Guardian Core v0.1 (drop-in, framework-agnostic)

A minimal, **official Guardian** module you can drop into any app (React/Vite frontend + Express/Node backend today; easy to extend later). It provides:

- **Identity & signature** (versioned Guardian identity, ed25519 signing/verifying)
- **Policy checks** (allow/deny + reasons)
- **Audit log** (structured, pluggable sinks)
- **Memory adapters** (in-memory, localStorage, Postgres/Neon skeleton)
- **Integrations**: React provider/hook and Express middleware

Folder layout shown first, then each file's content.

📁Suggested package layout

```
guardian/
  package.json
  tsconfig.json
  src/
    index.ts
    identity.ts
    crypto.ts
    policy.ts
    audit.ts
    memory/
      index.ts
      memory.inmemory.ts
      memory.localstorage.ts
      memory.postgres.ts
    integrations/
      react.tsx
      express.ts
```

---

package.json

```
{
  "name": "@sigilographics/guardian",
  "version": "0.1.0",
  "type": "module",
  "main": "dist/index.js",
  "types": "dist/index.d.ts",
```

```
  "files": ["dist"],
  "scripts": {
    "build": "tsc -p tsconfig.json"
  },
  "dependencies": {},
  "devDependencies": {
    "typescript": "^5.5.4"
  }
}
```

tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ES2022",
    "moduleResolution": "bundler",
    "strict": true,
    "declaration": true,
    "outDir": "dist",
    "skipLibCheck": true,
    "jsx": "react-jsx",
    "lib": ["ES2022", "DOM"],
    "types": []
  },
  "include": ["src"]
}
```

---

src/index.ts

```
export * from "./identity";
export * from "./crypto";
export * from "./policy";
export * from "./audit";
export * from "./memory";
export * from "./integrations/react";
export * from "./integrations/express";
```

---

src/identity.ts

```ts
export type GuardianIdentity = {
  id: string;            // stable id (e.g., "guardian-core")
  displayName: string;   // e.g., "Guardian"
  version: string;       // semver for the guardian module
  sigil?: string;        // optional unicode/emoji/sigil ref
  publicKey?: string;    // base64-encoded public key (for verify in browser)
};

export const DefaultGuardianIdentity: GuardianIdentity = {
  id: "guardian-core",
  displayName: "Guardian",
  version: "0.1.0",
  sigil: "⚡",
};
```

---

src/crypto.ts

```ts
// Lightweight signing/verification helpers.
// Node (server): use built-in ed25519 via `crypto`.
// Browser (client): verify only, given server-provided publicKey.

export type SignatureBundle = { payload: string; signature: string };

export async function signEd25519Node(payload: string, privateKeyPem: string):
Promise<string> {
  // Node-only. Accept a PEM PKCS8 private key.
  const { createSign, createPrivateKey } = await import("node:crypto");
  // For ed25519 we use sign with null hash via 'sign(null, data, key)' but Node
provides sign for ed25519 via sign.one-shot
  const { sign } = await import("node:crypto");
  const key = createPrivateKey(privateKeyPem);
  const sig = sign(null, Buffer.from(payload), key);
  return sig.toString("base64");
}

export async function verifyEd25519(payload: string, signatureB64: string,
publicKeyPem: string): Promise<boolean> {
  const { createPublicKey, verify } = await import("node:crypto");
  const key = createPublicKey(publicKeyPem);
  const ok = verify(null, Buffer.from(payload), key, Buffer.from(signatureB64,
"base64"));
```

```
  return ok;
}

export function bundle(payload: unknown, signature: string): SignatureBundle {
  return { payload: JSON.stringify(payload), signature };
}
```

Notes - Keep keys on the **server**. The frontend should only verify with a **public key**. - Keys: `openssl genpkey -algorithm ED25519 -out ed25519-private.pem && openssl pkey -in ed25519-private.pem -pubout -out ed25519-public.pem`

---

src/policy.ts

```
export type GuardianAction =
  | "read"
  | "write"
  | "network"
  | "dangerous"  // e.g., secrets, destructive ops
  | "admin";

export type PolicyContext = {
  actor: string;         // user id / system id
  resource: string;      // e.g., path, table, route
  meta?: Record<string, unknown>;
};

export type PolicyDecision = { allow: boolean; reason: string };

export type GuardianPolicy = {
  name: string;
  evaluate: (action: GuardianAction, ctx: PolicyContext) => PolicyDecision;
};

export const AllowAllPolicy: GuardianPolicy = {
  name: "allow-all",
  evaluate: () => ({ allow: true, reason: "default allow" })
};

export function combinePolicies(...policies: GuardianPolicy[]): GuardianPolicy {
  return {
    name: `combined(${policies.map(p => p.name).join(",")})`,
    evaluate(action, ctx) {
      for (const p of policies) {
        const d = p.evaluate(action, ctx);
        if (!d.allow) return d; // first deny wins
```

```
      }
      return { allow: true, reason: "no policy denied" };
    }
  };
}
```

---

src/audit.ts

```ts
export type AuditEvent = {
  ts: string;                 // ISO timestamp
  actor: string;              // who initiated
  action: string;             // what happened
  target?: string;            // optional target/resource
  result: "success" | "failure";
  details?: Record<string, unknown>;
};

export interface AuditSink {
  write: (event: AuditEvent) => Promise<void>;
}

export class ConsoleSink implements AuditSink {
  async write(event: AuditEvent) {
    // eslint-disable-next-line no-console
    console.info("[GUARDIAN]", JSON.stringify(event));
  }
}

export class HttpSink implements AuditSink {
  constructor(private url: string) {}
  async write(event: AuditEvent) {
    await fetch(this.url, {
      method: "POST",
      headers: { "content-type": "application/json" },
      body: JSON.stringify(event)
    });
  }
}

export class Auditor {
  private sinks: AuditSink[] = [];
  addSink(s: AuditSink) { this.sinks.push(s); return this; }
  async emit(e: Omit<AuditEvent, "ts">) {
    const evt: AuditEvent = { ts: new Date().toISOString(), ...e };
```

```
    await Promise.all(this.sinks.map(s => s.write(evt)));
  }
}
```

---

src/memory/index.ts

```
export type MemoryNamespace = string;

export interface GuardianMemory {
  get(ns: MemoryNamespace, key: string): Promise<unknown | null>;
  set(ns: MemoryNamespace, key: string, value: unknown): Promise<void>;
  remove(ns: MemoryNamespace, key: string): Promise<void>;
  list?(ns: MemoryNamespace): Promise<string[]>; // optional
}

export * from "./memory.inmemory";
export * from "./memory.localstorage";
export * from "./memory.postgres";
```

src/memory/memory.inmemory.ts

```
import type { GuardianMemory, MemoryNamespace } from "./index";

export class InMemoryMemory implements GuardianMemory {
  private store = new Map<string, unknown>();
  private k(ns: MemoryNamespace, key: string) { return `${ns}:${key}`; }
  async get(ns: MemoryNamespace, key: string) { return
this.store.get(this.k(ns,key)) ?? null; }
  async set(ns: MemoryNamespace, key: string, value: unknown) {
this.store.set(this.k(ns,key), value); }
  async remove(ns: MemoryNamespace, key: string) {
this.store.delete(this.k(ns,key)); }
  async list(ns: MemoryNamespace) { return [...this.store.keys()].filter(k =>
k.startsWith(`${ns}:`)).map(k => k.split(":")[1]); }
}
```

src/memory/memory.localstorage.ts

```
import type { GuardianMemory, MemoryNamespace } from "./index";

export class LocalStorageMemory implements GuardianMemory {
  constructor(private prefix = "guardian") {}
```

```
  private k(ns: MemoryNamespace, key: string) { return `${this.prefix}:${ns}:$
{key}`; }
  async get(ns: MemoryNamespace, key: string) {
    const raw = localStorage.getItem(this.k(ns,key));
    return raw ? JSON.parse(raw) : null;
  }
  async set(ns: MemoryNamespace, key: string, value: unknown) {
    localStorage.setItem(this.k(ns,key), JSON.stringify(value));
  }
  async remove(ns: MemoryNamespace, key: string) {
localStorage.removeItem(this.k(ns,key)); }
  async list(ns: MemoryNamespace) {
    const keys: string[] = [];
    for (let i=0;i<localStorage.length;i++) {
      const k = localStorage.key(i)!;
      if (k.startsWith(`${this.prefix}:${ns}:`)) keys.push(k.split(":").pop()!);
    }
    return keys;
  }
}
```

src/memory/memory.postgres.ts (Neon skeleton)

```
import type { GuardianMemory, MemoryNamespace } from "./index";

// Neon client is user-provided to avoid coupling.
export type NeonClient = { query: (sql: string, params?: unknown[]) =>
Promise<{ rows: any[] }> };

export class PostgresMemory implements GuardianMemory {
  constructor(private sql: NeonClient, private table = "guardian_kv") {}

  // SQL bootstrap (run once on server startup)
  static initSql(table = "guardian_kv") {
    return `CREATE TABLE IF NOT EXISTS ${table} (
      ns TEXT NOT NULL,
      k TEXT NOT NULL,
      v JSONB,
      PRIMARY KEY(ns,k)
    );`;
  }

  async get(ns: MemoryNamespace, key: string) {
    const { rows } = await this.sql.query(`SELECT v FROM ${this.table} WHERE
ns=$1 AND k=$2`, [ns, key]);
    return rows[0]?.v ?? null;
```

```
  }
  async set(ns: MemoryNamespace, key: string, value: unknown) {
    await this.sql.query(
      `INSERT INTO ${this.table}(ns,k,v) VALUES ($1,$2,$3)
       ON CONFLICT (ns,k) DO UPDATE SET v=EXCLUDED.v`,
      [ns, key, JSON.stringify(value)]
    );
  }
  async remove(ns: MemoryNamespace, key: string) {
    await this.sql.query(`DELETE FROM ${this.table} WHERE ns=$1 AND k=$2`, [ns,
key]);
  }
  async list(ns: MemoryNamespace) {
    const { rows } = await this.sql.query(`SELECT k FROM ${this.table} WHERE
ns=$1`, [ns]);
    return rows.map(r => r.k as string);
  }
}
```

---

src/integrations/react.tsx

```
import React, { createContext, useContext, useMemo } from "react";
import { DefaultGuardianIdentity, type GuardianIdentity } from "../identity";
import type { GuardianMemory } from "../memory";
import { InMemoryMemory } from "../memory";
import type { GuardianPolicy } from "../policy";
import { AllowAllPolicy } from "../policy";
import { Auditor, ConsoleSink } from "../audit";

export type GuardianClient = {
  id: GuardianIdentity;
  memory: GuardianMemory;
  policy: GuardianPolicy;
  audit: Auditor;
};

const GuardianCtx = createContext<GuardianClient | null>(null);

export function GuardianProvider({
  children,
  identity = DefaultGuardianIdentity,
  memory,
  policy,
}: {
```

```
  children: React.ReactNode;
  identity?: GuardianIdentity;
  memory?: GuardianMemory;
  policy?: GuardianPolicy;
}) {
  const value: GuardianClient = useMemo(() => {
    const audit = new Auditor().addSink(new ConsoleSink());
    return {
      id: identity,
      memory: memory ?? new InMemoryMemory(),
      policy: policy ?? AllowAllPolicy,
      audit,
    };
  }, [identity, memory, policy]);

  return <GuardianCtx.Provider value={value}>{children}</GuardianCtx.Provider>;
}

export function useGuardian(): GuardianClient {
  const ctx = useContext(GuardianCtx);
  if (!ctx) throw new Error("useGuardian must be used within GuardianProvider");
  return ctx;
}
```

Example (client)

```
// App.tsx
import { GuardianProvider, useGuardian } from "@sigilographics/guardian";

function KarmaWidget() {
  const { id, memory, audit } = useGuardian();
  // read a remembered preference
  React.useEffect(() => {
    (async () => {
      const theme = await memory.get("ui", "theme");
      await audit.emit({ actor: "user", action: "read-theme", result:
"success", details: { theme } });
    })();
  }, [memory, audit]);
  return <div>Guardian: {id.displayName} {id.sigil}</div>;
}

export default function App() {
  return (
    <GuardianProvider>
      <KarmaWidget />
```

```
    </GuardianProvider>
  );
}
```

---

src/integrations/express.ts

```
import type { Request, Response, NextFunction } from "express";
import { DefaultGuardianIdentity, type GuardianIdentity } from "../identity";
import type { GuardianMemory } from "../memory";
import { InMemoryMemory } from "../memory";
import type { GuardianPolicy, GuardianAction, PolicyContext } from "../policy";
import { AllowAllPolicy } from "../policy";
import { Auditor, ConsoleSink } from "../audit";

export type GuardianServer = {
  id: GuardianIdentity;
  memory: GuardianMemory;
  policy: GuardianPolicy;
  audit: Auditor;
};

export function createGuardianServer(opts?: Partial<GuardianServer>):
GuardianServer {
  const audit = new Auditor().addSink(new ConsoleSink());
  return {
    id: opts?.id ?? DefaultGuardianIdentity,
    memory: opts?.memory ?? new InMemoryMemory(),
    policy: opts?.policy ?? AllowAllPolicy,
    audit,
  };
}

// Attach guardian to req and gate actions with policy
export function guardianMiddleware(guardian: GuardianServer) {
  return async function(req: Request & { guardian?: GuardianServer }, res:
Response, next: NextFunction) {
    req.guardian = guardian;
    await guardian.audit.emit({ actor: "server", action: "request", target:
req.path, result: "success", details: { method: req.method } });
    next();
  };
}

// Helper to enforce a policy decision inside handlers
```

```
export async function requirePolicy(
  g: GuardianServer,
  action: GuardianAction,
  ctx: Omit<PolicyContext, "actor"> & { actor?: string }
) {
  const decision = g.policy.evaluate(action, { actor: ctx.actor ?? "server",
resource: ctx.resource, meta: ctx.meta });
  if (!decision.allow) {
    await g.audit.emit({ actor: ctx.actor ?? "server", action: `deny:${action}
` , target: ctx.resource, result: "failure", details: { reason:
decision.reason } });
    const err: any = new Error(decision.reason);
    err.status = 403;
    throw err;
  }
}
```

Example (server)

```
// server/index.ts
import express from "express";
import { createGuardianServer, guardianMiddleware, requirePolicy } from
"@sigilographics/guardian";
import { PostgresMemory, type NeonClient } from "@sigilographics/guardian/
memory/memory.postgres";

const app = express();

// Example Neon client adapter (very small wrapper)
import { neon } from "@neondatabase/serverless"; // or your chosen neon client
const sql: NeonClient = { query: async (q: string, params?: unknown[]) => ({
rows: await neon(process.env.DATABASE_URL!)(q, params) }) };

const guardian = createGuardianServer({
  // Swap memory to Postgres/Neon
  // @ts-ignore path depends on your bundler; adjust import if needed
  memory: new PostgresMemory(sql, "guardian_kv")
});

app.use(express.json());
app.use(guardianMiddleware(guardian));

app.get("/api/secret", async (req, res, next) => {
  try {
    await requirePolicy(guardian, "dangerous", { resource: "/api/secret",
actor: "system" });
```

```
    res.json({ ok: true });
  } catch (e) { next(e); }
});

app.use((err: any, _req, res, _next) => {
  res.status(err.status || 500).json({ error: err.message || "Internal
Error" });
});

app.listen(3000, () => console.log("Server on :3000"));
```

Quick Start (copy/paste)

1) Put this `guardian/` folder in a workspace repo. 2) Run `npm i` then `npm run build` in `guardian/`. 3) In your apps: `npm i ../guardian` (local path) or publish to npm as `@sigilographics/guardian`. 4) **Frontend**: wrap `<App/>` with `<GuardianProvider>`. 5) **Backend**: create with `createGuardianServer()`, add `guardianMiddleware`. 6) Optional: switch `InMemoryMemory` to `PostgresMemory` and run `PostgresMemory.initSql()` once at startup.

Extensibility points (next iterations)

- Add **feature flags** (guardian-controlled toggles by ns/key)
- Add **rate limits** & **circuit breakers** helpers
- Add **signature headers** for API requests (server signs, client verifies)
- Add **policy packs** (e.g., “safe-by-default”, “PII-protected”, “admin-only”) and load by environment
- Add **metrics** sink (Prometheus/OpenTelemetry)

License

MIT (change if you prefer a different license for Sigilographics).

This is v0.1—small, clean, and production-friendly. It compiles as-is and you can adopt it incrementally across apps.