Absolutely—let’s ship a polished, self-contained “Where Your Prompts Go” page you can drop into the codebase. It includes:

A realistic 3D globe (day map + clouds)

Huginn & Muninn as gold raven sprites on different orbits

Data-center pins (curated list)

Prompt form + output panel

A pulse packet that flies hop-by-hop across data centers, then returns to the output box

Below are full files (no edits—paste as new files). I’ve kept them framework-agnostic React (Vite/Next both fine). Manus can wire the route and install deps.

---

0) Pre-flight (Manus / package)

Ask Manus (or run yourself) to add these:

npm i three @react-three/fiber @react-three/drei framer-motion

Public assets expected (you already have these):

/public/assets/earth\_daymap.jpg

/public/assets/earth\_nightmap.jpg (optional, not strictly used here)

/public/assets/earth\_clouds.jpg

/public/images/raven\_huginn.png

/public/images/raven\_muninn.png

---

1) src/data/datacenters.js — curated locations

// src/data/datacenters.js

export const DATA\_CENTERS = [

// --- Europe

{ name: "Dublin, IE", lat: 53.35, lng: -6.26, region: "EU", provider: "Azure/Equinix" },

{ name: "Frankfurt, DE", lat: 50.11, lng: 8.68, region: "EU", provider: "AWS/Azure/DRT" },

{ name: "London, UK", lat: 51.51, lng: -0.13, region: "EU", provider: "Multi" },

{ name: "Amsterdam, NL", lat: 52.37, lng: 4.90, region: "EU", provider: "Multi" },

{ name: "Paris, FR", lat: 48.86, lng: 2.35, region: "EU", provider: "Multi" },

{ name: "Warsaw, PL", lat: 52.23, lng: 21.01, region: "EU", provider: "Multi" },

// --- North America

{ name: "Ashburn, US-VA", lat: 39.04, lng: -77.49, region: "NA", provider: "AWS/Equinix/DRT" },

{ name: "Iowa, US-IA", lat: 41.60, lng: -93.61, region: "NA", provider: "GCP" },

{ name: "Quincy, US-WA", lat: 47.23, lng: -119.85,region: "NA", provider: "Azure" },

{ name: "Toronto, CA", lat: 43.65, lng: -79.38, region: "NA", provider: "Multi" },

// --- Asia / Oceania

{ name: "Singapore, SG", lat: 1.29, lng: 103.85, region: "APAC",provider: "Multi" },

{ name: "Tokyo, JP", lat: 35.68, lng: 139.69, region: "APAC",provider: "Multi" },

{ name: "Sydney, AU", lat: -33.86, lng: 151.21, region: "APAC",provider: "Multi" },

{ name: "Mumbai, IN", lat: 19.07, lng: 72.88, region: "APAC",provider: "Multi" },

{ name: "Seoul, KR", lat: 37.56, lng: 126.98, region: "APAC",provider: "Multi" },

{ name: "Hong Kong, HK", lat: 22.32, lng: 114.17, region: "APAC",provider: "Multi" },

// --- Africa

{ name: "Johannesburg, ZA", lat: -26.20, lng: 28.04, region: "AFR", provider: "Multi" },

{ name: "Cape Town, ZA", lat: -33.92, lng: 18.42, region: "AFR", provider: "Multi" },

{ name: "Nairobi, KE", lat: -1.29, lng: 36.82, region: "AFR", provider: "Emerging" },

// --- LatAm

{ name: "São Paulo, BR", lat: -23.55, lng: -46.63, region: "LATAM",provider: "Multi" },

{ name: "Mexico City, MX", lat: 19.43, lng: -99.13, region: "LATAM",provider: "Multi" },

];

---

2) src/utils/spherical.js — geo helpers

// src/utils/spherical.js

import \* as THREE from "three";

/\*\* Convert lat/lng to Vector3 on a sphere of given radius \*/

export function latLngToVec3(lat, lng, radius = 1) {

const phi = (90 - lat) \* (Math.PI / 180);

const theta = (lng + 180) \* (Math.PI / 180);

const x = -radius \* Math.sin(phi) \* Math.cos(theta);

const z = radius \* Math.sin(phi) \* Math.sin(theta);

const y = radius \* Math.cos(phi);

return new THREE.Vector3(x, y, z);

}

/\*\* Great-circle interpolation between two lat/lng points; returns positions on sphere \*/

export function greatCirclePoints(a, b, radius = 1, steps = 180) {

const va = latLngToVec3(a.lat, a.lng, radius).normalize();

const vb = latLngToVec3(b.lat, b.lng, radius).normalize();

const angle = Math.acos(THREE.MathUtils.clamp(va.dot(vb), -1, 1));

const pts = [];

for (let i = 0; i <= steps; i++) {

const t = i / steps;

const sinTotal = Math.sin(angle);

const s1 = Math.sin((1 - t) \* angle) / (sinTotal || 1e-6);

const s2 = Math.sin(t \* angle) / (sinTotal || 1e-6);

const v = new THREE.Vector3().addScaledVector(va, s1).addScaledVector(vb, s2).normalize().multiplyScalar(radius);

pts.push(v);

}

return pts;

}

---

3) src/components/globe/GlobeScene.jsx — the 3D scene

// src/components/globe/GlobeScene.jsx

import React, { useMemo, useRef, useImperativeHandle, forwardRef } from "react";

import { Canvas, useFrame, useLoader } from "@react-three/fiber";

import { OrbitControls, Html, Sprite } from "@react-three/drei";

import \* as THREE from "three";

import { DATA\_CENTERS } from "../../data/datacenters";

import { latLngToVec3, greatCirclePoints } from "../../utils/spherical";

const dayMapURL = "/assets/earth\_daymap.jpg";

const cloudsURL = "/assets/earth\_clouds.jpg";

const ravenHuginnURL = "/images/raven\_huginn.png";

const ravenMuninnURL = "/images/raven\_muninn.png";

const GLOBE\_R = 1.1;

const CLOUDS\_R = 1.115;

function Earth() {

const [colorMap, cloudsMap] = useLoader(THREE.TextureLoader, [dayMapURL, cloudsURL]);

colorMap.anisotropy = 8;

cloudsMap.anisotropy = 8;

return (

<>

<mesh>

<sphereGeometry args={[GLOBE\_R, 64, 64]} />

<meshStandardMaterial map={colorMap} roughness={1} metalness={0} />

</mesh>

<mesh>

<sphereGeometry args={[CLOUDS\_R, 64, 64]} />

<meshStandardMaterial map={cloudsMap} transparent opacity={0.25} depthWrite={false} />

</mesh>

</>

);

}

function DataPins() {

const geom = useMemo(() => new THREE.SphereGeometry(0.01, 8, 8), []);

const matEU = useMemo(() => new THREE.MeshBasicMaterial({ color: "#78a7ff" }), []);

const matNA = useMemo(() => new THREE.MeshBasicMaterial({ color: "#39d7c9" }), []);

const matAP = useMemo(() => new THREE.MeshBasicMaterial({ color: "#f6c650" }), []);

const matAF = useMemo(() => new THREE.MeshBasicMaterial({ color: "#ff7a7a" }), []);

const matLA = useMemo(() => new THREE.MeshBasicMaterial({ color: "#b17aff" }), []);

return (

<group>

{DATA\_CENTERS.map((d, i) => {

const pos = latLngToVec3(d.lat, d.lng, GLOBE\_R + 0.01);

const mat = d.region === "EU" ? matEU :

d.region === "NA" ? matNA :

d.region === "APAC" ? matAP :

d.region === "AFR" ? matAF : matLA;

return (

<mesh key={i} position={pos} geometry={geom} material={mat} />

);

})}

</group>

);

}

function Raven({ url, radius = 1.25, tilt = 23, speed = 0.25, scale = 0.18 }) {

const spriteTex = useLoader(THREE.TextureLoader, url);

const ref = useRef();

useFrame(({ clock }) => {

const t = clock.getElapsedTime() \* speed;

const inc = THREE.MathUtils.degToRad(tilt);

const theta = t \* 2.0;

// inclined circular orbit

const x = radius \* Math.cos(theta);

const y = radius \* Math.sin(theta) \* Math.sin(inc);

const z = radius \* Math.sin(theta) \* Math.cos(inc);

ref.current.position.set(x, y, z);

});

return (

<Sprite ref={ref} scale={[scale, scale, scale]}>

<spriteMaterial map={spriteTex} transparent />

</Sprite>

);

}

function PulsePacket({ route, playingRef }) {

// route = [ [vec3,...], [vec3,...], ... ]

const dot = useRef();

const trail = useRef([]);

const speed = 0.8; // points per frame-ish

const idxRef = useRef({ leg: 0, i: 0 });

useFrame(() => {

if (!route || !route.length || !playingRef.current) return;

const { leg, i } = idxRef.current;

const legPts = route[leg];

if (!legPts) return;

const nextI = i + speed;

const iInt = Math.floor(nextI);

const cur = legPts[Math.min(iInt, legPts.length - 1)];

if (cur) {

dot.current.position.copy(cur.clone().multiplyScalar(1.002));

// tiny fading spheres for a trail

const tl = new THREE.Mesh(

new THREE.SphereGeometry(0.006, 6, 6),

new THREE.MeshBasicMaterial({ color: "#39d7c9", transparent: true, opacity: 0.6 })

);

tl.position.copy(dot.current.position);

dot.current.parent.add(tl);

trail.current.push({ mesh: tl, life: 1 });

}

// fade and cleanup trail

trail.current.forEach(t => {

t.life -= 0.02;

t.mesh.material.opacity = Math.max(0, t.life);

if (t.life <= 0) {

dot.current.parent.remove(t.mesh);

t.mesh.geometry.dispose();

t.mesh.material.dispose();

}

});

trail.current = trail.current.filter(t => t.life > 0);

if (iInt >= legPts.length - 1) {

// next leg or stop

if (leg < route.length - 1) {

idxRef.current = { leg: leg + 1, i: 0 };

} else {

playingRef.current = false; // finished

}

} else {

idxRef.current.i = nextI;

}

});

return (

<mesh ref={dot}>

<sphereGeometry args={[0.012, 8, 8]} />

<meshBasicMaterial color="#39d7c9" />

</mesh>

);

}

const SceneInner = forwardRef(function SceneInner(\_, ref) {

const group = useRef();

const playingRef = useRef(false);

const routeRef = useRef(null);

useImperativeHandle(ref, () => ({

/\*\* Start a route animation: origin -> ...via -> origin \*/

playRoute({ origin = { lat: 53.35, lng: -6.26 }, via = [] }) {

const legs = [];

let prev = origin;

via.forEach(v => {

legs.push(greatCirclePoints(prev, v, GLOBE\_R + 0.02, 240));

prev = v;

});

legs.push(greatCirclePoints(prev, origin, GLOBE\_R + 0.02, 240));

routeRef.current = legs;

playingRef.current = true;

}

}));

useFrame((\_, dt) => {

// slow auto-rotation

group.current.rotation.y += dt \* 0.06;

});

return (

<group ref={group}>

<ambientLight intensity={0.7} />

<directionalLight intensity={1.2} position={[3, 2, 2]} />

<Earth />

<DataPins />

{/\* Ravens \*/}

<Raven url={ravenHuginnURL} radius={1.28} tilt={24} speed={0.30} scale={0.18} />

<Raven url={ravenMuninnURL} radius={1.36} tilt={-15} speed={0.22} scale={0.20} />

{/\* Pulse packet (only visible when playing) \*/}

{routeRef.current && <PulsePacket route={routeRef.current} playingRef={playingRef} />}

{/\* Controls \*/}

<OrbitControls enablePan={false} enableZoom={true} enableDamping dampingFactor={0.06} />

</group>

);

});

export default function GlobeScene(props) {

const sceneRef = React.useRef();

return (

<div className="relative aspect-[16/10] w-full rounded-2xl border border-edge bg-[#090f1c]">

<Canvas camera={{ position: [0, 0, 3.2], fov: 45 }}>

<Html position={[0, 0, 0]} zIndexRange={[1, 0]}>

{/\* Optional overlay content \*/}

</Html>

<SceneInner ref={sceneRef} />

</Canvas>

{/\* Expose an imperative handle to parent \*/}

{props.innerRef && (props.innerRef.current = sceneRef.current)}

</div>

);

}

---

4) src/pages/GlobeVisualizer.jsx — full page with UI + API

// src/pages/GlobeVisualizer.jsx

import React from "react";

import { motion } from "framer-motion";

import GlobeScene from "../components/globe/GlobeScene";

import { DATA\_CENTERS } from "../data/datacenters";

const API\_BASE = ""; // same-origin via proxy (/api -> server), or set http://localhost:8787

// pick a few canonical hops per run (example: Dublin -> Frankfurt -> Ashburn)

const HOP\_SETS = [

["Dublin, IE", "Frankfurt, DE", "Ashburn, US-VA"],

["Dublin, IE", "London, UK", "Toronto, CA"],

["Dublin, IE", "Amsterdam, NL", "Singapore, SG"],

["Dublin, IE", "Frankfurt, DE", "Tokyo, JP"],

];

function findDC(name) {

return DATA\_CENTERS.find(d => d.name === name);

}

export default function GlobeVisualizer() {

const [prompt, setPrompt] = React.useState("");

const [out, setOut] = React.useState("");

const [runInfo, setRunInfo] = React.useState(null);

const globeRef = React.useRef(null);

async function runDemo(e) {

e?.preventDefault?.();

setOut("Running…");

const hops = HOP\_SETS[Math.floor(Math.random() \* HOP\_SETS.length)];

const via = hops.map(findDC).filter(Boolean);

// trigger globe route

globeRef.current?.playRoute?.({ origin: findDC("Dublin, IE"), via });

try {

const r = await fetch(`${API\_BASE}/api/deepseek`, {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({

agentId: "SovereignAssistant",

prompt: prompt || "Explain sovereign AI in one sentence.",

context: { page: "globe-visualizer" }

})

});

const ct = r.headers.get("content-type") || "";

const data = ct.includes("application/json") ? await r.json() : await r.text();

// naive “energy” estimate: length-based placeholder

const chars = JSON.stringify(data).length;

const estJoules = Math.round(chars \* 0.5); // demo only

const storedLikely = via[Math.floor(Math.random() \* via.length)]?.name || "Local";

setRunInfo({ hops, estJoules, storedLikely });

setOut(typeof data === "string" ? data : (data.choices?.[0]?.message?.content || JSON.stringify(data, null, 2)));

} catch (err) {

setOut(`Error: ${String(err)}`);

}

}

return (

<main className="py-10">

<div className="container-wrap grid gap-6 lg:grid-cols-[1.1fr\_.9fr] items-start">

{/\* Left: Globe \*/}

<section>

<motion.div

initial={{ opacity: 0, y: 10 }}

animate={{ opacity: 1, y: 0 }}

transition={{ duration: .5 }}

>

<h1 className="text-[clamp(28px,3.2vw,44px)] font-semibold mb-2">

Where Your Prompts Go

</h1>

<p className="text-muted max-w-[68ch]">

A sovereignty visualizer. Two ravens guide your request across representative data centers,

then return with the result — fully auditable.

</p>

</motion.div>

<div className="mt-4">

<GlobeScene innerRef={globeRef} />

</div>

{/\* Legend \*/}

<div className="mt-3 text-[13px] text-muted">

<span className="inline-block w-2 h-2 rounded-full bg-[#39d7c9] mr-2" />

Primary route &nbsp;·&nbsp;

<span className="inline-block w-2 h-2 rounded-full bg-[#78a7ff] mx-2" />

Regional pins &nbsp;·&nbsp; Huginn & Muninn in gold orbits

</div>

</section>

{/\* Right: Prompt/Output \*/}

<aside>

<form onSubmit={runDemo} className="rounded-2xl border border-edge bg-panel p-4">

<label className="block text-[13px] mb-1 text-muted">Prompt</label>

<textarea

value={prompt}

onChange={(e) => setPrompt(e.target.value)}

placeholder="Ask something…"

rows={5}

className="w-full rounded-lg border border-edge bg-[#0f1828] text-[15px] p-3 outline-none focus:border-[#39d7c9]"

/>

<div className="mt-3 flex items-center gap-2">

<button type="submit" className="btn btn-gold">Run Simulation</button>

<span className="text-[12px] text-muted">IntegAI Simulation Mode</span>

{/\* place your IntegAI logo small here if desired \*/}

</div>

</form>

<div className="rounded-2xl border border-edge bg-panel p-4 mt-4">

<div className="text-[13px] text-muted mb-1">Output</div>

<pre className="text-[13px] whitespace-pre-wrap leading-relaxed">

{out || "(answer will appear here)"}

</pre>

{runInfo && (

<div className="mt-3 text-[13px] text-muted">

<div>Route: {runInfo.hops.join(" → ")}</div>

<div>Est. energy used (demo): {runInfo.estJoules} J</div>

<div>Likely storage region (demo): {runInfo.storedLikely}</div>

</div>

)}

</div>

{/\* Small data-center table \*/}

<div className="rounded-2xl border border-edge bg-panel p-4 mt-4">

<div className="text-[13px] text-muted mb-2">Regions & Pins</div>

<div className="max-h-[260px] overflow-auto text-[13px] leading-6">

{DATA\_CENTERS.map((d, i) => (

<div key={i} className="flex justify-between gap-2 border-b border-edge/40 py-1">

<span>{d.name}</span>

<span className="text-muted">{d.provider}</span>

</div>

))}

</div>

</div>

</aside>

</div>

</main>

);

}

---

5) Hooking it into navigation

If you’re still on a single-page Vite app: create a menu link to #globe or make this page a separate route using react-router-dom. Manus can wire:

npm i react-router-dom

Then mount <Route path="/globe" element={<GlobeVisualizer/>} /> and link your “Where Your Prompts Go” buttons to /globe.

If you move to Next.js (recommended on Vercel), place the file at app/globe/page.tsx (or pages/globe.tsx) and convert imports to TypeScript if needed (identical JSX works in .tsx).

---

Notes / polish ideas

Raven art: we’re using your gold ravens as sprites, so they always face the camera and look crisp.

Energy metrics: right now it’s a demo estimate. When you’re ready, we can swap to a real API (e.g., cloud provider region KPIs or Arbor.eco-style CO₂ factors).

Heatmap overlay: we can add a faint atmospheric shader or colored hemisphere per region; for now pins keep the scene clean.

Pulse trails: implemented as a flying packet + fading breadcrumbs (fast and pretty). If you want neon arcs, we can add curve tubes with gradient materials next.

---

If you want, I can also give you a matching “MiniGlobe” hero component (lighter canvas) so the landing page globe visually matches this page.