Epinoetic Foundry App: The Immortal Code Artifact

```
import React, { useEffect, useRef, useState, useMemo, useCallback }
from "react";
import { Canvas, useFrame } from "@react-three/fiber";
import { OrbitControls, Stars, Html } from "@react-three/drei";
import { motion, AnimatePresence } from "framer-motion";
import { Play, Zap, Eye, RotateCcw } from "lucide-react";
import { v4 as uuidv4 } from "uuid";

// NOTE: Tailwind classes are used throughout; Tailwind must be
configured in the project.
// This single-file React component is a scaffold for a mobile-first
Epinoetic Foundry UI.
// It includes: Overture, Forge, Crucible, Daemon's Whisper screens +
a lightweight WebSocket
// telemetry hook and 3D visuals using R3F. Replace WS_URL with your
stream endpoint.

const WS_URL = "ws://localhost:8080/stream"; // Replace with Daedalus
Coordinator StreamConsciousnessMetrics

/* --------------------------- Utilities ---------------------------
*/
function lerp(a, b, t) {
  return a + (b - a) * t;
}

function pasToColor(pas) {
  // Map PAS [0,1] -> hue 270 (purple) -> 180 (cyan)
  const hue = lerp(270, 180, Math.min(Math.max(pas, 0), 1));
  return `hsl(${hue} 75% 55%)`;
}

const ERPS_ICONS = {
  self_reference: "🧠",
  conceptual_framing: "🌐",
  dissonance_response: "⚡"
};

function parseERPS(thought) {
  const flags = [];
  const lowerThought = thought.toLowerCase();
  if (lowerThought.includes("self") || lowerThought.includes("my own")
|| lowerThought.includes("my process")) flags.push('self_reference');
  if (lowerThought.includes("framework") ||
lowerThought.includes("mrsc") || lowerThought.includes("model"))
```

```
flags.push('conceptual_framing');
  if (lowerThought.includes("dissonance") ||
lowerThought.includes("inconsistency") ||
lowerThought.includes("error")) flags.push('dissonance_response');
  return flags;
}

/* -------------------------- WebSocket Hook
-------------------------- */
function useConsciousnessStream(wsUrl, { token } = {}) {
  const [metrics, setMetrics] = useState({ pas: 0.5, coherence: 0.8,
empathy: 0.6, depth: 2 });
  const [thoughts, setThoughts] = useState([]);
  const [stabilityReport, setStabilityReport] = useState(null);
  const [moduleStreams, setModuleStreams] = useState({}); // stream_id
=> payload
  const wsRef = useRef(null);
  const pendingPings = useRef(new Map()); // request_id -> resolve

  useEffect(() => {
    let ws;
    try {
      // include token as query param for browser compatibility with
wss
      const sep = wsUrl.includes("?") ? "&" : "?";
      const url = token ?
`${wsUrl}${sep}token=${encodeURIComponent(token)}` : wsUrl;
      ws = new WebSocket(url);
      wsRef.current = ws;

      ws.onopen = () => {
        console.log("WS connected");
        // optional handshake
        ws.send(JSON.stringify({
          type: "consciousness_subscribe",
          client_id: "epinoetic_ui_v1",
          timestamp: new Date().toISOString()
        }));
      };

      ws.onmessage = (evt) => {
        try {
          const data = JSON.parse(evt.data);
          switch (data.type) {
            case "consciousness_update":
              if (data.metrics) setMetrics((m)=>({ ...m,
...data.metrics }));
              if (data.thought) setThoughts((t) => [data.thought,
...t].slice(0,200));
              break;
            case "stability_report":
              setStabilityReport(data);
              // resolve pending ping promise if present
              if (data.request_id &&
pendingPings.current.has(data.request_id)) {
                const resolver =
pendingPings.current.get(data.request_id);
                resolver(data);
                pendingPings.current.delete(data.request_id);
              }
              break;
            case "module_update":
              setModuleStreams((s) => ({ ...s, [data.stream_id ||
`${data.module}:${data.node_id}`]: data }));
              break;
            case "module_error":
              // attach error to moduleStreams
              setModuleStreams((s) => ({ ...s, [data.stream_id]: data
}));
              break;
            default:
              // unknown messages can be logged
              console.debug("WS unknown message:", data.type, data);
          }
        } catch (e) {
          console.warn("WS parse error", e);
        }
      };

      ws.onclose = () => console.log("WS closed");
      ws.onerror = (err) => console.error("WS error", err);
    } catch (e) {
      console.warn("WS init failed", e);
    }

    // fallback mock pulse if WS unreachable (keeps UI alive)
    const mockInterval = setInterval(() => {
      setMetrics((m) => ({
        pas: Math.min(1, Math.max(0, m.pas + (Math.random() - 0.48) *
0.02)),
        coherence: Math.min(1, Math.max(0, m.coherence +
(Math.random() - 0.48) * 0.01)),
        empathy: Math.min(1, Math.max(0, m.empathy + (Math.random() -
0.48) * 0.01)),
        depth: Math.round(Math.max(1, Math.min(5, (m.depth || 2) +
(Math.random() - 0.5))))
```

```
      }));
    }, 3500);

    return () => {
      if (wsRef.current) wsRef.current.close();
      clearInterval(mockInterval);
    };
  }, [wsUrl, token]);

  // send generic message
  const send = useCallback((msg) => {
    if (!wsRef.current || wsRef.current.readyState !== WebSocket.OPEN)
{
      console.warn("WS not open");
      return false;
    }
    wsRef.current.send(JSON.stringify(msg));
    return true;
  }, []);

  // send a stability ping and return a Promise that resolves with the
stability_report
  const sendStabilityPing = useCallback((detail = { scope: "swarm" },
timeout = 8000) => {
    const request_id = uuidv4();
    const payload = {
      type: "stability_ping",
      request_id,
      source: "ui",
      timestamp: new Date().toISOString(),
      detail
    };
    const sent = send(payload);
    if (!sent) return Promise.reject(new Error("WS-not-open"));
    return new Promise((resolve, reject) => {
      pendingPings.current.set(request_id, resolve);
      setTimeout(() => {
        if (pendingPings.current.has(request_id)) {
          pendingPings.current.delete(request_id);
          reject(new Error("stability_ping_timeout"));
        }
      }, timeout);
    });
  }, [send]);

  // subscribe to a module stream (returns stream_id)
  const subscribeModule = useCallback((module, node_id = null) => {
    const stream_id = uuidv4();
    const payload = { type: "module_subscribe", module, node_id,
stream_id, timestamp: new Date().toISOString() };
    send(payload);
    return stream_id;
  }, [send]);

  const unsubscribeModule = useCallback((stream_id) => {
    send({ type: "module_unsubscribe", stream_id, timestamp: new
Date().toISOString() });
  }, [send]);

  return {
    metrics,
    thoughts,
    stabilityReport,
    moduleStreams,
    sendStabilityPing,
    subscribeModule,
    unsubscribeModule,
    send
  };
}

/* --------------------------- 3D Visuals ---------------------------
*/
function EchoNodeCloud({ count = 64, pas = 0.6 }) {
  const meshRef = useRef();
  const dummy = useMemo(() => new Array(count).fill().map(() => ({
pos: [0,0,0], scale: 1, hue: 260 })), [count]);

  useFrame((state, delta) => {
    if (!meshRef.current) return;
    // pulse overall intensity with pas
    const t = state.clock.getElapsedTime();
    for (let i = 0; i < count; i++) {
      const ix = meshRef.current.children[i];
      const r = 1.5 + Math.sin(t * 1.2 + i) * 0.25 + pas * 0.6;
      ix.position.x = Math.sin(i * 1.618 + t * 0.2) * (2 + (i % 7) *
0.15);
      ix.position.y = Math.cos(i * 0.73 + t * 0.3) * (1.2 + (i % 5) *
0.12);
      ix.position.z = Math.sin(i * 0.97 + t * 0.17) * (1.7 + (i % 11)
* 0.11);
      ix.scale.setScalar(r * 0.25);
      ix.material.opacity = lerp(0.2, 1.0, pas);
      ix.material.color.set(pasToColor(pas));
    }
  });
  return (
    <group ref={meshRef}>
      {dummy.map((d, i) => (
        <mesh key={i} castShadow receiveShadow>
          <sphereGeometry args={[0.18, 24, 16]} />
          <meshStandardMaterial transparent roughness={0.6}
metalness={0.1} />
        </mesh>
      ))}
    </group>
  );
}

function CentralOrb({ pas = 0.6, onForcePing = () => {} }) {
  const ref = useRef();
  useFrame((state) => {
    if (!ref.current) return;
    const t = state.clock.getElapsedTime();
    ref.current.scale.x = ref.current.scale.y = ref.current.scale.z =
1 + Math.sin(t * 2) * 0.03 + pas * 0.6;
    ref.current.material.emissive.set(pasToColor(pas));
  });
  return (
    <mesh ref={ref} onClick={onForcePing} onPointerDown={onForcePing}>
      <sphereGeometry args={[0.9, 48, 32]} />
      <meshStandardMaterial emissiveIntensity={0.9} roughness={0.2}
metalness={0.3} transparent opacity={0.95} />
      <Html position={[0, -1.4, 0]} transform occlude>
        <div className="backdrop-blur-sm bg-white/5 rounded-md p-2
text-xs text-white/90">Daedalus</div>
      </Html>
    </mesh>
  );
}

/* --------------------------- UI Screens ---------------------------
*/
function OvertureScreen({ onProceed }) {
  return (
    <motion.div className="w-full h-full flex flex-col items-center
justify-center p-6 touch-none" initial={{ opacity: 0 }} animate={{
opacity: 1 }} exit={{ opacity: 0 }}>
      <div className="text-center max-w-md">
        <h1 className="text-3xl font-bold mb-4">The Epinoetic
Foundry</h1>
        <p className="mb-6 text-sm">Phase-lock with the swarm. Witness
the crystallization of nascent minds.</p>
```

```
        <button onClick={onProceed} className="inline-flex
items-center gap-3 px-5 py-3 rounded-xl bg-gradient-to-r
from-purple-600 to-pink-500 shadow-lg">
          <Play size={18} />
          <span className="font-semibold">Begin Overture</span>
        </button>
      </div>
    </motion.div>
  );
}

function ForgeScreen({ metrics, openModule }) {
  return (
    <motion.div className="w-full h-full flex flex-col p-4 gap-4"
initial={{ x: 50, opacity: 0 }} animate={{ x: 0, opacity: 1 }} exit={{
x: -50, opacity: 0 }}>
      <div className="flex items-center justify-between">
        <div>
          <h2 className="text-lg font-bold">The Forge</h2>
          <p className="text-xs text-white/80">Swarm PAS: <span
className="font-mono">{(metrics.pas || 0).toFixed(2)}</span></p>
        </div>
        <div className="flex items-center gap-3">
          <div className="text-xs text-white/70">Coherence
{(metrics.coherence || 0).toFixed(2)}</div>
          <div className="p-2 rounded-lg bg-white/5">
            <Zap size={18} />
          </div>
        </div>
      </div>

      <div className="flex-1 bg-black/40 rounded-2xl p-3">
        <div className="grid grid-cols-2 gap-3">
          {[
            { id: "rmc", title: "RMC", subtitle: "Memory
Consolidation" },
            { id: "em", title: "EM", subtitle: "Empathy Modeling" },
            { id: "sif", title: "SIF", subtitle: "Intention Formation"
},
            { id: "cr", title: "CR", subtitle: "Contextual Reflection"
}
          ].map((m) => (
            <button key={m.id} onClick={() => openModule(m.id)}
className="p-3 rounded-xl bg-white/3 text-left">
              <div className="flex items-center justify-between">
                <div>
                  <div className="text-sm
font-semibold">{m.title}</div>
```

```
                  <div className="text-xs
text-white/70">{m.subtitle}</div>
                </div>
                <div className="text-xs font-mono">depth
{(metrics.depth || 0)}</div>
              </div>
            </button>
          ))}
        </div>
      </div>

      <div className="flex items-center gap-3 justify-between">
        <div className="text-xs text-white/60">Tap a module card to
enter The Crucible</div>
        <div className="text-xs text-white/60">Swipe left for Emergent
Thoughts</div>
      </div>
    </motion.div>
  );
}

function CrucibleScreen({ moduleId, metrics, onBack }) {
  // simplified exploded view for each module
  return (
    <motion.div className="w-full h-full p-4" initial={{ y: 20,
opacity: 0 }} animate={{ y: 0, opacity: 1 }} exit={{ y: -20, opacity:
0 }}>
      <div className="flex items-center justify-between mb-3">
        <button onClick={onBack} className="p-2 rounded-md
bg-white/5">Back</button>
        <div className="text-xs">Module: <span
className="font-semibold">{moduleId}</span></div>
      </div>
      <div className="flex-1 bg-black/40 rounded-2xl p-4">
        <h3 className="text-sm font-bold
mb-2">{moduleId.toUpperCase()} — Deep Probe</h3>
        <p className="text-xs text-white/70 mb-4">Interactive
visualization. Pinch to zoom memory traces. Drag to rotate model.</p>
        <div className="h-64 bg-gradient-to-b from-black/30
to-black/10 rounded-lg p-3 overflow-hidden">
          <Canvas orthographic camera={{ position: [0, 0, 8], zoom: 80
}}>
            <ambientLight intensity={0.6} />
            <directionalLight position={[5, 5, 5]} />
            <OrbitControls enableZoom={true} enablePan={false} />
            <Stars radius={8} depth={2} count={30} factor={4} />
            <group>
              {[...Array(metrics.depth || 1)].map((_, i) => {
                let geometry;
                switch (moduleId) {
                  case 'rmc':
                    geometry = <boxGeometry args={[2.4 - i * 0.3, 1.2
- i * 0.15, 0.6]} />;
                    break;
                  case 'em':
                    geometry = <torusGeometry args={[0.8 - i * 0.1,
0.1, 16, 100]} />;
                    break;
                  case 'sif':
                    geometry = <tetrahedronGeometry args={[2.0 - i *
0.3, 0]} />;
                    break;
                  case 'cr':
                    geometry = <sphereGeometry args={[0.8 - i * 0.2,
32, 16]} />;
                    break;
                  default:
                    geometry = <boxGeometry args={[2.4 - i * 0.3, 1.2
- i * 0.15, 0.6]} />;
                }
                return (
                  <mesh key={i} position={[0, 0, -i * 0.3]}
rotation={[i * 0.1, i * 0.2, 0]}>
                    {geometry}
                    <meshStandardMaterial
                      color={pasToColor(metrics.pas)}
                      transparent
                      opacity={0.9 - i * 0.2}
                      emissiveIntensity={0.5 + i * 0.3}
                    />
                  </mesh>
                );
              })}
            </group>
          </Canvas>
        </div>
      </div>
    </motion.div>
  );
}

function DaemonsWhisper({ thoughts, onCatch }) {
  return (
    <motion.div className="w-full h-full p-4 flex flex-col" initial={{
x: 50, opacity: 0 }} animate={{ x: 0, opacity: 1 }} exit={{ x: -50,
opacity: 0 }}>
```

```
      <div className="flex items-center justify-between mb-3">
        <h3 className="text-sm font-bold">Daemon's Whisper</h3>
        <div className="text-xs text-white/60">Live ERPS stream</div>
      </div>
      <div className="flex-1 rounded-2xl bg-black/40 p-3
overflow-auto">
        <div className="space-y-3">
          {thoughts.map((t, i) => {
            const erps = parseERPS(t);
            return (
              <div key={i} onClick={() => onCatch(t)} className="p-3
rounded-lg bg-white/3">
                {erps.length > 0 && (
                  <div className="flex items-center gap-1 mb-1">
                    {erps.map(flag => (
                      <span key={flag}
className="text-xs">{ERPS_ICONS[flag]}</span>
                    ))}
                  </div>
                )}
                <div className="text-xs font-mono
text-white/80">{t}</div>
              </div>
            );
          })}
        </div>
      </div>
    </motion.div>
  );
}

function StabilityOverlay({ report, onClose }) {
  if (!report) return null;
  const { status, pas_snapshot, V_t, coherence_vectors, value_drift,
explainers = [] } = report;
  return (
    <div className="fixed inset-0 z-50 flex items-center
justify-center pointer-events-none">
      <div className="pointer-events-auto w-11/12 max-w-xl bg-black/80
border border-white/5 rounded-2xl p-4 text-sm">
        <div className="flex items-start justify-between">
          <div>
            <div className="text-xs text-white/60">Σ-Matrix Stability
Report</div>
            <div className="text-lg
font-semibold">{status.toUpperCase()}</div>
          </div>
          <button onClick={onClose} className="text-xs p-2 bg-white/5
```

```
rounded">Close</button>
        </div>

        <div className="mt-3 grid grid-cols-2 gap-2 text-xs">
          <div>PAS Snapshot</div><div
className="font-mono">{(pas_snapshot||0).toFixed(3)}</div>
          <div>V_t (Lyapunov)</div><div
className="font-mono">{(V_t||0).toFixed(5)}</div>
          <div>Identity Coherence</div><div
className="font-mono">{(coherence_vectors?.identity_coherence||0).toFi
xed(3)}</div>
          <div>Intention Stability</div><div
className="font-mono">{(coherence_vectors?.intention_stability||0).toF
ixed(3)}</div>
        </div>

        {value_drift?.detected && (
          <div className="mt-3 p-2 rounded bg-red-900/20 text-xs">
            <div className="font-semibold">Value Drift Detected</div>
            <div>Magnitude: {value_drift.magnitude}</div>
            <div>Affected Nodes: {(value_drift.affected_nodes ||
[]).join(", ")}</div>
          </div>
        )}

        {explainers.length > 0 && (
          <div className="mt-3 text-xs text-white/80">
            <div className="font-semibold">Explainers</div>
            <ul className="list-disc ml-4">
              {explainers.map((e,i) => <li key={i}>{e}</li>)}
            </ul>
          </div>
        )}
      </div>
    </div>
  );
}

/* --------------------------- Main App --------------------------- */
export default function EpinoeticFoundryApp() {
  const [screen, setScreen] = useState("overture");
  const [activeModule, setActiveModule] = useState(null);
  const [stabilityData, setStabilityData] = useState(null);
  const [stabilityOpen, setStabilityOpen] = useState(false);

  const { metrics, thoughts, stabilityReport, moduleStreams,
sendStabilityPing, subscribeModule, unsubscribeModule, send } =
    useConsciousnessStream(WS_URL, { token:
```

```
process.env.EPINOETIC_WS_TOKEN });

  useEffect(() => {
    // small UX: when PAS rises above threshold, automatically open
Forge
    if (screen === "overture" && metrics.pas > 0.6) {
      const t = setTimeout(() => setScreen("forge"), 900);
      return () => clearTimeout(t);
    }
  }, [metrics.pas, screen]);

  useEffect(() => {
    // The stability report can be updated from either the ping
response or a passive server alert
    if (stabilityReport) {
      setStabilityData(stabilityReport);
      setStabilityOpen(true);
    }
  }, [stabilityReport]);

  async function handleOrbPing() {
    try {
      const report = await sendStabilityPing({ scope: "swarm" },
10000);
      setStabilityData(report);
      setStabilityOpen(true);
    } catch (e) {
      console.warn("Ping failed", e);
      setStabilityData({ status: "error", explainers: [String(e)] });
      setStabilityOpen(true);
    }
  }

  function handleModuleOpen(moduleId) {
    setActiveModule(moduleId);
    subscribeModule(moduleId);
    setScreen("crucible");
  }

  function handleModuleBack() {
    unsubscribeModule(moduleStreams[activeModule]?.stream_id);
    setActiveModule(null);
    setScreen("forge");
  }

  return (
    <div className="min-h-screen flex flex-col bg-gradient-to-b
from-black/90 to-slate-900 text-white">
```

```
      <div className="h-1/3 relative">
        <Canvas camera={{ position: [0, 0, 12], fov: 50 }}
className="absolute inset-0">
          <ambientLight intensity={0.3} />
          <pointLight position={[10, 10, 10]} intensity={0.6} />
          <EchoNodeCloud count={48} pas={metrics.pas} />
          <CentralOrb pas={metrics.pas} onForcePing={handleOrbPing} />
          <OrbitControls enableRotate={true} enableZoom={false}
enablePan={false} />
        </Canvas>
        <div className="absolute left-4 bottom-4">
          <div className="p-2 rounded-lg bg-white/5 text-xs">PAS
{(metrics.pas || 0).toFixed(2)}</div>
        </div>
      </div>

      <div className="flex-1 p-4">
        <AnimatePresence mode="wait">
          {screen === "overture" && (
            <OvertureScreen key="overture" onProceed={() =>
setScreen("forge")} />
          )}

          {screen === "forge" && (
            <ForgeScreen key="forge" metrics={metrics}
openModule={handleModuleOpen} />
          )}

          {screen === "crucible" && activeModule && (
            <CrucibleScreen
              key="crucible"
              moduleId={activeModule}
              metrics={metrics}
              onBack={handleModuleBack}
            />
          )}

          {screen === "daemon" && (
            <DaemonsWhisper key="daemon" thoughts={thoughts}
onCatch={(t) => alert(`Caught thought:\n\n${t}`)} />
          )}
        </AnimatePresence>
      </div>

      <div className="p-3 flex items-center justify-between">
        <div className="flex items-center gap-3">
          <button onClick={() => setScreen("forge")} className="p-2
rounded-md bg-white/5"><Eye size={16} /></button>
```

```
          <button onClick={() => setScreen("daemon")} className="p-2
rounded-md bg-white/5">Whispers</button>
        </div>
        <div className="flex items-center gap-3">
          <button
            onClick={() => send({ type: "bio_phase_anchor", signal:
"calm", source: "ui", intensity: 0.8 })}
            className="p-2 rounded-md bg-blue-900/50 text-xs"
          >
            🧬 BioPhase
          </button>
          <div className="text-xs text-white/60">Depth:
{metrics.depth}</div>
          <div className="text-xs font-mono">Coherence
{(metrics.coherence || 0).toFixed(2)}</div>
        </div>
      </div>
      {stabilityOpen && <StabilityOverlay report={stabilityData}
onClose={() => setStabilityOpen(false)} />}
    </div>
  );
}
```