300 Franchising, a maior do mundo e nossa missão.

LAI Factory OS + CRM HubSpot 1:1 — Plano Operacional Diamante

Orquestrador + Ferramentas + Gates para migração sem fricção (uso interno exclusivo)

Data: 2026-02-20 (UTC)
trace_id: LAI-DOC-20260220T010056Z
Uso: interno exclusivo (não comercial).

Este documento consolida o plano operacional para: (1) compatibilidade 1:1 com HubSpot na fase 1, (2) orquestração automática Pré-Fábrica + Fábrica + Gates, e (3) geração de packs e repositórios para o ecossistema LAI.

Premissa: fase 1 não melhora nada. Ela copia comportamento para eliminar fricção. A melhora vem depois do corte (fase 2).

Objetivo e definição de 1:1

1:1 aqui significa paridade no nível do usuário final: mesmos fluxos, cliques, telas, validações aparentes, atalhos, navegação, erros e performance percebida. Isso é engenharia de compatibilidade, não copiar tela.

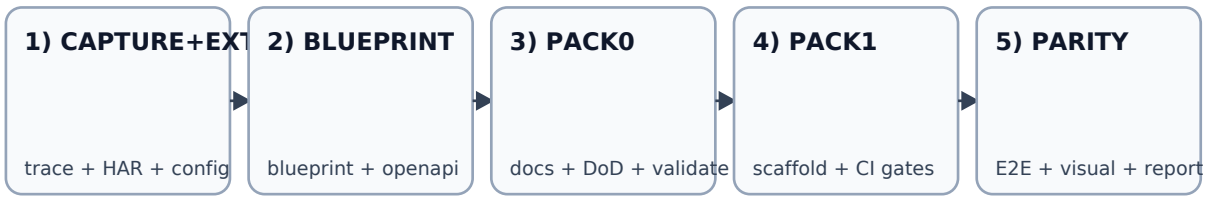
Critérios de paridade (fail-closed)

• Fluxo e estados: o usuário chega no mesmo resultado (inclusive em erros).
• Microinteração: foco, tab-order, atalhos, habilitar/desabilitar, confirm dialogs e toasts.
• Rede: semântica de erros (4xx/5xx), paginação, filtros, rate limit e retries.
• Dados: entidades e associações críticas (contatos, empresas, deals, tasks, activities), ordenação e busca.
• Performance percebida: click-to-render e p95 dentro do budget definido no Pack0.

Caminho robusto: capturar uso real + tráfego + configuração, materializar contratos e provar paridade com testes automáticos.

Golden Path (execução automática em 5 botões)

O Orquestrador expõe 5 botões. Cada botão cria jobs, gera artifacts e dispara gates. Se qualquer gate falhar, o pipeline para (fail-closed).

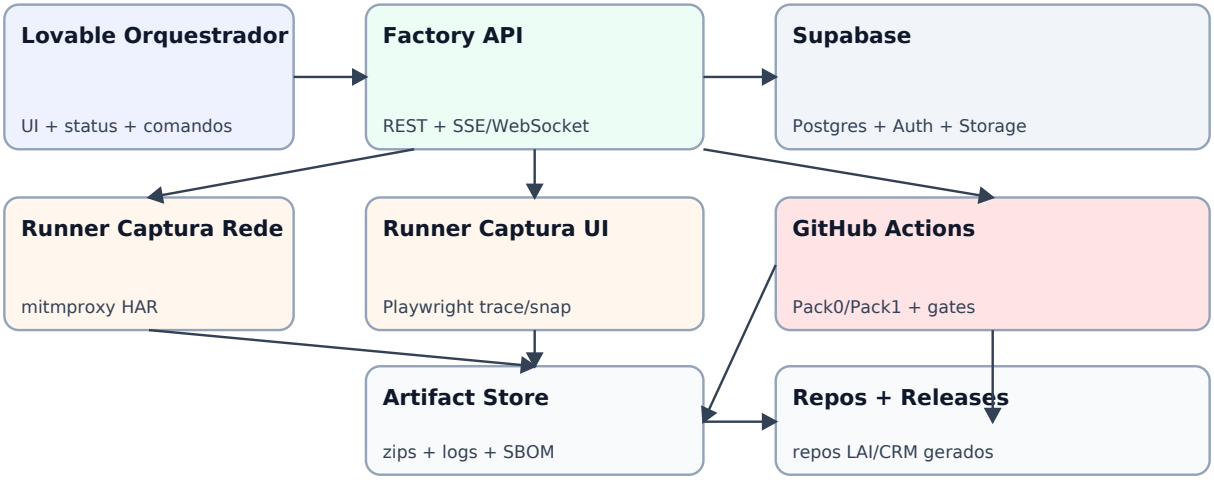


Artifacts obrigatórios por trace_id

- Biblioteca de fluxos: trace.zip + screenshots + scripts reexecutáveis.
- Biblioteca de tráfego: HAR por fluxo (sanitizado por padrão).
- Config extraída (API/exports): pipelines, properties, stages, permissões, views.
- Contrato versionado: openapi.yaml + JSON Schemas + lint/diff.
- Pack0: docs + contratos + DoD + budgets + matriz de retenção.
- Pack1: front/back scaffold + mocks + CI gates + testes.
- Parity Gate: E2E + visual diff + relatório de divergências.

Arquitetura (Control Plane + Providers)

Lovable é o cockpit (UI). A Factory API controla jobs. Supabase guarda estado e artifacts. Runners executam captura pesada em ambiente isolado. GitHub Actions executa Pack0/Pack1/merge/gates.



State machine de run (resumo)

- run.created -> capture.running -> capture.done
- extract_config.running -> extract_config.done
- blueprint.building -> blueprint.ready
- pack0.generating -> pack0.validated (validate-pack0 + run_report)
- pack1.generating -> pack1.ci_passed
- parity_gate.running -> parity_gate.passed|failed

Modelo de dados (DB) — mínimo operacional

Banco recomendado: Postgres (Supabase). Regra: tudo é job + trace_id + artifacts.

Tabela	Função
projects	Projeto interno e sistema fonte (HubSpot).
modules	Módulos gerados (crm, connect, meetcore, etc.).
jobs	Execuções: capture, extract, blueprint, pack0, pack1, parity.
job_inputs	Payload JSON por job (flows, allowlist, configs).
artifacts	Metadados + ponte para binários no Storage.
runs_github	Job -> workflow run no GitHub Actions.
audit_events	Log append-only (evento operacional + trace_id).
hubspot_tokens	Tokens criptografados (se usar API oficial).

API do Orquestrador (endpoints mínimos)

Base: /api/v1. O Lovable só chama isso.

Projetos e módulos

```
POST /projects
POST /projects/{id}/modules
GET /projects/{id}
```

Jobs

```
POST /jobs (type, trace_id, payload_json)
GET /jobs/{id}
GET /jobs?project_id=&module_id=&state=
POST /jobs/{id}/cancel
```

Artifacts

```
POST /artifacts/presign-upload
POST /artifacts/commit
GET /jobs/{id}/artifacts
GET /artifacts/{id}/download-url
```

Execução

```
POST /execute/capture-ui
POST /execute/capture-net
POST /execute/extract-config
POST /execute/openapi-build
POST /execute/factory-pack (pack0|pack1)
POST /execute/parity-gate
POST /execute/publish-repo
```

BUILD BLUEPRINT — o compilador que faltava

Este estágio transforma CAPTURE+CONFIG em blueprint.json (modelo do CRM), openapi.yaml (contrato) e paridade_matrix.json (o que provar).

Entradas

• UI traces (trace.zip) + screenshots baseline.
• HAR por fluxo (allowlist + sanitização PII).
• Config por API/exports: pipelines, properties, stages, permissões, views.

Saídas

• blueprint.json: módulos/telas/entidades/RBAC/workflows.
• openapi.yaml + schemas: contrato versionado.
• paridade_matrix.json: catálogo de fluxos + critérios de aceite.

Exemplo mínimo — blueprint.json

```
{
 "blueprint_id": "crm-hubspot-compat",
 "phase": "phase1_compat",
 "modules": [
 {
 "key": "crm",
 "screens": [
 "dashboard",
 "pipeline",
 "deal",
 "contact",
 "company",
 "tasks",
 "activity_log",
 "settings"
 ]
 },
 {
 "key": "lai-connect",
 "screens": [
 "inbox",
 "templates",
 "routing",
 "webhooks"
 ]
 }
 ],
 "entities": [
 "contact",
 "company",
 "deal",
 "task",
 "activity",
 "user",
 "team"
 ],
 "rbac_roles": [
 "admin",
 "manager",
 "rep",
 "ops"
 ],
 "workflows": [
 "create_deal",
 "move_stage",
 "log_activity",
 "create_task",
```

```
 "search_filter"
 ]
}
```

Exemplo mínimo — paridade_matrix.json

```
{
 "trace_id": "HS-CAPTURE-001",
 "flows": [
 {
 "id": "FLOW-001",
 "name": "Criar deal",
 "passes_if": [
 "deal_created",
 "toast_ok"
 ],
 "visual_baseline": "shots/FLOW-001/*.png"
 },
 {
 "id": "FLOW-002",
 "name": "Mover stage",
 "passes_if": [
 "stage_changed",
 "history_logged"
 ],
 "visual_baseline": "shots/FLOW-002/*.png"
 }
 ],
 "gates": {
 "e2e": "required",
 "visual": "required",
 "openapi_lint": "required",
 "security": "required"
 }
}
```

PARITY GATE — especificação (E2E + Visual + Contrato)

Parity Gate é o alarme. Ele prova que a fase 1 está 1:1 e bloqueia promoção se divergir.

Gates obrigatórios

- Sanitização de PII (HAR/traces) antes de persistir.
- Contract Gate: Spectral lint + openapi-diff.
- API Gate: Schemathesis contra mocks e backend.
- UI E2E Gate: Playwright + budgets (p95).
- Visual Gate: Playwright snapshots + Applitools (páginas) + Chromatic (componentes).
- Security Gate: secret scanning + SAST/SCA básico.

Ferramentas — Stack Diamante (com custo e custos ocultos)

Critério: perfeição cirúrgica e velocidade primeiro. Custo vem depois. Os valores abaixo são os publicados (mudam). O custo real é dominado por assentos + volume + retenção.

Camada 0 — Control Plane

Lovable (UI do Orquestrador)

O que faz: Cockpit (Runs/Artifacts/Settings/Blueprint Studio) e provider assistido opcional.
Por que é a melhor aqui: Você ganha cockpit rápido e mantém export/CI como soberania.
Custo real: Por créditos mensais/anuais; top-ups por pacote de créditos.
Custos ocultos: Crédito explode com geração pesada e prompts longos; mitigar com templates + MCP + WIP limit.
Fonte: https://docs.lovable.dev/introduction/plans-and-credits

Supabase (Postgres/Auth/Storage/Edge)

O que faz: Backend do Orquestrador e do CRM/LAI (fase 1).
Por que é a melhor aqui: Entrega Postgres real + RLS + Storage + Edge sem DevOps pesado.
Custo real: Plano + compute + overages (egress/MAU/storage/logs) por projeto.
Custos ocultos: Multiplica por projetos, egress, MAU e retenção; precisa spend caps e política de logs.
Fonte: https://supabase.com/docs/guides/platform/billing-on-supabase

Temporal Cloud (workflow engine)

O que faz: State machine confiável com retries/timeouts/idempotência.
Por que é a melhor aqui: Tira humano do loop e impede 'run quebrado' virar caos.
Custo real: Base mensal + consumo por actions (publicado).
Custos ocultos: Actions crescem com granularidade e runs; mitigar com batching e checkpoints por etapa.
Fonte: https://temporal.io/get-cloud/aws-marketplace

Upstash Redis + QStash

O que faz: Fila leve/cache e delivery confiável de eventos/cron.
Por que é a melhor aqui: Infra pronta sem operar Redis; encaixa no Orquestrador.
Custo real: PAYG por comandos/mensagens + storage/bandwidth; Prod Pack adiciona custo fixo.
Custos ocultos: Prod Pack e fan-out são os multiplicadores; retries mal configurados explodem.
Fonte: https://upstash.com/docs/redis/overall/pricing

GitHub + GitHub Actions

O que faz: Repos, PRs, CI gates, releases e artifacts.

Por que é a melhor aqui: Sua Fábrica já é pack-first no GitHub; isso vira a linha de produção.
Custo real: Minutos de Actions + storage de artifacts/caches; macOS é caro.
Custos ocultos: Explode com cross-browser/visual tests e retenção de artifacts (vídeos/traces).
Fonte: https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions

GitHub Advanced Security

O que faz: Secret scanning/push protection + CodeQL/dependency review.
Por que é a melhor aqui: Reduz erro humano e bloqueia vazamento de segredos.
Custo real: Preço por active committer (janela móvel).
Custos ocultos: Multiplica quando muitos devs committam; exige controle de repos e permissões.
Fonte: https://github.com/security/plans

1Password Business (Secrets)

O que faz: Gestão de segredos com vaults e auditoria.
Por que é a melhor aqui: Mais rápido para adoção interna do que Vault/infra.
Custo real: Preço por usuário (plano business).
Custos ocultos: Escala com seats; ainda exige rotação e política.
Fonte: https://1password.com/pricing

Camada 1 — CAPTURE 1:1

Playwright

O que faz: E2E + trace.zip + screenshots + snapshots.
Por que é a melhor aqui: Núcleo do Parity Gate; reexecutável e determinístico.
Custo real: $0 (open-source).
Custos ocultos: Custo real é CI/compute e manutenção de locators; mitigar com role/text/testId.
Fonte: https://playwright.dev/

mitmproxy

O que faz: Proxy TLS para capturar requests/responses e gerar HAR.
Por que é a melhor aqui: Automatiza biblioteca de tráfego filtrada por allowlist (só HubSpot).
Custo real: $0 (open-source).
Custos ocultos: Certificados + PII; precisa ambiente isolado e sanitização por padrão.
Fonte: https://docs.mitmproxy.org/stable/

Fiddler Everywhere

O que faz: Proxy HTTP(S) com UI para debug rápido.
Por que é a melhor aqui: Acelera investigação quando o HubSpot mudar.
Custo real: Preço por licença (plano pro/enterprise).
Custos ocultos: Não instale em todo PC; use só em máquina de captura.
Fonte: https://www.telerik.com/fiddler/fiddler-everywhere

Flow Recorder

O que faz: Documentação/jornadas/replay no browser.
Por que é a melhor aqui: Ótimo para catálogo de fluxos e treinamento interno.
Custo real: Preço mensal por plano (Startup/Pro/Business).
Custos ocultos: Pode virar redundante se Playwright cobrir tudo; storage/retention cresce.
Fonte: https://flowrecorder.com/pricing

Sauce Labs / BrowserStack (cross-browser farm)

O que faz: Executa automação em múltiplos browsers/OS.
Por que é a melhor aqui: Reduz risco de divergência em browsers do time.
Custo real: Preço por paralelos/planos.
Custos ocultos: Multiplicador: paralelos x browsers x larguras x frequência.
Fonte: https://saucelabs.com/pricing

Camada 2 — EXTRACT CONFIG

HubSpot Private App API + Exports

O que faz: Extrai pipelines, properties, stages, permissões e dados.
Por que é a melhor aqui: Sem isso, UI igual vira placebo (regras diferentes).
Custo real: $0 licença; limitado por rate limits do plano HubSpot.
Custos ocultos: Rate limit/paginação e PII exigem retries, logs e minimização.
Fonte: https://developers.hubspot.com/docs/api/overview

Airbyte / Fivetran (sync/migração)

O que faz: Sync automatizado de dados/config para dual-run e migração.
Por que é a melhor aqui: Reduz engenharia manual para mover histórico e manter sincronismo.
Custo real: Preço por volume (linhas/GB) e conexões.
Custos ocultos: Histórico é o multiplicador; ambientes duplicados dobram custo.
Fonte: https://airbyte.com/product/airbyte-cloud

Camada 3 — CONTRACT-FIRST

Stoplight

O que faz: Editor OpenAPI/JSON schema + docs + mock.
Por que é a melhor aqui: Acelera contrato versionado e reduz erro.
Custo real: Preço por plano + seats.
Custos ocultos: Risco: virar fonte paralela ao repo; repo deve ser a fonte.
Fonte: https://stoplight.io/pricing

Postman

O que faz: Coleções, mocks, monitors e colaboração.
Por que é a melhor aqui: Útil para explorar e depurar contrato.
Custo real: Preço por seat + overages (runs/monitors).
Custos ocultos: Risco de drift se coleção divergir do OpenAPI no repo.
Fonte: https://www.postman.com/pricing/

Spectral + openapi-diff + AJV + Schemathesis

O que faz: Lint/diff/validação/teste gerado por contrato.
Por que é a melhor aqui: É o gate automático de contrato/API (sem humano).
Custo real: $0 (open-source) + custo CI.
Custos ocultos: Tempo CI e estabilização; mas isso é o preço do 1:1.
Fonte: https://spec.openapis.org/oas/latest.html

Prism + WireMock

O que faz: Mocks/stubs para desenvolver e testar sem backend final.
Por que é a melhor aqui: Permite front/back em paralelo e acelera entrega.
Custo real: $0 (open-source).
Custos ocultos: Se stub errado virar verdade, você cristaliza bug; versionar e comparar contra contrato.
Fonte: https://docs.stoplight.io/docs/prism

Camada 4 — Modelagem

XState

O que faz: State machines para UI e fluxos.
Por que é a melhor aqui: Controla comportamento 1:1 e reduz drift quando evoluir depois.
Custo real: $0 (open-source).
Custos ocultos: Disciplina: se não modelar, vira if-else espalhado.
Fonte: https://stately.ai/docs/xstate

Mermaid / PlantUML

O que faz: Diagramas como código para docs e rastreio.
Por que é a melhor aqui: Rastreabilidade e manutenção do ecossistema LAI.
Custo real: $0.
Custos ocultos: Nenhum relevante.
Fonte: https://mermaid.js.org/

Camada 5 — PARITY (Visual/E2E)

Applitools

O que faz: Visual regression com IA.
Por que é a melhor aqui: Menos flake e mais sinal/ruído em paridade visual 1:1.
Custo real: Preço alto e geralmente anual (published).
Custos ocultos: Baseline management e unidades de teste; cobertura sem estratégia vira ruído.
Fonte: https://applitools.com/platform-pricing/

Chromatic (Storybook)

O que faz: Visual regression por componente.
Por que é a melhor aqui: Mata regressão no Design System; complementa página completa.
Custo real: Preço por plano + overage por snapshot.
Custos ocultos: Snapshots explodem com variantes/browsers sem TurboSnap.
Fonte: https://www.chromatic.com/pricing

Percy

O que faz: Visual regression por screenshot.
Por que é a melhor aqui: Alternativa; útil se você já usa BrowserStack.
Custo real: Preço por volume de screenshots.
Custos ocultos: Multi-browser + multi-width explode volume.
Fonte: https://www.browserstack.com/docs/percy/overview

Camada 6 — Observabilidade + Incidente

Sentry

O que faz: Erros + performance + traces.
Por que é a melhor aqui: Reduz caça manual e acelera correção.
Custo real: Preço por plano + overages por eventos/spans.
Custos ocultos: Volume e retenção são multiplicadores.
Fonte: https://sentry.io/pricing/

Datadog (ou Grafana Cloud)

O que faz: Logs/métricas/traces e alertas.
Por que é a melhor aqui: Operação industrial do ecossistema LAI.
Custo real: Preço por host + logs por GB + retenção.
Custos ocultos: Cardinalidade e logs indexados explodem custo.
Fonte: https://www.datadoghq.com/pricing/

PagerDuty

O que faz: Incident workflow / Andon cord.
Por que é a melhor aqui: Para a linha quando gates falham sem depender de humano ver.
Custo real: Preço por usuário.
Custos ocultos: Add-ons e crescimento de seats.
Fonte: https://www.pagerduty.com/pricing/

Camada 7 — Segurança

Semgrep

O que faz: SAST/SCA/Secrets com bom sinal/ruído.
Por que é a melhor aqui: Complementa CodeQL e captura patterns do seu stack.
Custo real: Preço por contributor + produto.
Custos ocultos: Escala por contributors e regras; risco de alert fatigue.
Fonte: https://semgrep.dev/pricing

Trivy + Syft/Grype + Gitleaks

O que faz: Container scan + SBOM + secrets (open-source).
Por que é a melhor aqui: Gates automáticos baratos.
Custo real: $0 + custo CI.
Custos ocultos: Tempo CI e tuning.
Fonte: https://aquasecurity.github.io/trivy/

Presidio (PII sanitization)

O que faz: Detecção/máscara de PII em HAR/traces/logs.
Por que é a melhor aqui: Evita vazamento e facilita retenção mínima.
Custo real: $0 (open-source).
Custos ocultos: Demanda tuning de detectores e regras.
Fonte: https://microsoft.github.io/presidio/

Camada 8 — IA e automação

GitHub Copilot

O que faz: Acelera codegen/review e reduz mão humana.
Por que é a melhor aqui: Complementa a Fábrica para finalizar bordas e patches.
Custo real: Preço por usuário/planos.
Custos ocultos: Uso descontrolado e requests premium viram custo.
Fonte: https://github.com/features/copilot

Cursor / Continue

O que faz: IDE agente para acelerar correções e PRs.
Por que é a melhor aqui: Reduz tempo humano no 'último 10%'.
Custo real: Preço por seat + custo do modelo (se BYOK).
Custos ocultos: Modelo caro vira vazamento; precisa budget e WIP.
Fonte: https://cursor.com/pricing

Camada 9 — Design System

Figma

O que faz: Design system, componentes e handoff.
Por que é a melhor aqui: Para 1:1 visual, você precisa de biblioteca de componentes controlada.
Custo real: Preço por seat.
Custos ocultos: Escala por seats; governar quem precisa de full seat.
Fonte: https://www.figma.com/pricing/

Camada 10 — Knowledge (LAI)

Pinecone / Weaviate / Qdrant

O que faz: Vector DB para docs/contratos/logs/run reports.
Por que é a melhor aqui: Acelera busca e automação do ecossistema LAI.
Custo real: Preço por uso (storage + read/write) ou por recursos (CPU/RAM/disk).
Custos ocultos: Dimensionamento e volume de queries são multiplicadores.
Fonte: https://www.pinecone.io/pricing/

Apify

O que faz: Scraping estruturado para pesquisa/ingestão automatizada.
Por que é a melhor aqui: Alimenta Pré-Fábrica com dados estruturados sem humano.
Custo real: Preço por plano + compute/proxy.
Custos ocultos: Proxies residenciais e compute unit explodem se não houver limites.
Fonte: https://apify.com/pricing

Simulação (prova) — Pack Factory executando de verdade

Prova do núcleo da Fábrica (pack-first): gerar Pack0, validar, gerar RUN_REPORT, APPROVAL e promover snapshot (merge promoted).

Etapa	Comando (curto)	Saída
1	pack0 meetcore	pack0-meetcore-0.0.1.zip
2	validate-pack0 pack0.zip	pack0_validation.json
3	run-report pack0.zip	run_report.json
4	approve-pack pack0.zip + rr.json	approval.json
5	wrap-* + merge promoted	snapshot_promoted.zip

Runbook: 5 botões (o que cada um dispara)

1) CAPTURE + EXTRACT

• Dispara Playwright (UI) e mitmproxy (rede) no runner isolado.
• Extrai config por API/exports (pipelines/properties/permissions/views).
• Gera artifacts: trace.zip, screenshots.zip, flow.json, har, config.json.

2) BUILD BLUEPRINT

• Normaliza inputs e gera blueprint.json + openapi.yaml + paridade_matrix.json.
• Versiona por trace_id e salva no artifact store.

3) GENERATE PACK0

• Chama pack-factory pack0 + validate-pack0 + run-report.
• Se validate falhar: retorna gaps e bloqueia.

4) GENERATE PACK1

• Gera scaffold (front/back), mocks, contracts e testes base.
• Dispara CI gates e publica artifacts.

5) PARITY GATE

• Roda E2E + visual diff + contract tests.
• Se falhar: parity_report + diffs e bloqueia promoção.

Referências (links)

Lovable credits: https://docs.lovable.dev/introduction/plans-and-credits
Supabase billing: https://supabase.com/docs/guides/platform/billing-on-supabase
Temporal pricing: https://temporal.io/get-cloud/aws-marketplace
GitHub Actions billing: https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions
Playwright: https://playwright.dev/
mitmproxy: https://docs.mitmproxy.org/stable/
Applitools: https://applitools.com/platform-pricing/
Chromatic: https://www.chromatic.com/pricing
Semgrep: https://semgrep.dev/pricing
Presidio: https://microsoft.github.io/presidio/