ERES PlayNAC "KERNEL" Codebase (v5.0 Draft)

🚀 Project Overview

ERES PlayNAC (New Age Cybernetic Game Theory) is a modular framework designed to empower User-GROUPs with real-time merit-based progression, bio-ecologic scoring, and immersive voice-navigated game environments. The **KERNEL** serves as the core engine, integrating simulations (EarnedPath), planetary planning (GiantERP), bio-energetic proof-of-work (BEE), consensus mechanics (BERC), and advanced media processing for adaptive RT Media delivery.

Version 5.0 refines the architecture into discrete modules, enhances security and logging, and lays out a clear developer onboarding path.

📦 Repository Structure

```
├── docs/               # Design docs, architecture diagrams, and whitepapers
├── src/                # Core Python packages
│   ├── kernel/          # Initialization and orchestration layer
│   ├── earnedpath/      # Simulation engine and merit calculus
│   ├── gianterp/        # Global Earth Resource Planner interface
│   ├── bee/             # Bio-Energetic PoW utilities
│   ├── berc/            # Bio-Electric Ratings Codex & consensus logic
│   ├── media/           # RT Media processor, adaptive algorithms
│   ├── nav/             # Voice navigation and NLP module
│   └── utils/           # Common utilities (logging, config, exceptions)
├── examples/           # Sample scripts and notebooks
├── tests/              # Unit and integration tests (pytest)
├── .github/            # CI/CD workflows (lint, tests, build)
├── Dockerfile          # Container definition for reproducible environments
├── requirements.txt    # Python dependencies (pinned)
├── pyproject.toml      # Build system and metadata
└── README.md           # This file
```

⚙️ Installation

Clone the repository

```
git clone https://github.com/ERES-Institute-for-New-Age-Cybernetics/PlayNAC-KERNEL.git
```

1. cd PlayNAC-KERNEL

Set up a virtual environment

```
python3 -m venv venv
```

2. source venv/bin/activate

Install dependencies

```
pip install --upgrade pip
```

3. pip install -r requirements.txt

Configure environment variables

```
export WEB3_RPC_URL="https://your-node-url"
```

4. export BEE_SECRET_KEY="your-secret"
5. **Run tests to verify setup**
   ```
   pytest --maxfail=1 --disable-warnings -q
   ```

🏗️ Architecture & Modules

1. `kernel/`

- `PlayNACKernel`: Orchestrates initialization, module loading, and global configuration.
- `ConfigManager`: Reads and validates settings from environment and config files.

2. `earnedpath/`

- Implements core simulation: modules, CPM/WBS/PERT pipelines, and merit scoring.
- **Key classes**: `EPNode`, `MeritCalculator`, `SimulationEngine`.

3. `gianterp/`

- Interfaces with the GiantERP API to fetch planetary resource grids and projections.
- Handles rate-limiting, caching, and data normalization.

4. `bee/`

- Provides bio-energetic proof-of-work (BEE) algorithms using kirlianographic metrics.
- **Utilities**: `BEEGenerator`, `AuraAnalyzer`.

5. `berc/`

- Encapsulates the Bio-Electric Ratings Codex consensus protocol.
- **Components**: `BERCConsensus`, `NodeRegistry`, `RatingValidator`.

6. `media/`

- Manages real-time media ingestion, adaptive encoding, and dynamic filters.
- Supports image/video streams, 3D scene graphs, and holographic overlays.

7. `nav/`

- Voice-driven navigation: ASR, NLP intent parsing, and context-aware dialogues.
- Integrates with external services for speech-to-text and TTS.

8. `utils/`

- Cross-cutting utilities: `exceptions.py`, `logger.py` (with configurable levels), `helpers.py`.

🧪 Testing & CI/CD

- **Unit Tests**: Located under `tests/`, covering >90% of core logic.
- **Integration Tests**: Simulate end-to-end flows (EarnedPath ➔ BEE ➔ BERC consensus).
- **CI Pipeline** (`.github/workflows/ci.yml`): Runs lint (`flake8`), type-check (`mypy`), and tests on push.
- **Coverage**: Enabled via `coverage.py`, with thresholds enforced in CI.

🔒 Security & Best Practices

- **Input Sanitization**: All external inputs (e.g., blockchain txs, media uploads) are validated.

- **Secret Management**: Use environment variables or vault integration; no hard-coded credentials.
- **Dependency Pinning**: See `requirements.txt` for specific version constraints.
- **Logging**: Centralized via `utils/logger.py` with structured JSON output for observability.
- **Containerization**: Dockerfile defines a minimal, production-ready build image.

📚 Documentation

- **Design Documents**: Located in `docs/architecture/`, including component interaction diagrams (UML), sequence charts, and data-flow schematics in both PDF and Markdown formats.
- **API Reference**: Auto-generated via Sphinx (in `docs/api/`):
  - `kernel/index.html` – Core orchestration and configuration classes.
  - `earnedpath/index.html` – Simulation engine classes: `EPNode`, `MeritCalculator`, `SimulationEngine`, and related utilities.
  - `gianterp/index.html` – GiantERP interface classes and data models.
  - `bee/index.html` – BEE proof-of-work algorithms and aura analysis tools.
  - `berc/index.html` – Consensus protocol implementation and rating validators.
  - `media/index.html` – RT Media processing pipeline, format handlers, and adaptive filter APIs.
  - `nav/index.html` – NLP intent parser, ASR clients, and TTS integration interfaces.
  - `utils/index.html` – Exception hierarchy, logging utilities, and helper functions.

🛠️ Developer Code Documentation

Below is the full API reference for all core packages, including class-level docstrings, attributes, and public method signatures. Use these definitions to navigate and extend the codebase.

---

1. `src/kernel/`

`ConfigManager`

```
class ConfigManager:
    """
    Reads, validates, and provides access to environment variables and config files.

    Attributes:
        env_file (str): Path to .env file.
        config (Dict[str, Any]): Loaded configuration values.

    Methods:
        load_env() -> None
            Load environment variables from the .env file into the process environment.

        validate(required_keys: List[str]) -> None
            Ensure that all keys in `required_keys` are present in `config` or env.

        get(key: str, default: Any = None) -> Any
            Retrieve a configuration value, returning `default` if missing.
    """
    def load_env(self) -> None: ...

    def validate(self, required_keys: List[str]) -> None: ...

    def get(self, key: str, default: Any = None) -> Any: ...
```

PlayNACKernel

```
class PlayNACKernel:

    """

    Main orchestrator for the PlayNAC Kernel AI system.



    Attributes:

        bio_pow (BioPoW): Bio-energetic PoW engine.

        media_processor (MediaProcessor): Real-time media processor.

        jas_consensus (JASConsensus): Graph-based consensus manager.

        blockchain (List[Block]): Chain of mined blocks.

        pending_tasks (List[MediaTask]): Tasks awaiting processing.

        mining_active (bool): Mining loop status flag.

    """

    def __init__(self) -> None: ...

    def submit_media_task(self, frame: np.ndarray, task_type: str = "style_transfer") -> str: ...

    def mine_block(self, max_iterations: int = 1000) -> Optional[Block]: ...

    def get_status(self) -> Dict[str, Any]: ...
```

---

2. src/earnedpath/

EPNode

```
class EPNode:
    """
    Represents a node in the EarnedPath graph.

    Attributes:
        node_id (str): Unique identifier.
        dependencies (List[str]): IDs of prerequisite nodes.
        state (Enum): Current status (LOCKED, UNLOCKED, COMPLETED).
    """
    def unlock(self) -> None: ...
    def complete(self, result: Any) -> None: ...
```

MeritCalculator

```
class MeritCalculator:
    """
    Calculates merit scores based on user actions and earned-path progress.
    """
    def calculate_merit(self, actions: List[Action]) -> float: ...
```

SimulationEngine

```
class SimulationEngine:
    """
    Executes simulation scenarios using CPM/WBS/PERT pipelines.
    """
    def setup_scenario(self, config: Dict[str, Any]) -> None: ...
    def step(self) -> SimulationResult: ...
    def report(self) -> Report: ...
```

3. `src/gianterp/`

`GiantERPClient`

```
class GiantERPClient:

    """

    HTTP client for the Global Earth Resource Planner API.


    Methods:

        fetch_grid(region_id: str) -> ResourceGrid

        submit_projection(data: ProjectionInput) -> ProjectionResult

    """

    def fetch_grid(self, region_id: str) -> ResourceGrid: ...

    def submit_projection(self, data: ProjectionInput) -> ProjectionResult: ...
```

`ResourceGrid`

```
@dataclass

class ResourceGrid:

    region_id: str

    capacity: float

    forecast: Dict[str, float]
```

4. `src/bee/`

`AuraScanner`

```
class AuraScanner:

    """
    Interface to biofeedback hardware (EEG) for aura data capture.
    """
    def capture(self) -> np.ndarray: ...
    def is_device_connected(self) -> bool: ...
```

`BioPoW`

```
class BioPoW:
    """
    Generates and validates bio-energetic proof-of-work values (EP).
    """
    def generate_ep(self) -> float: ...
    def validate_bio_work(self, ep_value: float, target: float, tolerance: float = 0.01) -> bool: ...
    def get_aura_entropy(self) -> float: ...
```

---

5. `src/berc/`

`JASLink`

```
@dataclass
class JASLink:
    source_hash: str
    target_hash: str
    weight: float
    timestamp: float
    ep_correlation: float
```

`MediaTask`

```
@dataclass
```

```
class MediaTask:
    id: str
    input_frame: np.ndarray
    task_type: str
    nonce: int
    timestamp: float
    ep_value: float = 0.0
```

JASConsensus

```
class JASConsensus:
    """
    Manages graph-based consensus for media tasks in the JAS network.
    """
    def create_link(self, source_task: MediaTask, target_task: MediaTask, ep_correlation: float) -> JASLink: ...
    def validate_consensus(self, task_hash: str) -> bool: ...
    def get_graph_metrics(self) -> Dict[str, Any]: ...
```

6. src/media/

MediaProcessor

```
class MediaProcessor:
    """
    Real-time media processing kernel with MD-Complexity and GERP transformation.
    """
    def calculate_md_complexity(self, frame: np.ndarray) -> float: ...
    def validate_md_complexity(self, frame: np.ndarray) -> bool: ...

    def gerp_transform(self, frame: np.ndarray, ep_value: float) -> np.ndarray: ...

    def process_media_task(self, task: MediaTask) -> np.ndarray: ...
```

---

7. `src/nav/`

`ASRClient`

```
class ASRClient(ABC):

    """

    Abstract base for speech-to-text providers.

    """

    @abstractmethod

    def transcribe(self, audio: Any) -> str: ...
```

`IntentParser`

```
class IntentParser:

    """

    Parses user utterances into structured intents.

    """

    def parse(self, text: str) -> Intent: ...
```

`DialogueManager`

```
class DialogueManager:

    """

    Manages conversation context, slot-filling, and routing.

    """

    def handle_intent(self, intent: Intent) -> Response: ...
```

8. `src/utils/`

`exceptions.py`

```
class KernelError(Exception): pass

class ModuleLoadError(KernelError): pass

class ConfigError(KernelError): pass
```

`logger.py`

```
from logging import Logger


def get_logger(name: str) -> Logger:

    """Returns a configured JSON logger for the given module name."""
```

`helpers.py`

```
def retry(func=None, *, retries: int = 3, delay: float = 1.0): ...


def timed_cache(maxsize: int = 128, ttl: int = 300): ...
```

For interactive exploration, generate HTML docs via Sphinx:

cd docs/ && make html && open _build/html/index.html

🤝 Contributing

We welcome community contributions! Please follow these steps:

1. **Fork** the repo and create a feature branch: `git checkout -b feature/YourFeature`
2. **Implement** your changes, adhering to our code style (`flake8`, `black`).
3. **Add** tests for new functionality.
4. **Submit** a Pull Request against the `main` branch.
5. **Review**: A maintainer will review, suggest changes, and merge.

Please read `CONTRIBUTING.md` for detailed guidelines.

📝 Changelog

See `CHANGELOG.md` for details on version changes. Highlights for v5.0 Draft:

- Modular repo structure
- Added logging & error-handling across modules
- Introduced CI pipeline with GitHub Actions
- Secured default configs and secret management
- Provided Docker support and test coverage enforcement

📜 License

This project is licensed under **CC BY-NC-SA 4.0**. See LICENSE for details.

Prepared by the ERES Institute for New Age Cybernetics — advancing civilization through empirical, real-time, and meritocratic systems.