portal	Every portal-reachable discovery surface dead-ended, so resolution moved to deploy time using a developer credential
Unpublished custom actions are only visible in the developer portal	The deploy action must be published or the lifecycle workflow fails to install in every install. Close to a one-way door

Behaviour	Consequence
The Run agent action appears in no published action-type table. It is 0-218870689, version 4, taking a single agent_instance_id field	Learned only by building a flow by hand in the test portal and reading it back. The catalogue endpoint that would list it answers HTML 404 on this portal, so there was nothing to query
`enrollmentSchedule` is part of the stable v4 flow representation , not only of the beta spec where it is documented	This corrects an earlier finding of our own that scheduling was beta-gated and therefore unreachable. A scheduled flow is list-based criteria plus a sibling schedule block, and it works today
An agent instance id is per-portal	And stays per-portal even if agents are one day distributed centrally, because the instance is created inside the account. So an app cannot ship knowing it

9.6 What we would ask HubSpot for

Stated as requests rather than grievances, in rough order of how much they would change what a partner can build.

- 1. A programmatic path to create and configure agents, and to bind tools to them.** This is the single largest per-install cost in the product, at roughly 10 to 15 manual interface selections per customer portal, and it is the one thing that cannot be engineered away under a "built once, installed many times" constraint.
- 2. App objects as agent inputs.** The input picker offers the standard objects only, so the record type the entire product is about cannot be a typed input.
- 3. App objects in the CMS GraphQL association schema.** They are present as top-level collections and absent from traversals, which retired five queries and pushed the walk into HubL.
- 4. A published contract for workflow action fields.** A definition wired to a field name the action does not recognise is accepted, runs, and does nothing, permanently and silently.
- 5. Support for `objectHandlingConfig` in the app-event builder**, or its removal from the advertised schema. As it stands, validation passes and upload fails.
- 6. More than one supported value type on an agent tool input**, or a schema that says only one is allowed. The current mismatch forces a permanent choice on incomplete information.
- 7. A readable `publicAccessRules`.** A write-only gating field means an installer cannot verify that a page is actually private.
- 8. A free-text field type in theme `fields.json`**, or rejection of unsupported types. Silently rendering no input for a declared field cost sixteen fields pinned to their defaults before anyone noticed.

10 Scale, impact, and the road to September

10.1 Why this scales, structurally

Franchising is a large and structurally repetitive market. There are on the order of 830,000 franchise establishments in the Locationed States alone, organised into networks that share the exact same shape: a franchisor, a field consulting function carrying books of 30 to 150 locations, and independent operators who cannot be instructed.

That repetition is the point. The problem FPI solves is not specific to one brand or one vertical, and neither is the solution. What varies between two franchise brands is the chart of accounts, the fiscal calendar, the word they use for a location, the primary KPI, and the brand identity. **Every one of those is a configuration row, not a code change.**

What varies per brand	Where it lives
The word for a location, singular and plural	A HubDB configuration column
The word for a period	A HubDB configuration column
Currency, locale, timezone	HubDB configuration columns
Seven chart-of-accounts labels	HubDB configuration columns. Renaming a label renames what the member reads, never the underlying property
Fiscal calendar	One of three supported shapes: calendar month, 13 four-week periods, or 13 periods with an extended final period
Primary KPI, benchmark floor, variance tolerance	HubDB configuration columns, floored so an administrator cannot configure below the anonymity minimum
Logo, three colours, typeface	The theme identity fields
Everything else	Ships with the app

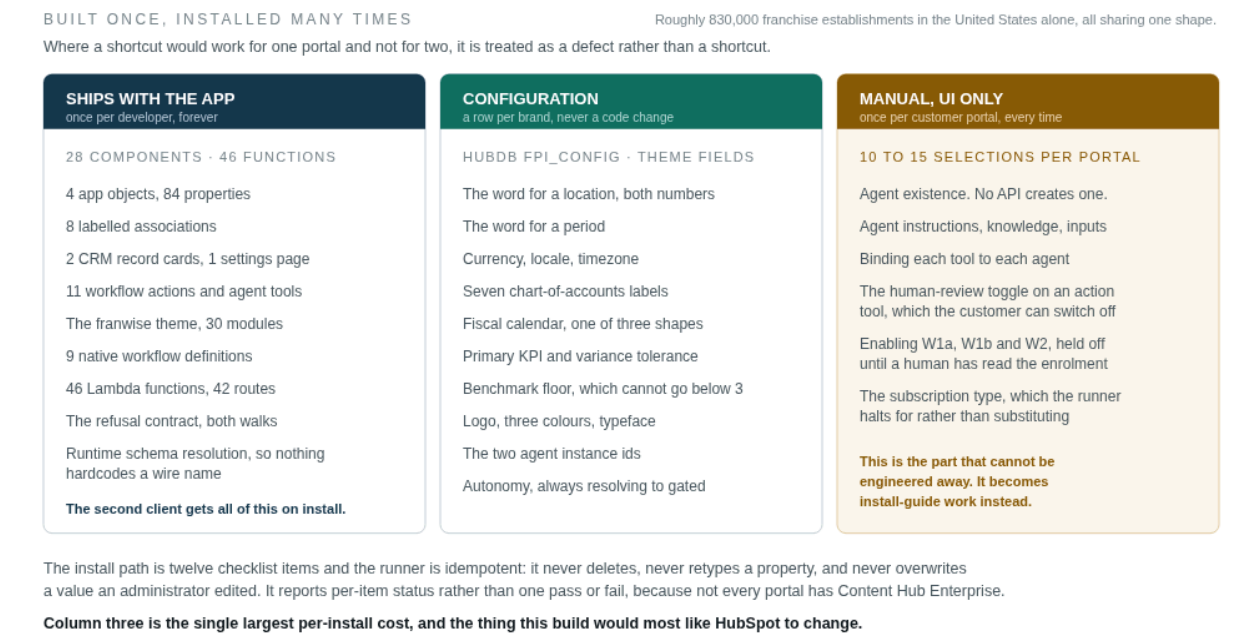


Figure 18. What ships, what configures, and what stays manual. The third column is the per-install cost the governing rule cannot remove.

The install path is twelve checklist items, and the runner is idempotent

A franchise client is allowlisted, their administrator opens the install URL and grants scopes, and the four objects with their eight associations are present immediately because they shipped with the app. Then the settings page's setup panel creates what is missing, in dependency order: standard properties, the properties hsmeta cannot declare, association verification, the configuration table, domain detection, the subscription type, three access lists, the theme, the eight pages, an automation preflight, the nine workflows, and a final verify.

The runner never deletes, never retypes a property, and never overwrites a configuration value an administrator edited. It reports per-item status rather than one pass or fail, because a portal without Content Hub Enterprise fails the access-list step while everything else succeeds, and the panel has to say exactly that.

10.2 Customer impact

The impact case rests on three specific changes to how a franchise organisation operates, each of which is measurable.

Change	Today	With FPI
When a consultant learns a location is drifting	When the lagging metric arrives, typically a full reporting period late	The period it happens. Every location is rescored the moment its financials land, and the drift flag fires a workflow

Change	Today	With FPI
What a consultant does about it	Starts from a blank page and their own recollection of what has worked	Starts from a drafted intervention carrying its own track record across the network
What a franchisee gets for their financial data	A compliance obligation and no return	A benchmark against an anonymised cohort and a shortlist of interventions that moved that metric for locations like theirs

The measurement writeback is what turns this from an assertion into a number. Every deployed play records its baseline, its result, and the delta, against the specific KPI it targeted. After a few laps, a franchisor can answer a question they currently cannot: **did the programme cause anything.**

10.3 The road to September, and past it

Before the pitch

- 1. Batch benchmarking.** The single last blocker. The per-location benchmark writer already exists and is correct; what is missing is a pass that runs it across a cohort. Revenue Clarity is that section of every briefing.
- 2. The fail-open walk test,** which takes two minutes and is the one open item that is a data breach rather than a defect.
- 3. Agent 3, Draft Recommendation,** plus confirmation that the human-review toggle on the action tool stays on.
- 4. Deploying the scoring action and W7,** which is now written and merged rather than scoped. Until it deploys, six properties stay empty in every portal and W5 has nothing able to fire it. The endpoint must deploy before the app uploads.
- 5. The custom domain,** which is the gate before any non-demo portal.
- 6. Agents 4 through 7,** whose tools are already deployed and published. This is configuration work rather than platform work, and it is the part HubSpot exposes no API for.

After it

The contest is a deadline rather than the goal. What survives the event is a productised offer: an installable HubSpot app that a franchise brand buys, that Denamico implements, and that gets better for every brand as each one runs laps. The app events restore in two lines the day the builder supports the field. The marketplace listing is a form, the tool ids, and a demo video. The second franchise client installs the same app.

Appendix A Numbers at a glance

Verified 4 August 2026 against the repository, AWS account 713881813219, and HubSpot build #15. Figures marked pending reflect work merged to main on 5 August that has not yet been deployed or uploaded.

Metric	Value
HubSpot app id	46836361
Developer platform version	2026.03
Developer test portal	51781063 (fpi-demo)
HubSpot project components in the build	28
Components held back (app events)	8
App objects	4
App-object properties	84 (16 / 25 / 13 / 30)
App-object associations	8, all integrator-defined
Calculated properties	2
Unique-value properties	2
OAuth scopes	44
Webhook subscriptions	4
Custom workflow actions	3 (2 uploaded, 1 merged and pending)
Agent tools	9 (7 read, 2 action)
Breeze agents built and verified	2 built and verified, 1 in build, 4 designed
Agent invocation surfaces	4 (agent inbox, agent builder, workflow, record). No chat surface exists
Refusal codes	16 agent-plane, 7 deploy
Native workflows	9 (7 uploaded, 2 merged and pending), one schedule-enrolled
Setup checklist items	12
AWS Lambda functions	46
API routes	42
Authorization modes	5
Authorization planes	4

Metric	Value
DynamoDB tables / SQS queues incl. DLQs / secrets / alarms	3 / 4 / 3 / 5
Backend TypeScript	~29,650 lines
Backend tests	759, all passing (719 as at build #15)
Theme version	franwise 1.8.0
Portal page templates / system templates	8 / 11
Custom modules	30
HubL macros	56 across 6 files
CMS serverless endpoints	3
Theme HubL, JS and CSS	~14,950 lines
Theme inspection checks	52 passed, 0 failed
Live isolation tests	24 passed, 0 failed, 0 inconclusive
Live governance gate tests	18 passed, 0 failed, 0 inconclusive
Design system components / tokens	31 / 198
Fiscal calendars supported	3
Play Assignment lifecycle stages	5
FPI roles	4
Benchmark anonymity floor	3 absolute / 5 default

Appendix B Glossary

Term	Meaning
Location	A franchise location, tracked for performance and coaching. One of four app objects
Financial Period	One row per location per period, carrying seven P&L line items plus a benchmark snapshot
Play	A reusable intervention that has worked somewhere in the network
Play Assignment	One attempt to apply one Play to one Location. The spine of the loop
The walk	The association traversal that determines what a given member may reach. There are exactly two
Consultant	A franchise business consultant carrying a book of 30 to 150 locations. The system's hero
Franchisee	An independent operator. Holds no HubSpot seat; a contact with a membership password
Band	A location's trajectory classification: leading, steady, drifting, at risk. A consultant coaching signal, never franchisee-facing
Drift	Measured deterioration in a location's per-period financials and in its standing against its cohort. Deliberately not the original brief's "a leading indicator moves before the lagging metric does", which turned out not to be computable
The anonymity floor	The minimum cohort size below which a benchmark returns no numbers at all
Review gate	The consultant's approval that a draft is good enough to show a franchisee
Consent gate	The franchisee's approval that something may go to their customers
Agent-owned property	A property no external caller may ever write, whatever it claims to be
Built once, installed many times	The governing architectural rule. A shortcut that works for one portal and not for two is a defect