OSIREON - Core Development Guidance

Overview

This document provides the foundational architecture and development guidelines for the Osireon Core System. The goal is to build a modular, scalable, and transparent backend that connects seamlessly with the frontend and is ready to support plug-and-play applications, modules, and simulations. The system will initially be developed with **Italy** as the first target country, with plans to expand internationally.

---

1. Objectives

- **Modularity**: The system must be built around independent modules with clear APIs.
- **Transparency**: Every process must be traceable and auditable.
- **Scalability**: Must support future integrations and increased computational load.
- **Security**: Built-in protection for data integrity, privacy, and access control.
- **Interoperability**: Can connect with various LLMs, frontends, external datasets, and applications.
- **Multinational Readiness**: Although initially focused on Italy, the system must support configuration and extension for other countries.

---

2. Tech Stack (Recommended)

- **Backend**: Python (FastAPI preferred for modular endpoints)
- **Database**: PostgreSQL (Relational), Redis (for caching and queueing)
- **Queueing**: Celery or FastAPI BackgroundTasks

- **Containerization**: Docker + Docker Compose
- **Orchestration**: Kubernetes (future scaling)
- **Frontend Bridge**: REST API (JSON), ready for GraphQL/WebSocket layer
- **Authentication**: JWT-based access (to be upgraded with OAuth2/Keycloak in v2)
- **LLM Interface**: OpenAI/LLM agnostic connector via API wrapper

---

3. Folder Structure (Initial Proposal)

```
osireon-core/
├── app/
│   ├── main.py            # Entry point
│   ├── core/              # Core logic engine
│   │   ├── engine.py      # Main reasoning/processing loop
│   │   ├── decision_tree.py # Core decision/simulation structure
│   ├── modules/           # Plug-in logic modules (LLM, economy, politics)
│   ├── agents/            # Independent reasoning agents (Analyst, Critic, Historian, etc.)
│   ├── adapters/          # Data source adapters (Eurostat, OWID, etc.)
│   ├── api/               # REST API endpoints
│   ├── db/                # DB models and interactions
│   ├── services/          # Services (auth, logging, mail, utils)
│   ├── schemas/           # Pydantic schemas
│   ├── config.py
│   ├── dependencies.py
├── diagrams/              # Architecture and agent diagrams
├── tests/
├── Dockerfile
├── docker-compose.yml
├── README.md
```

---

4. Core Features (MVP Phase)

4.1. System Bootstrap

- Launches the API, database, and logger

- Loads configuration (ENV-based)

4.2. LLM Connector Module

- Standard interface to plug any LLM (e.g., OpenAI, Claude, open-source)
- Logs all prompts and responses
- Configurable models per task type

4.3. Policy Simulation Engine (Minimal)

- Receives inputs (JSON)
- Processes logic via a configurable reasoning tree
- Returns simulated impact or response

4.4. Modular Input-Output Layer

- Accepts input from:
    - Web frontend
    - Admin backend
    - External apps via API key
- Responds with JSON packages (logs, results, metadata)

4.5. Agent-Based Execution Model

- Each task is assigned to one or more **Agents** with specific skills:
    - AnalystAgent: runs simulations and evaluations
    - CriticAgent: provides counterarguments and detects inconsistencies
    - HistorianAgent: checks precedents and timeline impact

- Agents communicate via internal task queue or event bus
- Can be extended with future Agents (e.g., LegalAgent, EthicsAgent)

4.6. Data Adapter Layer

- Each data source is accessed via a standard adapter:

```
class BaseAdapter:
    def fetch(self, query): ...
    def normalize(self, raw): ...
```

- Adapters available:
    - `italy_istat_adapter.py`
    - `eurostat_adapter.py`
    - `owid_adapter.py`
    - `mock_adapter.py`

---

5. API Specification (Draft)

Base URL: `/api/v1`

POST /simulate

- Description: Runs a simulation based on provided policy input
- Payload:

```
{
  "country": "Italy",
  "domain": "Tax Reform",
  "proposals": ["Flat tax 20%", "Universal Basic Income"],
```

```
  "constraints": ["No increase in national debt"]
}
```

- Response:

```
{
  "results": {...},
  "reasoning_tree": {...},
  "llm_output": "...",
  "meta": {
    "execution_time": 2.4,
    "modules_used": ["economy", "social"]
  }
}
```

---

6. Modules & Extensibility

Each module (e.g., economy, climate, education) is a Python class/function set with defined:

- `register()` method
- `execute(input)` method
- `metadata` and `version`

Modules are auto-loaded from `/modules` folder using a loader script. Each module also contains:

- Status: `stable`, `experimental`, or `deprecated`
- Compatible countries
- Required data inputs

---

7. Frontend Interface Hook

- REST API returns ready-to-render JSON
- Standard data format for charts, timelines, recommendations
- Add WebSocket hooks for live simulation and stream response (future)

---

8. Dev Notes

- Codebase must be well-documented and fully typed
- Use environment variables for all secrets and model keys
- Set up logging and error tracking (e.g., Sentry integration optional)
- All inputs/outputs must be logged in the DB for transparency
- Include initial unit test framework with coverage tracking

---

9. Future Goals (not MVP but planned)

- User management system
- Admin panel for simulation logs
- Full pipeline of scenario comparison and rating
- External API for contributors (public endpoint + rate limiting)
- Governance module integration
- Agent-to-agent communication layer
- Plugin interface for civic tech tools (Decidim, vTaiwan, etc.)
- Versioning dashboard for modules

---

10. Licensing & Ethics

- Core must follow AGPL v3 license
- Commercial/Pro modules can be added under dual-licensing system
- Code of ethics check in every module (auto-log if violated)

---

11. Diagrams & Visual Architecture (To Be Added)

- High-level system architecture (backend ↔ LLM ↔ frontend ↔ DB)
- Internal flow diagram (input → agents → modules → output)
- Modular architecture map
- Agent hierarchy and collaboration model

All diagrams will be saved in `/diagrams` in `.drawio` and `.png` formats.

Summary for Dev

Start by building:

- The API base (FastAPI)
- The logic engine scaffold with module loading
- A working simulation endpoint
- LLM wrapper for OpenAI
- Logs into PostgreSQL
- Mock modules: `economy_mock`, `ethics_mock`, and `italy_stats`

12. Extended Architecture & Advanced Features

The following elements expand the Osireon Core into a fully verifiable, ethics-driven, and community-integrated system.

12.1. Ethics Engine Codified on Blockchain

- A module within the Core Engine that verifies every decision against an immutable **Code of Ethics**.
- Each rule is versioned, hashed, and optionally stored on a blockchain (e.g., Ethereum/Arbitrum).
- The engine signs each simulation and stores the signature hash for later auditing.
- Verifier nodes (optional) can independently confirm that simulations respected all ethical rules.

Files and Functions:

/ethics/

├── ruleset_v1.json

├── validator.py # Applies rules

├── blockchain_client.py # Interface with smart contracts or decentralized log

12.2. License & Module Validation Layer

- A LicenseManager validates modules using:
    - License keys (for commercial modules)
    - Signature hash verification for integrity
- Open source modules are tagged as AGPL by default.
- Modules are loaded only if license and checksum pass.

Files:

/licenses/

├── manager.py

├── agpl_checker.py

├── commercial_checker.py

12.3. Contributor & DAO Interface

- Community members can submit new policies, modules, or feedback.
- Contributions go through a ProposalReviewFlow, optionally governed by:
    - Moderators
    - DAO voting (future)
- Approved contributions are logged and versioned.

Files:

/contributors/

├── portal.py

├── proposal_flow.py

├── dao_interface.py (future)

12.4. Public Audit Logging

- All simulations and decisions are saved to:
    - Local DB (PostgreSQL)
    - Decentralized log (e.g., Arweave/IPFS) for transparency

Logged data:

- Simulation ID
- Input, timestamp
- Output, reasoning
- Ethics hash & validator status
- Modules used & agents involved

12.5. Real-Time Simulation Layer (WebSocket API)

- For live dashboards and interactions.
- Streaming data (chunked responses) from agent evaluations and LLM reasoning.

Endpoint example:

/ws/simulation/live?id=...

12.6. Country-Specific Configurations

- Each country node includes:
    - Language pack
    - Local data adapters

    - Localized modules (legal, economic)
- Country configs can override default modules

Structure:

/countries/

├── italy/

│ ├── modules/

│ ├── adapters/

│ ├── translations.json

├── france/

├── usa/

Next Steps

- Implement core components from section 12 in parallel with MVP.
- Ethics and License layers can begin as mock checkers with logs.
- Community portal can be launched with proposal intake only, DAO later.

This extended architecture completes the vision of Osireon as a transparent, ethical, decentralized decision engine for the public.

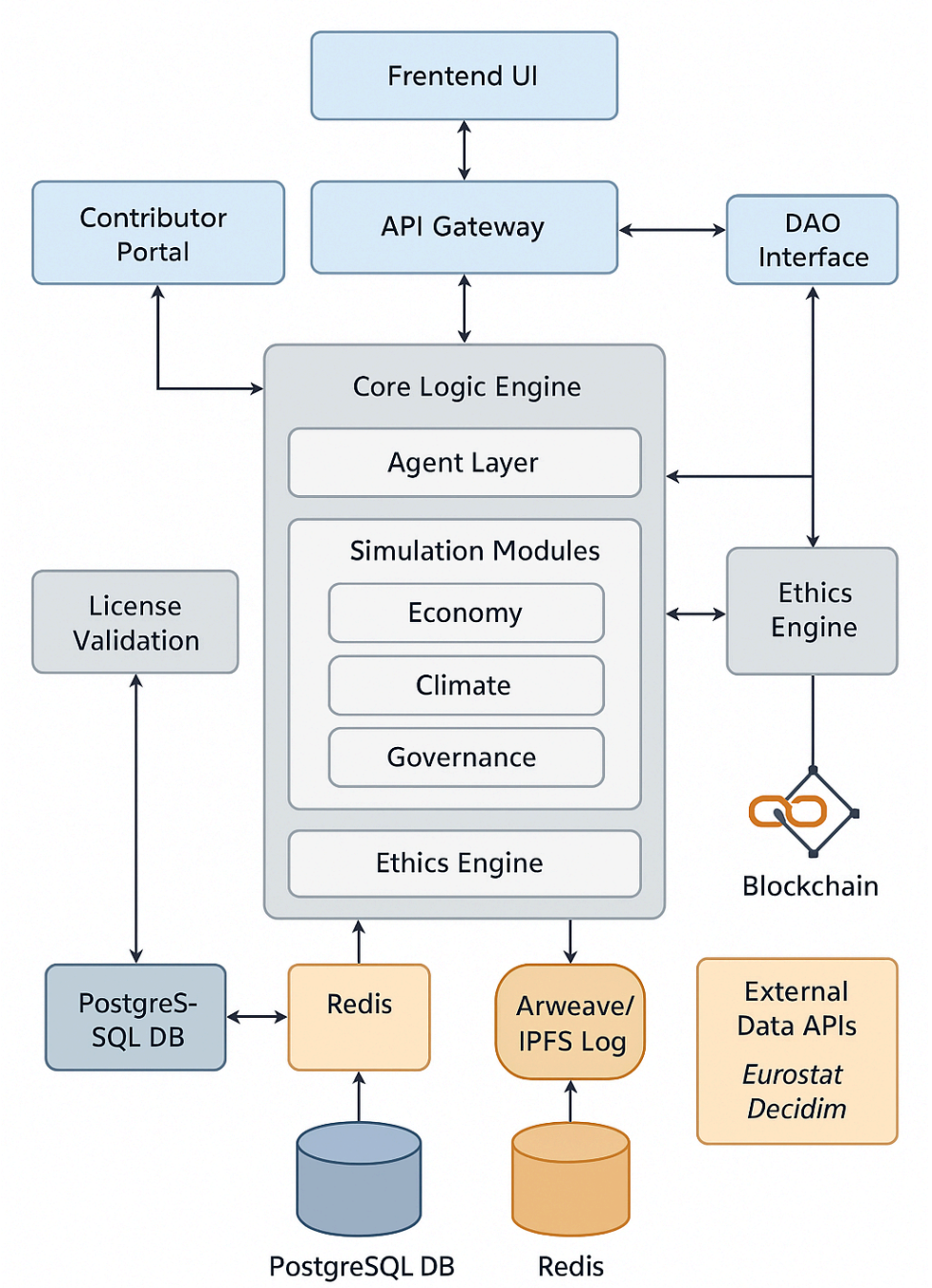
🧩 OSIREON System Diagrams – Overview & Prompts

Each of the following diagrams represents a critical layer of the Osireon Core Architecture. They help developers, contributors, and users visually understand how data flows, agents interact, and modules connect.

---

1. High-Level System Architecture

🎯 **Purpose:** Show the main components of Osireon and how they communicate: API, Core Engine, Modules, Agents, Frontend, LLM, and DB.



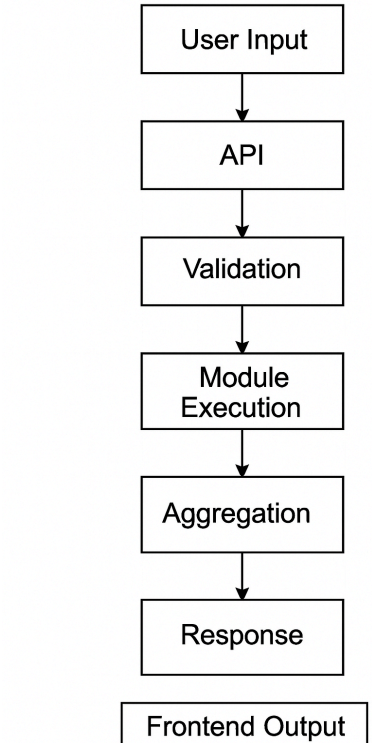
💬 **Diagram Comment:**

This shows the entire Osireon ecosystem at a glance. Each box is an independent service or layer that can evolve separately while staying interoperable with the whole system.

---

2. Simulation Data Flow

🎯 **Purpose:** Visualize the internal path of a simulation request, from input to final response.



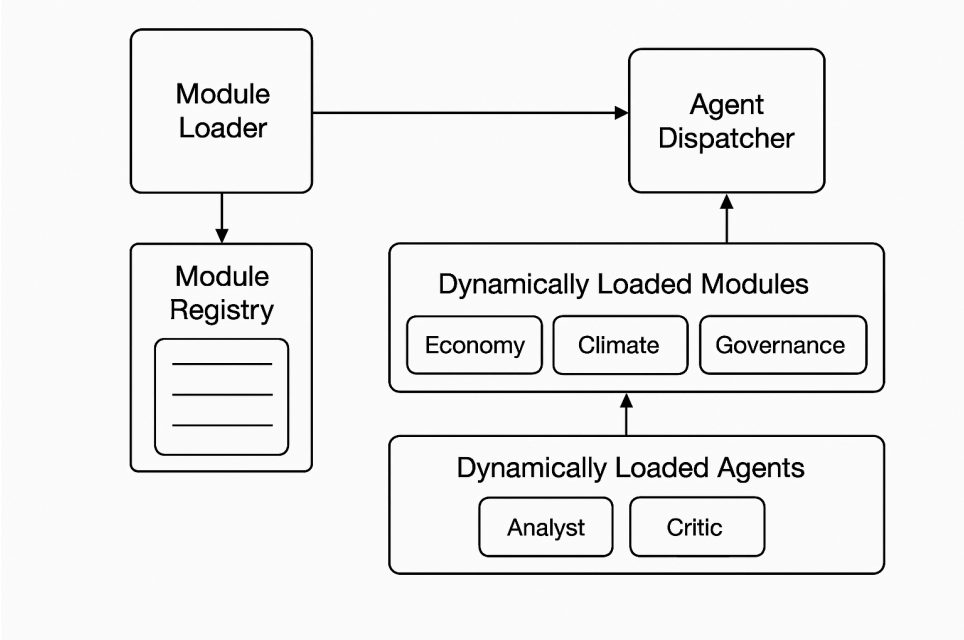
💬 **Diagram Comment:**

> This diagram reveals how Osireon transforms a single user input into a rich, multi-layered simulation using structured reasoning and AI assistance.

---

3. Modular Agent & Module Architecture

🎯 **Purpose:** Show how agents and modules are dynamically loaded, registered, and invoked.



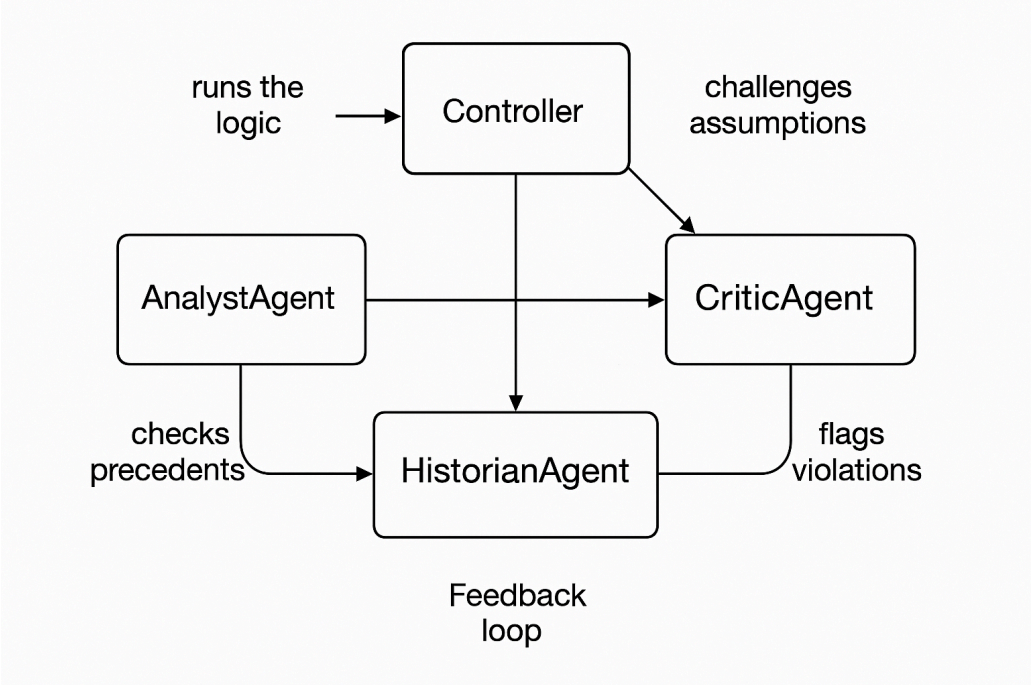
💬 **Diagram Comment:**

> This is the brain of Osireon. Everything is modular: agents act as specialized minds, modules as policy knowledge domains. The loader connects them in real time based on context.

---

4. Agent Collaboration & Interaction Map

🎯 **Purpose:** Display how agents collaborate, correct each other, and converge on a final output.



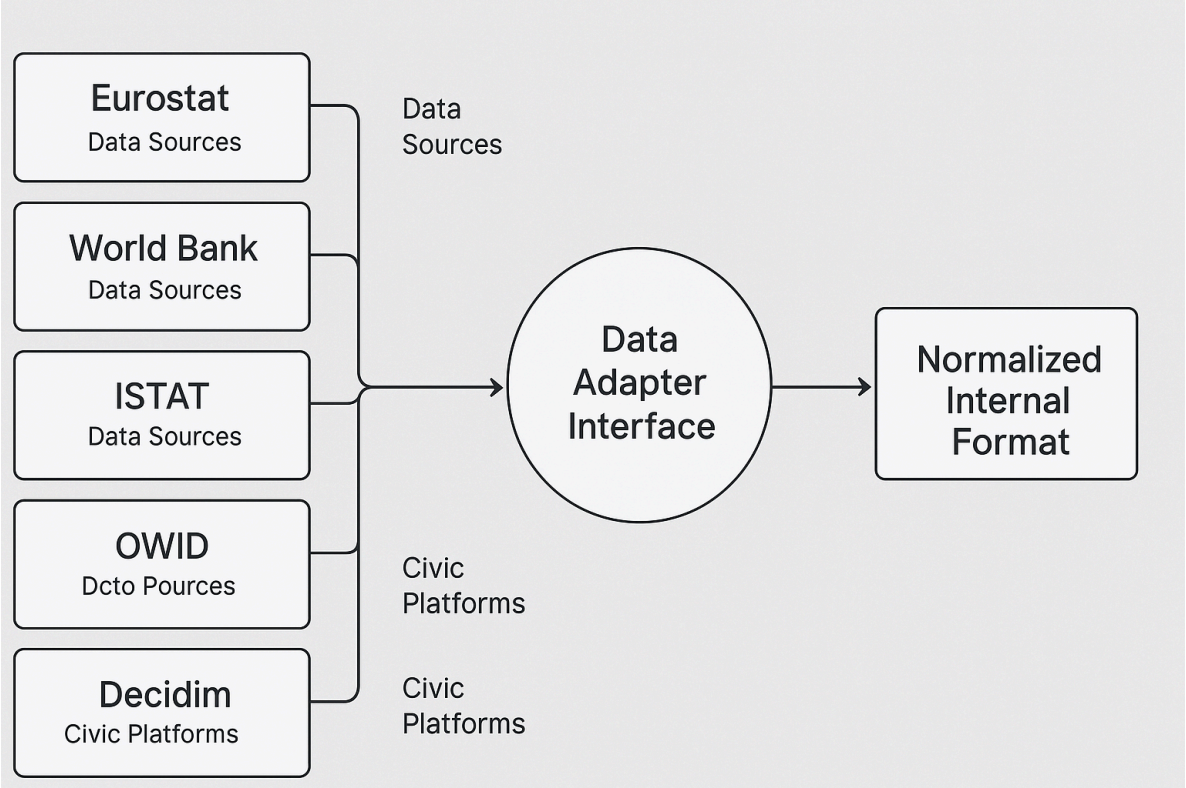
💬 **Diagram Comment:**

> Osireon simulates reasoned debate. Each agent has a defined purpose and checks the others, ensuring that proposals are thoroughly tested before becoming recommendations.

---

5. External Data Adapter Layer

🎯 **Purpose:** Show how Osireon connects to external databases and civic tech APIs via adapters.



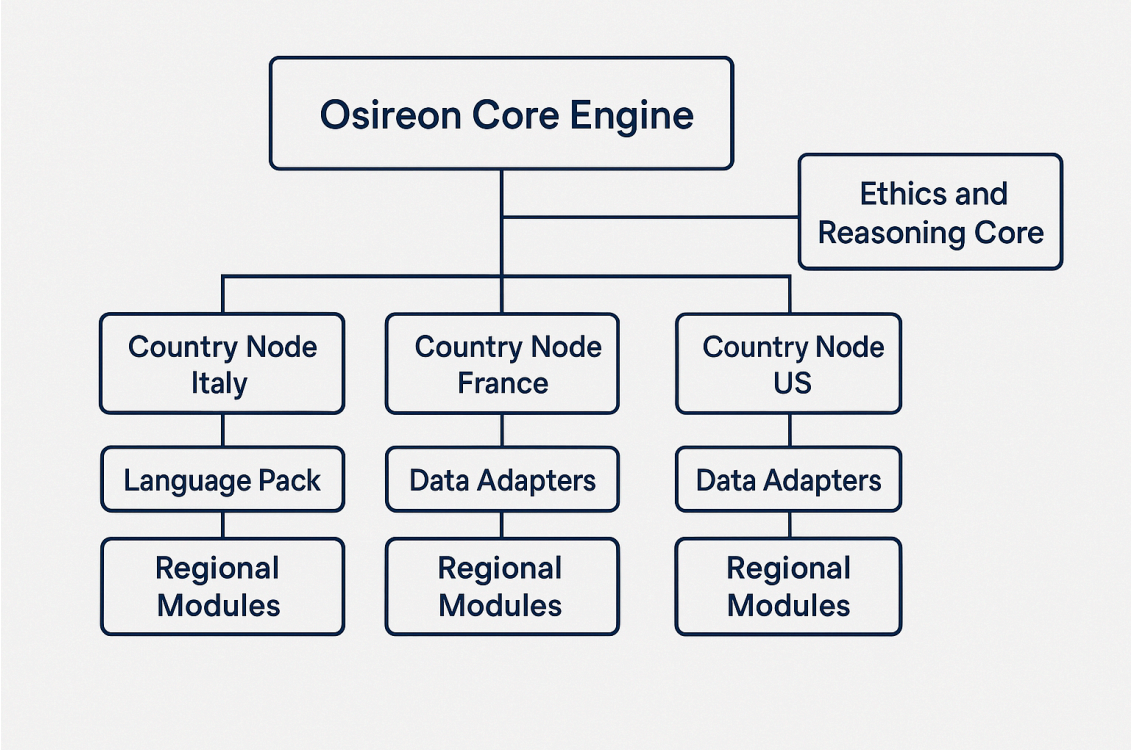
💬 **Diagram Comment:**

> Osireon doesn't guess – it grounds its logic in real-world facts. This adapter layer acts like a translator between diverse external datasets and the internal logic engine.

---

6. Scalable Multi-Country Configuration

🎯 **Purpose:** Display how Osireon can support multiple countries via independent configurations.



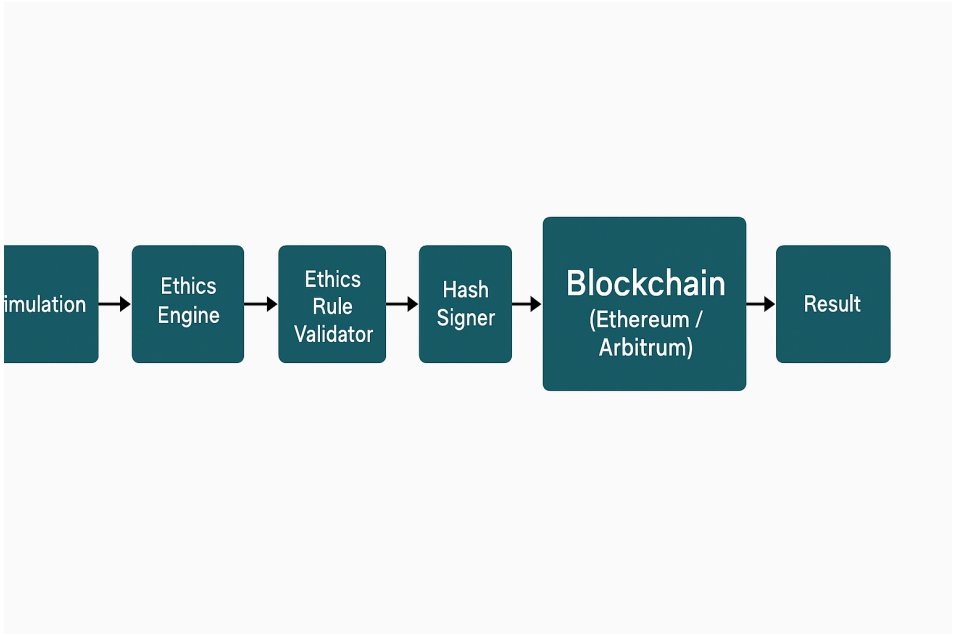
💬 **Diagram Comment:**

> Osireon is designed for global impact. Every country can have a custom layer on top of a shared reasoning core—allowing local adaptation without compromising global coherence.

7. Ethics Engine & Blockchain Validation Flow

🎯 **Objective:**

To visualize how policy simulations are validated against a codified code of ethics and recorded on the blockchain for immutability and transparency.



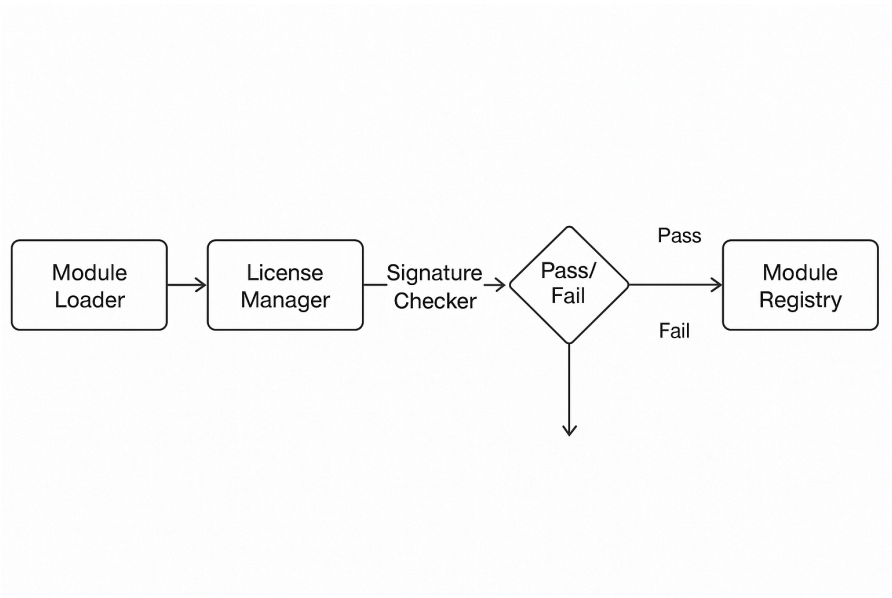
💬 **Diagram Comment:**

> Every simulation in Osireon is checked against an ethics ruleset. Once validated, the result is cryptographically signed and optionally stored on-chain to ensure public accountability and traceability.

8. License and Module Integrity Verification Flow

🎯 **Objective:**

To illustrate the process that verifies the license and integrity of each module before it is loaded into the system.



💬 **Diagram Comment:**

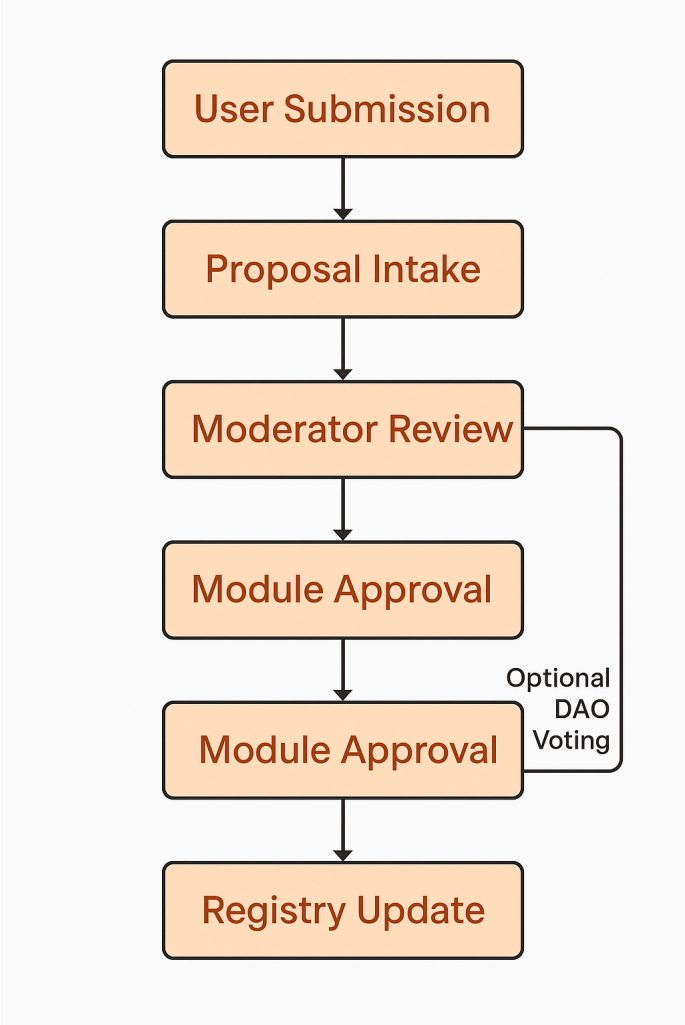
> This security layer protects Osireon's ecosystem from tampered or unauthorized modules. Only signed and licensed modules are allowed to interact with the core engine.

---

9. Contributor Proposal Flow & DAO Governance

🎯 **Objective:**

To show how new modules, simulations, or ideas submitted by users are reviewed, optionally voted by a DAO, and then added to the system.



💬 **Diagram Comment:**

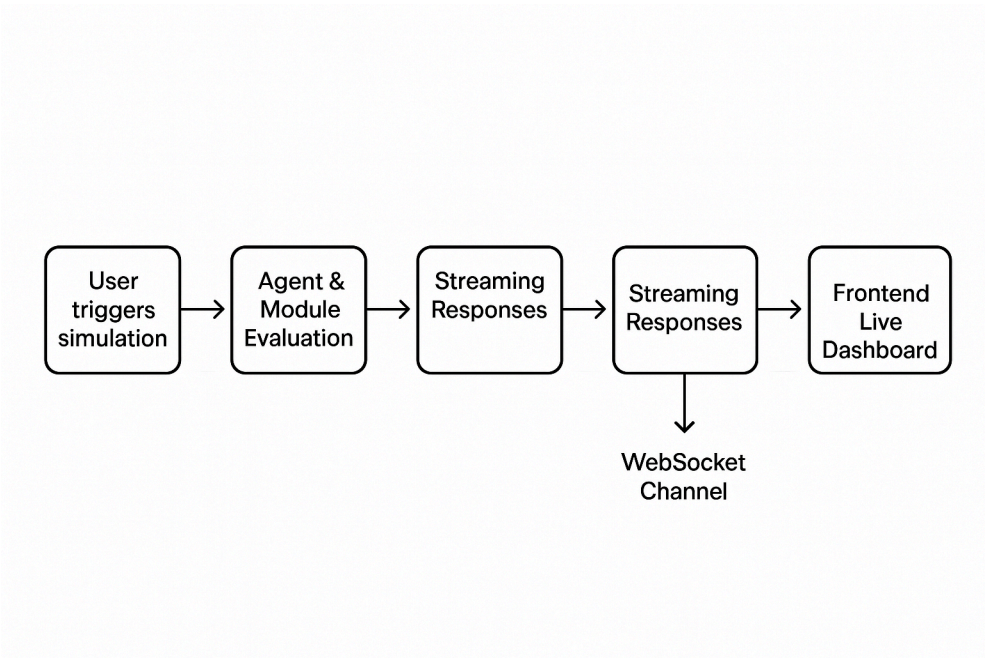
> This community flow ensures that innovation is decentralized, reviewed fairly, and managed transparently through democratic or delegated processes.

---

10. Real-Time Simulation & WebSocket Layer

🎯 **Objective:**

To demonstrate how a user-triggered simulation streams results live to the frontend using WebSocket for real-time feedback.



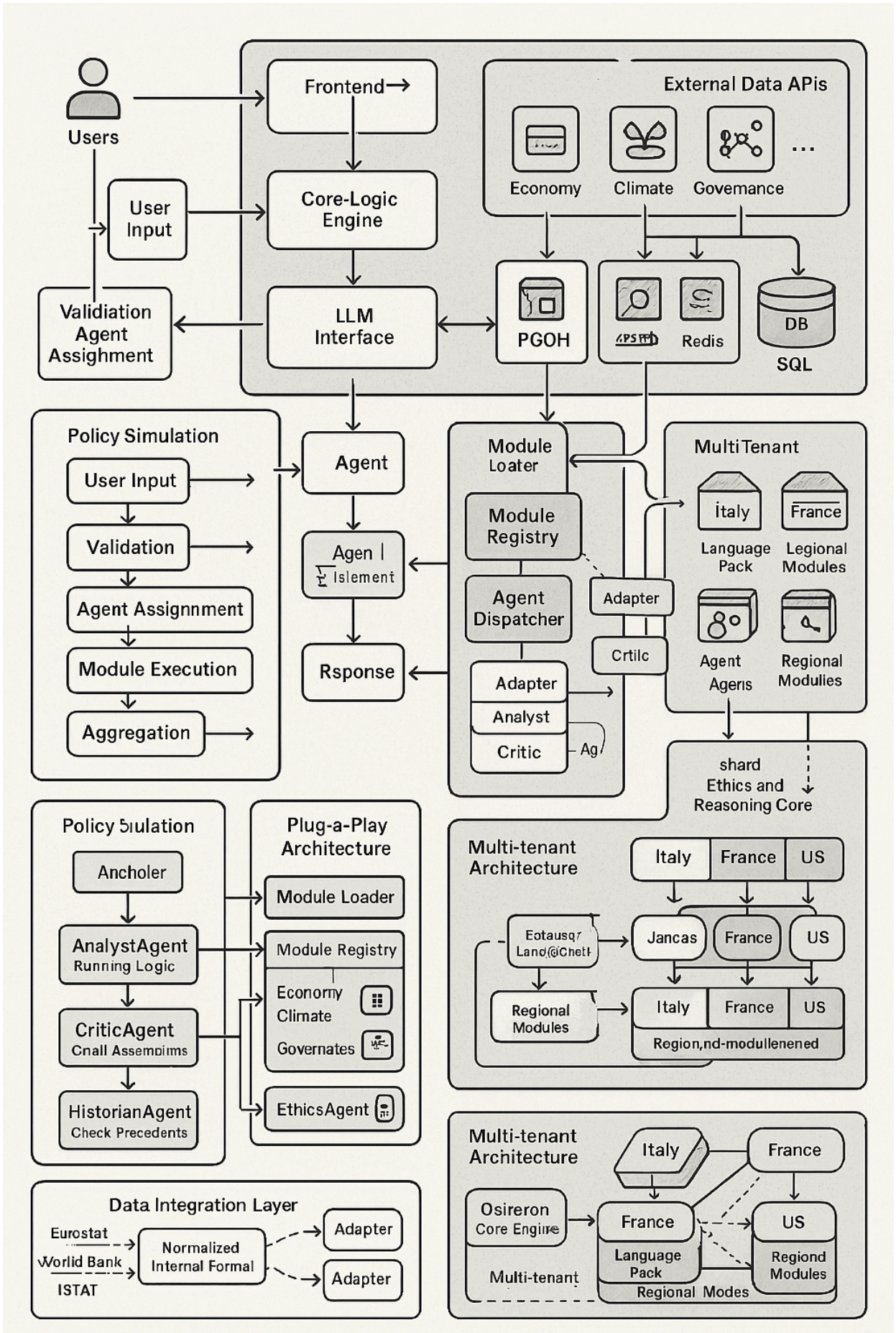
💬 **Diagram Comment:**

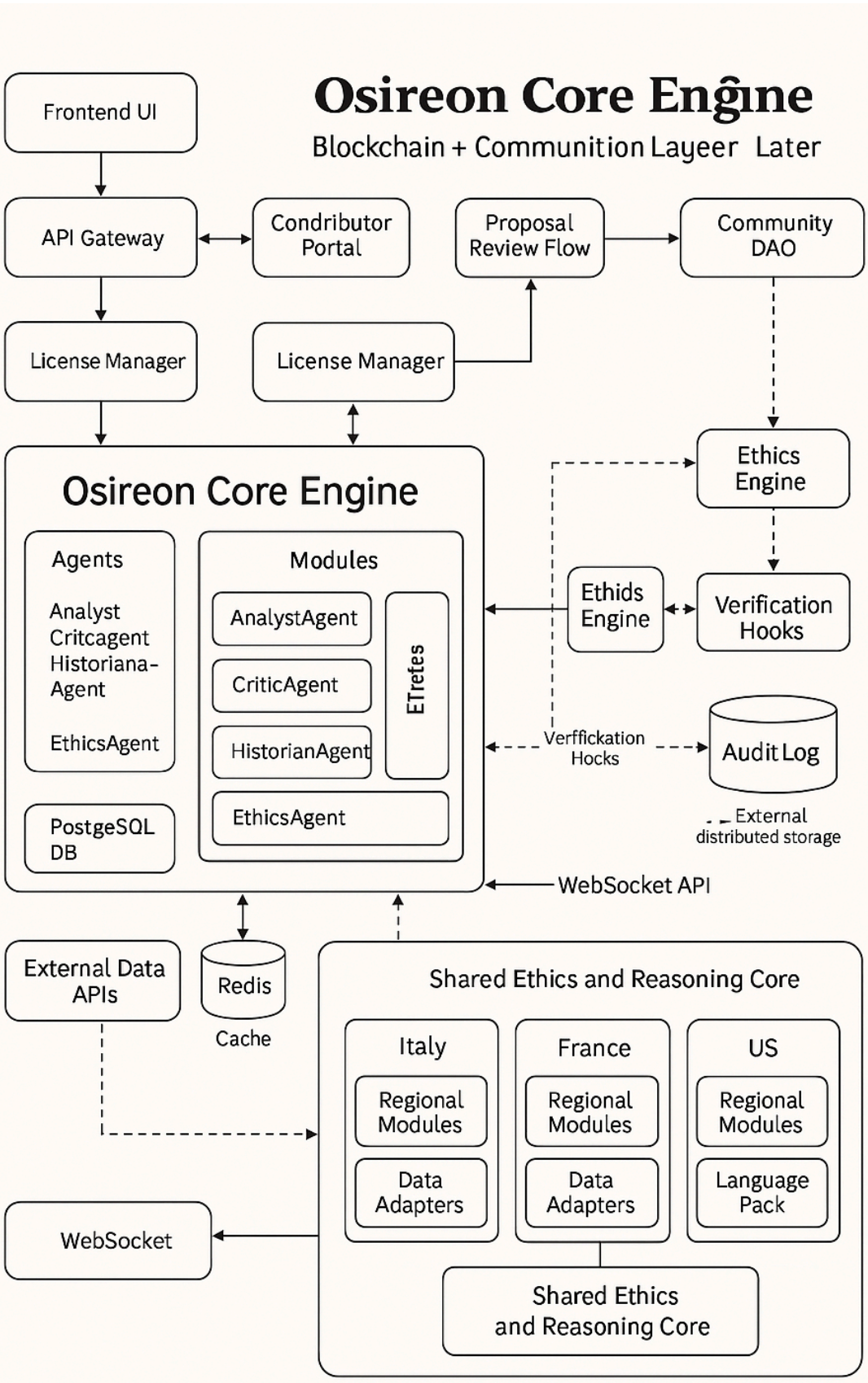
> With WebSocket integration, Osireon becomes a live civic lab. Users can visualize how decisions evolve in real time as agents and modules process them collaboratively.

Osireon Full System Architecture

🎯 Objective:

To provide a comprehensive overview of the Osireon v1.5+ architecture, including new advanced components like the Ethics Engine, Licensing System, DAO, Public Audit Log, and WebSocket support.


Users
Frontend
External Data APIs
Economy
Climate
Govemance
...
User Input
Core-Logic Engine
Validiation Agent Assighment
LLM Interface
PGOH
Redis
DB
SQL
Policy Simulation
User Input
Validation
Agent Assignnment
Module Execution
Aggregation
Agent
Agen I
Rsponse
Module Loater
Module Registry
Agent Dispatcher
Adapter
Crtilc
Adapter
Analyst
Critic
Ag
MultiTenant
İtaly
France
Language Pack
Legional Modules
Agent Ageris
Regional Modulies
shard Ethics and Reasoning Core
Policy 5iulation
Ancholer
AnalystAgent
Running Logic
CriticAgent
Cnall Assemplims
HistorianAgent
Check Precedents
Plug-a-Play Architecture
Module Loader
Module Registry
Economy
Climate
Governates
EthicsAgent
Multi-tenant Architecture
Italy
France
US
Jancas
Regional Modules
Region,nd-modullenened
Multi-tenant Architecture
Italy
France
Osireron
Core Engine
France
US
Language Pack
Regiond Modules
Multi-tenant
Regional Modes
Data Integration Layer
Eurostat
Worlid Bank
ISTAT
Normalized Internal Formal
Adapter
Adapter


Osireon Core Engine
Blockchain + Communition Layeer Later
Frontend UI
API Gateway
Condributor Portal
Proposal Review Flow
Community DAO
License Manager
License Manager
Ethics Engine
Osireon Core Engine
Agents
Analyst
Critcagent
Historiana-Agent
EthicsAgent
Modules
AnalystAgent
CriticAgent
HistorianAgent
EthicsAgent
ETrefes
Ethids Engine
Verification Hooks
Verffickation Hocks
Audit Log
External distributed storage
PostgeSQL DB
WebSocket API
External Data APIs
Redis
Cache
Shared Ethics and Reasoning Core
Italy
France
US
Regional Modules
Regional Modules
Regional Modules
Data Adapters
Data Adapters
Language Pack
WebSocket
Shared Ethics and Reasoning Core

CANLQE

💬 **Diagram Comment:**

> This is the complete system view of Osireon. It integrates community participation, ethical enforcement, decentralized governance, and real-time capabilities — all connected to a modular and auditable simulation engine.

Questions?

Ping Giorgio or submit ideas/issues directly into the GitHub repo.
