🕶️ Nightshift: Analyst

A narrative-driven terminal simulation about power, surveillance, and moral choice.

🧩 1. Concept Overview

Premise
You play as a night-shift systems analyst for a classified government division.
Your console is your world — an endless stream of logs, alerts, and messages.
One night, you intercept a message meant for someone else — the director, who shares your name.
It speaks of an imminent conspiracy.
From that moment, every keystroke becomes a choice between truth and survival.

Core Loop

Read → Decide → Act → Observe → Adapt.

Themes

- Bureaucracy vs. conscience
- Information as weapon
- Obedience, loyalty, and moral decay

⚙️ 2. System Architecture Overview

Frontend (SvelteKit)
⇅ REST / SSE / WebSockets
Backend (FastAPI)
⇅
PostgreSQL / Redis / S3

Layer	Tech Stack	Purpose
Frontend	SvelteKit + Tailwind + DaisyUI	Interactive terminal UI

Backend	FastAPI + SQLModel + Redis	Async API, live events, persistence
Database	PostgreSQL	Core data & player state
Cache/Jobs	Redis + Celery	Background task processing
Storage	S3 / MinIO	File & log storage
Auth	FastAPI Users (JWT/OAuth2)	Stateless authentication
Observability	Sentry + OpenTelemetry	Error tracking & tracing

---

🧠 3. Frontend System Design — *SvelteKit Stack*

🪄 Core Framework

SvelteKit

- Fast, file-based routing
- SSR/SSG-ready
- Ideal for dynamic CLI-like UIs with minimal boilerplate

🎨 Styling & UI

- **TailwindCSS** — utility-first, theme-ready styling
- **DaisyUI** — prebuilt dark/cyber themes
- **Heroicons / Lucide Icons** — vector status indicators
- **clsx** — conditional class management (active command states)

🧩 Logic & Validation

- **TypeScript** — strict typing for game events and commands
- **Zod** — schema validation synced with backend OpenAPI
- **Axios** or `fetch` — REST communication

- **TanStack Query (Svelte Query)** — real-time state + cache management
- **Day.js / date-fns** — timestamp and shift-time utilities

🧰 Developer Experience

- **PostCSS + Autoprefixer** — CSS transforms
- **Prettier + Tailwind plugin** — clean formatting
- **ESLint + eslint-plugin-svelte** — linting and type checks
- **Vitest** — fast unit testing
- **Playwright** — end-to-end tests for game flow

✨ Optional Enhancements

- **Motion One** — subtle CRT flicker and animation
- **Svelte Headless UI** — unstyled accessible components

🔧 4. Backend System Design — *FastAPI Stack*

⚙️ Framework

FastAPI

- Async-first, type-safe (Pydantic)
- Built-in OpenAPI → seamless TypeScript integration
- Ideal for REST + SSE APIs

🧩 Core Libraries

- **SQLAlchemy / SQLModel** — ORM for relational DBs
- **Alembic** — database migrations
- **Uvicorn** — ASGI server (with Gunicorn in prod)

- **httpx** — async HTTP client
- **loguru** — structured logging
- **python-dotenv** — env configuration
- **fastapi-users** — user/auth handling (optional)

🧱 Database & Storage

- **PostgreSQL** — persistent state (messages, sessions, endings)
- **Redis** — cache, rate limiting, live stream tracking
- **Celery / Dramatiq** — async background events
- **S3 / MinIO** — exported logs, saved playthroughs

🔐 Security & Auth

- **JWT/OAuth2** — stateless authentication
- **CORS** — secure frontend communication
- **HTTPS via Nginx/Cloudflare** — proxy and SSL termination
- **fastapi-limiter** — Redis-backed rate limiting

🧰 Developer Experience

- **pytest** — test framework
- **black + isort + ruff** — formatting and linting
- **mypy** — type checking
- **uv / poetry** — dependency management
- **pre-commit hooks** — enforce hygiene

🚀 Optional Extras

- **fastapi-mail** — in-game report/alert notifications
- **fastapi-background** — timed or delayed operations

- **sentry-sdk + opentelemetry** — observability stack
- **litellm** — optional LLM integration for adaptive AI messaging

---

🔄 5. Data Flow

[SvelteKit UI]
  ⇅ Axios / TanStack Query
[FastAPI REST API + SSE/WebSocket]
  ⇅ SQLModel / Redis / PostgreSQL

Interaction	Protocol	Description
Player commands	REST	`POST /api/commands`
Message feed	SSE/WebSocket	`GET /api/stream`
Orders	REST	`GET /api/orders`
Player state	REST	`GET /api/player/state`
Authentication	REST	`POST /api/auth/login`

🗒️ 6. Core Data Models

Entity	Description
Player	ID, name, metrics (technical, ethical, political), fatigue, attention
Message	category, priority, body, source, timestamp
Order	type, auth level, description, consequences
Event	triggers, actions, outcomes
Session	active shift log, start/end time

Action Log	every command and resulting consequence

Example — SQLModel

```
class Message(SQLModel, table=True):
    id: Optional[int] = Field(default=None, primary_key=True)
    category: str
    priority: int
    title: str
    body: str
    sender: str
    timestamp: datetime
    acknowledged: bool = False
```

⚡ 7. Real-Time Messaging

Option A — SSE (Server-Sent Events)

- Simple one-way data push (ideal for narrative flow)
- Low overhead
- Easy client subscription

Option B — WebSockets

- Two-way communication
- Enables multiplayer NOC ops or supervisor oversight
- Suitable for future expansions

🔬 8. Testing & QA Strategy

Layer	Tool	Focus
Frontend	Vitest	CLI parser, UI components
E2E	Playwright	Command flow, decision paths

Backend	pytest + httpx	API correctness
Linting	ESLint / Ruff / mypy	Code quality
Observability	Sentry / OpenTelemetry	Runtime tracing

🛰️ 9. Observability & Monitoring

- **Logging:** `loguru` (JSON mode)
- **Tracing:** `opentelemetry-instrumentation-fastapi`
- **Metrics:** Prometheus or OTEL exporter
- **Error Tracking:** Sentry

🤖 10. LLM Integration (Optional)

litellm — lightweight adapter for OpenAI, Anthropic, Mistral, or local models.

Use cases:

- Dynamic adversary messages (`AGGRESSOR`)
- Procedural supervisor tone changes
- Ethical tension scaling based on player history

> All generations validated via Pydantic/Zod before rendering.

🧱 11. Deployment Pipeline

Stage	Tool	Description
CI	GitHub Actions	Tests, lint, build
CD	Fly.io / AWS ECS	Deployment

Infra	Docker + Compose	Local & production parity
Secrets	.env + GitHub Secrets	Config management
Monitoring	Sentry + Logs	Runtime alerts

🧭 12. Production Flow Example

Player types:

> execute order-773

1.
2. Frontend sends REST request → `POST /api/commands/execute`.
3. FastAPI validates via Pydantic → updates `Order` in DB.
4. Event logged in `ActionLog`.

SSE emits response:

[OPS] Order 773 executed. Obedience +5 | Conscience -10

5.

🧠 13. Game Loop Recap

1. **Receive:** messages, alerts, or orders
2. **Interpret:** determine legitimacy & urgency
3. **Act:** execute, delay, investigate, deny
4. **React:** observe outcome, adjust reputation metrics
5. **Reflect:** each action shifts the world subtly

📜 14. Narrative Integration

Metric	Description	Influence
Technical	Accuracy & performance	System reliability feedback
Ethical	Conscience & empathy	Tone of coworkers & AI hints
Political	Obedience & loyalty	Supervisor trust & privileges

🧩 15. Stack Summary

Category	Technology	Purpose
Frontend	SvelteKit, Tailwind, DaisyUI, TypeScript	Terminal interface
Backend	FastAPI, SQLModel, Redis, Celery	API, logic, persistence
Database	PostgreSQL	Core state
Cache	Redis	Fast access, rate limiting
Storage	MinIO/S3	File storage
Auth	FastAPI Users + JWT	Secure sessions
AI Layer	LiteLLM	Dynamic text
Testing	Vitest, Playwright, pytest	Reliability
Monitoring	Sentry, OpenTelemetry	Stability
Deployment	Docker, ECS/Fly.io, GitHub Actions	CI/CD

🧭 16. Closing Statement

Nightshift: Analyst isn't a game about code — it's a game about conscience.
The tech stack is invisible to the player, yet vital to the illusion.

Every API call and message render exists to make the player **feel the weight of a decision** through the glow of a fake terminal.