PRIMEX AI — Professional Build Plan & Monorepo Scaffold v1

Owner: Tyler C. Hoag (Commander)
Mode: ABSOLUTE_BOOT | Commander-first | Deterministic
Goal: Finish PRIMEX AI (brain, clones, files, code, JSON, review) to Apple/OpenAI/Google-level polish.
Scope: Backend brain + clone system + config/data schemas + CLI/API layer. (UX layer to follow.)

---

1) Delivery Objectives (What “finished” means)

- **Monorepo** with clear packages, TypeScript-first, strict lint, tests, docs.
- **Kernel (brain)** that orchestrates clones via deterministic command loop.
- **Clone framework** with typed interfaces, base class, registry, and 10 clones implemented from spec (ARCHITECT, CORTEX, GHOSTLINE, GOODJEW, CENTURION, OVERSEER, SCRIBE, MINT, VAULT, GLITCH).
- **Data contracts**: Zod schemas + JSON Schemas for kernel, clones, directives, audit logs, sessions, policies.
- **Config set**: machine-readable `primex.kernel.json`, `clones/*.json` with validated content.
- **CLI** to issue directives, inspect audit logs, and start runtime.
- **API server** (tRPC/Express) stub for future UX connectivity.
- **Security & Compliance**: Role/permission checks, immutable logs, content policy hooks.
- **CI-quality standards**: formatting, linting, type-check, unit tests, release notes template.

---

2) Repository Structure (Monorepo)

```
primex/
├─ README.md
├─ LICENSE
├─ CODE_OF_CONDUCT.md
├─ SECURITY.md
├─ CONTRIBUTING.md
├─ CHANGELOG.md
├─ package.json                # workspaces root
├─ pnpm-workspace.yaml         # or npm/yarn; default pnpm
├─ tsconfig.base.json
├─ .editorconfig
├─ .gitignore
├─ .prettierignore
├─ .prettierrc.json
├─ .eslintrc.cjs
```

```
├─ .env.example
├─ docs/
│  ├─ ARCHITECTURE.md
│  ├─ RUNTIME.md
│  ├─ DATA_CONTRACTS.md
│  ├─ CLONES.md
│  └─ OPERATIONS.md
├─ configs/
│  ├─ primex.kernel.json        # kernel loyalty/policy/runtime config
│  ├─ primex.policy.json        # policy ruleset
│  ├─ clones/
│  │  ├─ ARCHITECT.json
│  │  ├─ CORTEX.json
│  │  ├─ GHOSTLINE.json
│  │  ├─ GOODJEW.json
│  │  ├─ CENTURION.json
│  │  ├─ OVERSEER.json
│  │  ├─ SCRIBE.json
│  │  ├─ MINT.json
│  │  ├─ VAULT.json
│  │  └─ GLITCH.json
│  └─ schemas/                  # JSON Schemas (generated from Zod)
│     ├─ kernel.schema.json
│     ├─ clone.schema.json
│     ├─ directive.schema.json
│     ├─ audit.schema.json
│     └─ policy.schema.json
├─ packages/
│  ├─ core/
│  │  ├─ package.json
│  │  ├─ src/
│  │  │  ├─ types.ts
│  │  │  ├─ schemas.ts
│  │  │  ├─ kernel.ts
│  │  │  ├─ logger.ts
│  │  │  ├─ policy.ts
│  │  │  ├─ bus.ts
│  │  │  ├─ storage/
│  │  │  │  ├─ index.ts
│  │  │  │  ├─ memory.ts
│  │  │  │  └─ file.ts
│  │  │  └─ utils/
│  │  │     └─ ids.ts
│  ├─ clones/
│  │  ├─ package.json
│  │  ├─ src/
```

```
│  │  │  ├─ cloneBase.ts
│  │  │  ├─ registry.ts
│  │  │  └─ agents/
│  │  │     ├─ ARCHITECT.ts
│  │  │     ├─ CORTEX.ts
│  │  │     ├─ GHOSTLINE.ts
│  │  │     ├─ GOODJEW.ts
│  │  │     ├─ CENTURION.ts
│  │  │     ├─ OVERSEER.ts
│  │  │     ├─ SCRIBE.ts
│  │  │     ├─ MINT.ts
│  │  │     ├─ VAULT.ts
│  │  │     └─ GLITCH.ts
│  ├─ cli/
│  │  ├─ package.json
│  │  └─ src/index.ts
│  └─ server/
│     ├─ package.json
│     └─ src/index.ts
├─ tests/
│  ├─ vitest.config.ts
│  ├─ schemas.test.ts
│  ├─ kernel.test.ts
│  └─ clones.test.ts
└─ tools/
   └─ generate-json-schemas.ts  # zod-to-json-schema
```

---

3) Coding Standards

- **Language:** TypeScript (strict mode). Node 20 LTS. ESM modules.
- **Formatting:** Prettier enforced. ESLint (airbnb-ish) with TypeScript plugin.
- **Testing:** Vitest + ts-node + NYC coverage gate (>=90%).
- **Commit style:** Conventional Commits (feat/fix/docs/chore/refactor/test/build/ci).
- **Versioning:** SemVer. Release notes via `changesets` or `standard-version`.
- **Docs:** Typedoc for `packages/core` public API.

---

4) Data Contracts (Zod + JSON Schema)

Authoritative source is Zod in `packages/core/src/schemas.ts`; JSON Schemas generated to `configs/schemas/*`.

```typescript
// packages/core/src/types.ts
export type UUID = string;

export enum CloneId {
  ARCHITECT = 'ARCHITECT',
  CORTEX = 'CORTEX',
  GHOSTLINE = 'GHOSTLINE',
  GOODJEW = 'GOODJEW',
  CENTURION = 'CENTURION',
  OVERSEER = 'OVERSEER',
  SCRIBE = 'SCRIBE',
  MINT = 'MINT',
  VAULT = 'VAULT',
  GLITCH = 'GLITCH',
}

export type Priority = 'LOW' | 'MEDIUM' | 'HIGH' | 'CRITICAL';

export interface Directive {
  id: UUID;
  issuedBy: string; // Commander or system
  target: CloneId | 'ALL';
  priority: Priority;
  content: string; // natural language directive
  timestamp: string; // ISO8601
  metadata?: Record<string, unknown>;
}

export interface AuditEntry {
  id: UUID;
  directiveId: UUID;
  actor: CloneId | 'KERNEL';
  stage: 'RECEIVED' | 'VALIDATED' | 'DISPATCHED' | 'EXECUTED' | 'ERRORED';
  timestamp: string;
  notes?: string;
}

export interface CloneSpec {
  id: CloneId;
  role: string;
  personality: string;
  directives: string[];
  restricted_actions: string[];
}

export interface PolicyRule {
  id: string;
```

```
  description: string;
  allow: boolean;
  match: {
    target?: CloneId | 'ALL';
    priority?: Priority;
    keywords?: string[];
  };
}
```

```
// packages/core/src/schemas.ts
import { z } from 'zod';

export const uuid = z.string().uuid();
export const iso = z.string().datetime();

export const DirectiveSchema = z.object({
  id: uuid,
  issuedBy: z.string().min(1),
  target: z.union([z.enum([

'ARCHITECT','CORTEX','GHOSTLINE','GOODJEW','CENTURION','OVERSEER','SCRIBE','MINT','VAULT','GLITCH
  ]), z.literal('ALL')]),
  priority: z.enum(['LOW','MEDIUM','HIGH','CRITICAL']),
  content: z.string().min(1),
  timestamp: iso,
  metadata: z.record(z.unknown()).optional(),
});

export const AuditSchema = z.object({
  id: uuid,
  directiveId: uuid,
  actor: z.union([z.literal('KERNEL'), z.enum([

'ARCHITECT','CORTEX','GHOSTLINE','GOODJEW','CENTURION','OVERSEER','SCRIBE','MINT','VAULT','GLITCH
  ])]),
  stage: z.enum(['RECEIVED','VALIDATED','DISPATCHED','EXECUTED','ERRORED']),
  timestamp: iso,
  notes: z.string().optional(),
});

export const CloneSpecSchema = z.object({
  id:
z.enum(['ARCHITECT','CORTEX','GHOSTLINE','GOODJEW','CENTURION','OVERSEER','SCRIBE','MINT','VAULT'
  role: z.string().min(1),
  personality: z.string().min(1),
  directives: z.array(z.string()),
```

```
  restricted_actions: z.array(z.string()),
});

export const PolicyRuleSchema = z.object({
  id: z.string().min(1),
  description: z.string(),
  allow: z.boolean(),
  match: z.object({
    target:
z.union([z.enum(['ARCHITECT','CORTEX','GHOSTLINE','GOODJEW','CENTURION','OVERSEER','SCRIBE','MINT
z.literal('ALL')]).optional(),
    priority: z.enum(['LOW','MEDIUM','HIGH','CRITICAL']).optional(),
    keywords: z.array(z.string()).optional(),
  }),
});
```

---

5) Kernel & Runtime

```
// packages/core/src/logger.ts
export interface Logger {
  info: (...args: unknown[]) => void;
  warn: (...args: unknown[]) => void;
  error: (...args: unknown[]) => void;
}
export const logger: Logger = console;
```

```
// packages/core/src/bus.ts
import { EventEmitter } from 'node:events';
export const bus = new EventEmitter();
```

```
// packages/core/src/storage/index.ts
export interface StorageAdapter {
  writeAudit(entry: import('../types').AuditEntry): Promise<void>;
  listAudit(): Promise<import('../types').AuditEntry[]>;
}
```

```
// packages/core/src/storage/memory.ts
import { StorageAdapter } from './index';
import { AuditEntry } from '../types';
export class MemoryStorage implements StorageAdapter {
  private entries: AuditEntry[] = [];
```

```
  async writeAudit(entry: AuditEntry) { this.entries.push(entry); }
  async listAudit() { return [...this.entries]; }
}
```

```
// packages/core/src/policy.ts
import { PolicyRule } from './types';

export class PolicyEngine {
  constructor(private rules: PolicyRule[]) {}
  allow(target: string, priority: string, content: string) {
    for (const r of this.rules) {
      const targetMatch = !r.match.target || r.match.target === 'ALL' ||
r.match.target === target;
      const prioMatch = !r.match.priority || r.match.priority === priority;
      const kwMatch = !r.match.keywords || r.match.keywords.some(k =>
content.includes(k));
      if (targetMatch && prioMatch && kwMatch) return r.allow;
    }
    return true; // default allow
  }
}
```

```
// packages/core/src/kernel.ts
import { v4 as uuid } from 'uuid';
import { Directive, AuditEntry } from './types';
import { DirectiveSchema } from './schemas';
import { StorageAdapter } from './storage';
import { PolicyEngine } from './policy';
import { logger } from './logger';
import { bus } from './bus';

export interface Clone {
  id: string;
  handle(d: Directive): Promise<string>; // returns summary
}

export interface KernelDeps {
  storage: StorageAdapter;
  policy: PolicyEngine;
  registry: Map<string, Clone>;
}

export class Kernel {
  constructor(private deps: KernelDeps) {}
```

```
  private async audit(stage: AuditEntry['stage'], directive: Directive, actor:
string, notes?: string) {
    const entry: AuditEntry = {
      id: uuid(),
      directiveId: directive.id,
      actor: actor as any,
      stage,
      timestamp: new Date().toISOString(),
      notes,
    };
    await this.deps.storage.writeAudit(entry);
    bus.emit('audit', entry);
  }

  async dispatch(d: Directive) {
    await this.audit('RECEIVED', d, 'KERNEL');
    const parsed = DirectiveSchema.safeParse(d);
    if (!parsed.success) {
      await this.audit('ERRORED', d, 'KERNEL',
JSON.stringify(parsed.error.issues));
      throw new Error('Invalid directive');
    }
    await this.audit('VALIDATED', d, 'KERNEL');
    if (!this.deps.policy.allow(String(d.target), d.priority, d.content)) {
      await this.audit('ERRORED', d, 'KERNEL', 'Policy denied');
      throw new Error('Policy denied');
    }
    if (d.target === 'ALL') {
      for (const clone of this.deps.registry.values()) {
        await this.executeFor(clone, d);
      }
    } else {
      const clone = this.deps.registry.get(String(d.target));
      if (!clone) {
        await this.audit('ERRORED', d, 'KERNEL', 'Clone not found');
        throw new Error('Clone not found');
      }
      await this.executeFor(clone, d);
    }
  }

  private async executeFor(clone: Clone, d: Directive) {
    await this.audit('DISPATCHED', d, clone.id);
    const result = await clone.handle(d).catch(e => `ERROR: ${e?.message ??
'unknown'}`);
    await this.audit('EXECUTED', d, clone.id, result);
  }
}
```

6) Clone Framework & Implementations

```
// packages/clones/src/cloneBase.ts
import type { Directive } from '../../core/src/types';
import type { Cl
```