

This codebase skeleton provides a runnable foundation for the ERES PlayNAC “KERNEL”—Empirical Realtime Education System × New Age Cybernetic Game Theory—integrating core modules into an end-to-end demo kernel. Developers can incrementally enhance each component per the Phase 1 roadmap.

```
## src/kernel/config.py
```

```
```python
```

```
import os
```

```
from typing import Any, List
```

```
class ConfigManager:
```

```
 """
```

```
 Loads environment files, validates required keys, and provides getters.
```

```
 Supports multiple .env files for different environments.
```

```
 """
```

```
 def __init__(self, env_files: List[str] = [".env"]):
```

```
 self.env_files = env_files
```

```
 self._loaded = False
```

```
 def load_env(self) -> None:
```

```
if self._loaded:

 return

for file in self.env_files:

 if os.path.isfile(file):

 with open(file) as f:

 for line in f:

 if line.strip().startswith('#') or '=' not in line:

 continue

 key, val = line.strip().split('=', 1)

 os.environ.setdefault(key, val)

self._loaded = True

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

 missing = [k for k in required_keys if k not in os.environ]

 if missing:

 raise KeyError(f"Missing required config keys: {missing}")

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

 return os.environ.get(key, default)

...

src/kernel/models.py
```

```
```python
from dataclasses import dataclass
from typing import Any

@dataclass
class Block:
    index: int
    timestamp: float
    data: Any
    previous_hash: str
    nonce: int
    hash: str
...

---

## src/kernel/playnac_kernel.py
```python
import time
import hashlib
from typing import Optional, List, Any
from .config import ConfigManager
from .models import Block
from bee.bio_pow import BioPoW
```

```
from earnedpath.simulation_engine import SimulationEngine

from berc.jas_consensus import JASConsensus

from media.media_processor import MediaProcessor

from berc.models import MediaTask

class PlayNACKernel:

 """

 Core orchestrator for ERES PlayNAC "KERNEL".

 Bootstraps modules, manages mining loop, and exposes status APIs.

 """

 def __init__(self):

 # Load and validate config

 self.config = ConfigManager(env_files=[".env", ".env.local"])

 self.config.load_env()

 self.config.validate(["WEB3_RPC_URL", "BEE_SECRET_KEY"])

 # Initialize core engines

 self.bio_pow = BioPoW(secret_key=self.config.get("BEE_SECRET_KEY"))

 self.sim_engine = SimulationEngine()

 self.media_processor = MediaProcessor()

 self.consensus = JASConsensus()

 # In-memory stores
```

```
self.blockchain: List[Block] = []

self.task_queue: List[MediaTask] = []

def submit_media_task(self, task: MediaTask) -> None:

 """Queue a media processing task for mining."""

 self.task_queue.append(task)

def _calculate_hash(self, index: int, timestamp: float, data: Any,
 previous_hash: str, nonce: int) -> str:

 block_string = f"{index}{timestamp}{data}{previous_hash}{nonce}".encode()

 return hashlib.sha256(block_string).hexdigest()

def _get_difficulty_target(self) -> float:

 """Adaptive difficulty: average EP of last N blocks."""

 recent = [b.data['ep'] for b in self.blockchain[-5:]]

 return sum(recent) / len(recent) if recent else 0.5

def mine_block(self, max_nonce: int = 10000) -> Optional[Block]:

 """

 Mine a new block using BioPoW + media processing.

 Returns the new Block or None if mining fails.

 """

 if not self.task_queue:

 return None
```

```
task = self.task_queue.pop(0)

task.ep_value = self.bio_pow.generate_ep()

for nonce in range(max_nonce):

 task.nonce = nonce

 try:

 processed = self.media_processor.process_media_task(task)

 except ValueError:

 continue

 target = self._get_difficulty_target()

 if not self.bio_pow.validate(task.ep_value, target):

 continue

 index = len(self.blockchain)

 timestamp = time.time()

 data = {

 'task_id': task.id,

 'ep': task.ep_value,

 'media_hash': hashlib.sha256(processed.tobytes()).hexdigest()

 }

 previous_hash = self.blockchain[-1].hash if self.blockchain else '0'*64

 block_hash = self._calculate_hash(index, timestamp, data, previous_hash, nonce)
```

```
block = Block(index, timestamp, data, previous_hash, nonce, block_hash)

self.blockchain.append(block)

Link consensus graph

if index > 0:

 prev_id = self.blockchain[index-1].data["task_id"]

 self.consensus.create_link(

 MediaTask(prev_id, None, "", 0, 0), task, task.ep_value

)

return block

return None

def run(self, iterations: int = 1) -> None:

 """Run the kernel mining loop for a set number of iterations."""

 for _ in range(iterations):

 block = self.mine_block()

 if block:

 print(f"✓ Mined block {block.index}: {block.hash[:8]}... EP={block.data['ep']:.4f}")

if __name__ == '__main__':

 from berc.models import MediaTask
```

```
Demo: submit a sample task and mine one block
```

```
kernel = PlayNACKernel()
```

```
sample = MediaTask('task0', b'sample_frame', 'style_transfer', 0, time.time())
```

```
kernel.submit_media_task(sample)
```

```
kernel.run(1)
```

```
...
```

```

```

**\*\*Next Steps:\*\***

- Persist blockchain to disk or database.
- Hook in real EarnedPath, GERP, NBERS, and other domain modules.
- Integrate context management, ingestion pipelines, and user-facing APIs.
- Add comprehensive tests, logging, and deployment configurations.