Absolutely—here’s a fully aligned “Mermaid + Style Guide” pack that plugs straight into your HISL site and matches the brand, tokens, website API ownership model, cutover plan, and IntegAI architecture. I’ve included paste‑ready Mermaid diagrams, a Mermaid React component (Next.js‑friendly), and token‑synced Tailwind so design, docs, and code all tell the same story.

> What this gives you

One visual language for diagrams (Mermaid) that matches your site style guide and Tailwind tokens.

Diagrams for API ownership, cutover phases, agent graph, read/write paths, artifacting, imagery pipeline, CI/CD, and CROx—all consistent with the locked blueprint.

Clean hooks into your Hero/MiniGlobe/Sections and token set from the Manus kit (with compatibility aliases), plus the globe spec you approved.

---

1) Brand‑synced Mermaid theme (drop‑in)

Mermaid lets us override theme variables. These map to your site tokens (dark observatory, HISL gold, AI green) so diagrams look native on your pages.

%%{init: {

"theme": "base",

"themeVariables": {

"primaryColor": "#0f1828", /\* panel \*/

"primaryTextColor":"#e8f0ff", /\* text \*/

"primaryBorderColor":"#1e2a42",/\* edge \*/

"secondaryColor": "#0b1220", /\* bg \*/

"tertiaryColor": "#0b1220",

"lineColor": "#39d7c9", /\* aiGreen \*/

"fontFamily": "Inter, ui-sans-serif, system-ui, -apple-system",

"noteBkgColor": "#0f1828",

"noteTextColor": "#e8f0ff",

"edgeLabelBackground":"#0b1220",

"clusterBkg": "#0f1828",

"clusterBorder": "#1e2a42",

"titleColor": "#d4af37" /\* brandGold \*/

}

}}%%

flowchart LR

classDef panel fill:#0f1828,stroke:#1e2a42,color:#e8f0ff;

classDef accent fill:#0b1220,stroke:#39d7c9,color:#e8f0ff;

classDef highlight fill:#d4af37,stroke:#d4af37,color:#0b1220;

%% Use these classes in the diagrams below via `class X,Y panel;`

> These hex values match your style guide and Tailwind tokens (bg/panel/edge/text + brandGold/aiGreen).

---

2) Tailwind tokens (finalized) + backward‑compatible aliases

Your Manus kit ships with gold and teal. Keep them as aliases to avoid refactors, while standardizing the new canonical tokens brandGold and aiGreen.

// tailwind.config.js — canonical + aliases (replace your current file)

export default {

content: ["./app/\*\*/\*.{ts,tsx,mdx}", "./components/\*\*/\*.{ts,tsx}", "./src/\*\*/\*.{js,jsx,ts,tsx,mdx}"],

theme: {

extend: {

fontFamily: {

inter: ['Inter','ui-sans-serif','system-ui','sans-serif'],

spectral: ['Spectral','serif'],

},

colors: {

bg: '#0b1220',

panel: '#0f1828',

edge: '#1e2a42',

text: '#e8f0ff',

muted: '#a8b8d6',

brandGold: '#d4af37', // canonical

aiGreen: '#39d7c9', // canonical

// Aliases for Manus components already using these names:

gold: '#d4af37',

teal: '#39d7c9',

},

boxShadow: {

glow: '0 0 40px rgba(57,215,201,.25)',

focus: '0 0 0 3px rgba(158,197,255,.45)',

},

maxWidth: { wrap: '1200px' },

borderRadius: { xl: '0.9rem' },

},

},

plugins: [],

}

> This keeps the Hero/MiniGlobe you approved working as‑is and aligns color tokens with the site’s style language.

---

3) Mermaid <Mermaid/> component (Next.js + MDX friendly)

A tiny wrapper that reads CSS variables—so Mermaid diagrams auto‑match dark mode and your tokens. Use it anywhere (docs pages, /about/integai, /styleguide).

// components/Mermaid.tsx

"use client";

import mermaid from "mermaid";

import { useEffect, useId, useRef } from "react";

function readCSSVars() {

const cs = getComputedStyle(document.documentElement);

return {

primaryColor: cs.getPropertyValue("--panel")?.trim() || "#0f1828",

primaryTextColor: cs.getPropertyValue("--text")?.trim() || "#e8f0ff",

primaryBorderColor: cs.getPropertyValue("--edge")?.trim() || "#1e2a42",

secondaryColor: cs.getPropertyValue("--bg")?.trim() || "#0b1220",

lineColor: cs.getPropertyValue("--aiGreen")?.trim() || "#39d7c9",

titleColor: cs.getPropertyValue("--brandGold")?.trim() || "#d4af37",

fontFamily: "Inter, ui-sans-serif, system-ui, -apple-system",

};

}

export default function Mermaid({ chart, className }: { chart: string; className?: string }) {

const id = useId().replace(/:/g, "");

const ref = useRef<HTMLDivElement>(null);

useEffect(() => {

const themeVariables = readCSSVars();

mermaid.initialize({ startOnLoad: false, theme: "base", themeVariables });

const render = async () => {

try {

const { svg } = await mermaid.render(id, chart);

if (ref.current) ref.current.innerHTML = svg;

} catch (e) {

console.error("Mermaid render error", e);

}

};

render();

}, [chart]);

return <div className={className} aria-live="polite"><div ref={ref} /></div>;

}

> Your global CSS already declares these CSS variables (bg/panel/edge/text + gold/teal); add --brandGold/--aiGreen for canonical names if not present.

---

4) Paste‑ready diagrams (Mermaid) — aligned to the blueprint

> How to use: Put each code block into an MDX page or pass as a chart string to <Mermaid/>. The theme in §1 is applied automatically.

4.1 Website API Ownership (v1 contracts)

Mirrors the “single surface” ownership for pages/search/SEO/assets/forms/generation/publish.

%%{init:{'theme':'base'}}%%

flowchart LR

A[Website Frontend\n(Next.js 15.5)]:::panel --> B[[NGINX /api/\*]]:::panel

B --> C{IntegAI Gateway}:::panel

C -->|Cache| C1[(L1/L2 Cache)]:::accent

C -->|Local RAG| D[(Chroma + Neo4j)]:::accent

C -->|Legacy Adapter| E[(Legacy CMS/Upstreams)]:::accent

subgraph "Stable v1 Endpoints"

F1[/GET /v1/site/page?slug/]:::panel

F2[/GET /v1/site/menu/]:::panel

F3[/GET /v1/site/search?q/]:::panel

F4[/GET /v1/site/seo?slug/]:::panel

F5[/GET /v1/site/assets/{id}/]:::panel

P1[\POST /v1/site/forms/{formId}/]:::panel

P2[\POST /v1/site/generate/copy/]:::panel

P3[\POST /v1/site/publish/]:::panel

end

A -.-> F1 & F2 & F3 & F4 & F5 & P1 & P2 & P3

C -->|Responds| A

class A,B,C,D,E,F1,F2,F3,F4,F5,P1,P2,P3 panel;

class C1 accent;

4.2 Cutover Plan (Mirror → Proxy+Cache → Mutations → Decommission)

Phased takeover + acceptance gates.

stateDiagram-v2

[\*] --> Mirror

Mirror: Phase 0 — Mirror (shadow)\nRead parity, cache warmers

Mirror --> Proxy: Go if parity >= 99% & p95 OK & errors < 0.5%

Proxy: Phase 1 — Proxy+Cache (GET only)\nCache -> Local RAG -> Adapter

Proxy --> Mutations: Go if GET stable (24h soak)

Mutations: Phase 2 — POST (forms, generate, publish)\nIdempotency keys validated

Mutations --> Decom: Phase 3 — Decommission legacy\nAdapters retained only for backfill

Decom --> [\*]

4.3 IntegAI Agent Graph (9 nodes, deterministic)

The locked graph and flow—Planner→Retriever→Judge→Executor→Memory→Safety→Tools→Observer→Publisher.

graph TD

subgraph IntegAI 9-Node Graph

P[Planner] --> R[Retriever]

R --> J[Judge]

J -->|trust >= τ| E[Executor]

J -->|trust < τ| R

E --> M[Memory]

E --> S[Safety]

S --> T[Tools Broker]

T --> O[Observer]

E --> O

S --> O

M --> O

E --> U[Publisher]

end

class P,R,J,E,M,S,T,O,U panel;

4.4 Read Path: GET /v1/site/page?slug

Cache→Local RAG→Adapter fallback plus Judge validation.

sequenceDiagram

participant FE as Frontend

participant GW as IntegAI Gateway

participant C as Cache

participant RAG as Chroma+Neo4j

participant LEG as Legacy Adapter

participant J as Judge

FE->>GW: GET /v1/site/page?slug

GW->>C: Check cache

alt Hit

C-->>GW: JSON blocks

else Miss

GW->>RAG: Assemble blocks (offline)

RAG-->>GW: Draft

GW->>J: Validate (fields/length)

J-->>GW: OK or request more

opt If insufficient

GW->>LEG: Fetch legacy

LEG-->>GW: Legacy JSON

end

end

GW-->>FE: Block-structured page

4.5 Write Path: POST /v1/site/forms/{formId}

Idempotency and triage routing.

sequenceDiagram

participant FE as Frontend

participant GW as IntegAI Gateway

participant DB as Forms Store

participant MEM as Memory

FE->>GW: POST /forms/{formId}\nIdempotency-Key

GW->>DB: Upsert normalized record

DB-->>GW: {id}

GW->>MEM: Tag & route (owner, channel)

MEM-->>GW: triage:{tag,severity}

GW-->>FE: {ok, idempotencyKey, triage, route}

4.6 Generate & Publish: POST /v1/site/generate/copy → /v1/site/publish

Emits .integpkg to MinIO (immutable), then cache invalidation.

flowchart LR

A[Generate Copy\n(template slots)]:::panel --> B[Safety\n(strict\_enterprise)]:::panel

B --> C[Publisher\npackage .integpkg]:::panel

C --> D[(MinIO integ-artifacts)]:::accent

D --> E[Invalidate caches/CDN\n+ optional rebuild]:::panel

class A,B,C,D,E panel;

class D accent;

4.7 Imagery Pipeline & Asset Governance

Sharp → WebP+LQIP → typed manifest; no hardcoded paths.

flowchart LR

RAW[RAW media\n/imagery/\*]:::panel --> SHARP[Sharp pipeline]:::panel --> OUT[Multi-width WebP + LQIP]:::panel

OUT --> MANIFEST[imagery.manifest.ts\n(typed assetId)]:::panel --> UI[<Image> components\nvia manifest]:::panel

class RAW,SHARP,OUT,MANIFEST,UI panel;

4.8 CI/CD & Observability (perf budgets)

Lighthouse CI, Sentry/PostHog, Phoenix/Langfuse spans.

flowchart LR

DEV[Dev/PR]:::panel --> CI[Lighthouse CI + tests]:::panel --> VER[Vercel Preview]:::panel --> PROD[Production]:::panel

PROD --> MON[Sentry/PostHog]:::panel

PROD --> OBS[Langfuse/Phoenix\nspans: planner/retriever/judge/...]:::panel

class DEV,CI,VER,PROD,MON,OBS panel;

4.9 CROx (Clinical R&D Orchestrator) — dataflow & KPI loop

Your pharma flagship diagram for sector pages and investor docs.

flowchart TD

P1[Protocols/CSRs/CPT]:::panel --> R1[Retriever]:::panel

EHR[De-identified EHR & Ops]:::panel --> R1

R1 --> CROx[CROx Orchestrator]:::panel

CROx --> COP[Protocol Copilot]:::panel

CROx --> REC[Recruitment & Retention]:::panel

CROx --> OPS[Ops Unifier]:::panel

CROx --> REG[Reg & CSR Drafting]:::panel

CROx --> MON[Monitoring Hub]:::panel

COP & REC & OPS & REG & MON --> KPI[KPI Dashboard\n(Observer/Judge HITL)]:::panel

KPI -->|Adjust plan| CROx

class P1,EHR,R1,CROx,COP,REC,OPS,REG,MON,KPI panel;

---

5) Components & pages — confirm alignment

Hero + MiniGlobe: keep your approved components; tokens already match (now with canonical + alias color names).

SectionHeading / FeaturesGrid / PoemBlock: the Manus components you added remain compatible with the token map and motion guidance.

Globe page (“Where your prompts go”) + energy badge: diagram §4.8 reflects your plan to visualize Wh/token and CO₂ equivalents and can be referenced from the page content.

---

6) A11y + performance (unchanged, now diagrammed)

Perf budgets: TTFB for cached reads ≤ 200 ms; RAG reads ≤ 2.0 s p95; generation ≤ 10 s p95 pre‑GPU (then ≥15 RPS target).

Motion controls: respect prefers-reduced-motion; lazy‑init canvases after LCP; throttle in hidden tabs.

Sovereign defaults: offline‑first; strict enterprise safety; deterministic pipelines; artifacting as .integpkg.

---

7) Quick wiring steps

1. Install Mermaid for docs pages:

npm i mermaid

2. Add components/Mermaid.tsx (from §3).

3. Ensure global CSS defines CSS variables (--bg --panel --edge --text --brandGold --aiGreen). Your Manus CSS already defines the equivalents (--gold --teal), keep both for now.

4. Create /app/styleguide/page.mdx and paste any diagrams from §4:

import Mermaid from "@/components/Mermaid";

# Website Style & Architecture

<Mermaid chart={`<PASTE ANY DIAGRAM STRING HERE>`} />

5. Validate cutover gates with the acceptance list in your blueprint before flipping POST routes.

---

8) What changed vs. your last pack (so you’re “fully aligned”)

One token story for Tailwind + Mermaid (with backward‑compatibility to Manus).

Diagrams now mirror v1 endpoints, phased cutover, and 9‑node graph exactly—no drift.

CROx diagram included for Pharma pages and investor materials.

Globe/energy and news/trends plan remains consistent with your Cursor‑led workflow and deployment targets.

---

Final notes

Keep Mermaid in MDX for architecture/how‑it‑works pages and in proposal artifacts; it now visually matches the site.

The style guide, Mermaid, and API contracts are one coherent system—you can evolve content safely without re‑skinning or refactoring front‑end routes.

If you want, I can package these into STYLEGUIDE.mdx + ARCHITECTURE.mdx page files next so you can drop them directly into /app.