Create the following in my Hardhat workspace.

1. File contracts/GoldReserveRegistry.sol with this content

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/AccessControl.sol";

interface IDIDRegistry {
  function isValid(address subject, bytes32 schema) external view returns (bool);
}

interface IAnalyticsHub {
  function publish(uint256 nodeId, uint256 epoch, bytes32 digest, bytes32 cid) external;
}

// GoldReserveRegistry
// Ledger-only registry for vaulted gold used as a transparency layer for AXIOM.
// No token custody. Stores certificate metadata, grams, custodian attestations, audit references.
// Includes enumerable label allocations with a hard cap that total allocated grams cannot
exceed total verified grams.
contract GoldReserveRegistry is AccessControl {
  bytes32 public constant MANAGER_ROLE = keccak256("MANAGER_ROLE");
  bytes32 public constant CUSTODIAN_ROLE = keccak256("CUSTODIAN_ROLE");
  bytes32 public constant AUDITOR_ROLE = keccak256("AUDITOR_ROLE");

  struct GoldAsset {
    string certId;
    string custodianName;
    address custodianAddr;
    string vaultId;
    string serial;
    uint256 grams;
    bool verified;
    uint64 lastAuditAt;
    bytes32 lastAuditCid;
    bool active;
  }

  IDIDRegistry public did;
  IAnalyticsHub public analytics;

  bytes32 public custodianSchema;
  bytes32 public auditorSchema;

  uint256 public nextId = 1;
  mapping(uint256 => GoldAsset) public assets;

  // enumerable allocation labels
  bytes32[] private allocationLabels;
  mapping(bytes32 => uint256) private allocationIndex; // label -> index plus one
  mapping(bytes32 => uint256) public allocationGrams;

  uint256 public totalGrams;
  uint256 public totalVerifiedGrams;
```

```
    event DidRegistrySet(address didRegistry, bytes32 custodianSchema, bytes32
auditorSchema);
    event AnalyticsHubSet(address analyticsHub);
    event Registered(uint256 indexed id, string certId, string custodianName, address
custodianAddr, string vaultId, string serial, uint256 grams, bool active, bool verified);
    event Updated(uint256 indexed id, uint256 grams, bool active);
    event Verified(uint256 indexed id, bool verified, uint64 when, bytes32 auditCid);
    event AllocationSet(bytes32 indexed label, uint256 grams);
    event AllocationRemoved(bytes32 indexed label);
    event CustodianChanged(uint256 indexed id, string name, address addr);

    constructor(address admin) {
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(MANAGER_ROLE, admin);
        _grantRole(CUSTODIAN_ROLE, admin);
        _grantRole(AUDITOR_ROLE, admin);
    }

    // wiring

    function setDidRegistry(address registry, bytes32 custodianSch, bytes32 auditorSch) external
onlyRole(MANAGER_ROLE) {
        did = IDIDRegistry(registry);
        custodianSchema = custodianSch;
        auditorSchema = auditorSch;
        emit DidRegistrySet(registry, custodianSch, auditorSch);
    }

    function setAnalyticsHub(address hub) external onlyRole(MANAGER_ROLE) {
        analytics = IAnalyticsHub(hub);
        emit AnalyticsHubSet(hub);
    }

    // asset lifecycle

    function registerAsset(
        string calldata certId,
        string calldata custodianName,
        address custodianAddr,
        string calldata vaultId,
        string calldata serial,
        uint256 grams,
        bool markActive
    ) external onlyRole(CUSTODIAN_ROLE) returns (uint256 id) {
        require(bytes(certId).length > 0, "cert required");
        require(custodianAddr != address(0), "custodian addr");
        require(grams > 0, "grams=0");

        id = nextId++;
        bool initialVerified = _isCustodianValid(custodianAddr);
        assets[id] = GoldAsset({
            certId: certId,
            custodianName: custodianName,
            custodianAddr: custodianAddr,
            vaultId: vaultId,
            serial: serial,
            grams: grams,
            verified: initialVerified,
            lastAuditAt: 0,
            lastAuditCid: 0x0,
            active: markActive
        });

        if (markActive) {
            totalGrams += grams;
            if (initialVerified) totalVerifiedGrams += grams;
        }
        emit Registered(id, certId, custodianName, custodianAddr, vaultId, serial, grams,
markActive, initialVerified);
    }

    function setCustodian(uint256 id, string calldata name, address addr) external
onlyRole(MANAGER_ROLE) {
        GoldAsset storage a = assets[id];
        require(bytes(a.certId).length > 0, "no asset");
        require(addr != address(0), "addr");
        bool wasVerified = a.verified;
        a.custodianName = name;
        a.custodianAddr = addr;
        a.verified = _isCustodianValid(addr);
        if (a.active) {
            if (wasVerified && !a.verified) totalVerifiedGrams -= a.grams;
            if (!wasVerified && a.verified) totalVerifiedGrams += a.grams;
        }
        _enforceAllocationCap();
        emit CustodianChanged(id, name, addr);
    }

    function setActive(uint256 id, bool active_) external onlyRole(MANAGER_ROLE) {
        GoldAsset storage a = assets[id];
        require(bytes(a.certId).length > 0, "no asset");
        if (a.active == active_) return;
        a.active = active_;
        if (active_) {
            totalGrams += a.grams;
            if (a.verified) totalVerifiedGrams += a.grams;
        } else {
            totalGrams -= a.grams;
            if (a.verified) totalVerifiedGrams -= a.grams;
        }
        _enforceAllocationCap();
        emit Updated(id, a.grams, a.active);
    }

    function updateGrams(uint256 id, uint256 newGrams) external
onlyRole(CUSTODIAN_ROLE) {
        GoldAsset storage a = assets[id];
        require(bytes(a.certId).length > 0, "no asset");
        require(newGrams > 0, "grams=0");

        if (a.active) {
            if (newGrams > a.grams) {
                uint256 delta = newGrams - a.grams;
                totalGrams += delta;
                if (a.verified) totalVerifiedGrams += delta;
            } else if (newGrams < a.grams) {
                uint256 delta2 = a.grams - newGrams;
                totalGrams -= delta2;
                if (a.verified) totalVerifiedGrams -= delta2;
            }
        }

        a.grams = newGrams;
        _enforceAllocationCap();
        emit Updated(id, a.grams, a.active);
    }

    // audits and verification

    function verifyAsset(uint256 id, bool verified, bytes32 auditCid) external
onlyRole(AUDITOR_ROLE) {
        GoldAsset storage a = assets[id];
        require(bytes(a.certId).length > 0, "no asset");
        bool wasVerified = a.verified;

        // allow auditor to set verified true, or let DID verification decide for false
        a.verified = verified ? true : _isAuditorValid(msg.sender);
        a.lastAuditAt = uint64(block.timestamp);
        if (auditCid != bytes32(0)) a.lastAuditCid = auditCid;

        if (a.active) {
            if (wasVerified && !a.verified) totalVerifiedGrams -= a.grams;
            if (!wasVerified && a.verified) totalVerifiedGrams += a.grams;
        }

        if (address(analytics) != address(0) && auditCid != bytes32(0)) {
            analytics.publish(id, a.lastAuditAt, auditCid, auditCid);
        }

        _enforceAllocationCap();
        emit Verified(id, a.verified, a.lastAuditAt, a.lastAuditCid);
    }

    // allocations with enumerable labels

    function setAllocation(bytes32 label, uint256 grams) external onlyRole(MANAGER_ROLE) {
        require(label != bytes32(0), "label");
        if (!_labelExists(label)) {
            allocationLabels.push(label);
            allocationIndex[label] = allocationLabels.length; // index plus one
        }
        allocationGrams[label] = grams;
        _enforceAllocationCap();
        emit AllocationSet(label, grams);
    }

    function removeAllocation(bytes32 label) external onlyRole(MANAGER_ROLE) {
        require(_labelExists(label), "no label");
        // swap and pop
        uint256 idx = allocationIndex[label] - 1;
        uint256 last = allocationLabels.length - 1;
        if (idx != last) {
            bytes32 moved = allocationLabels[last];
            allocationLabels[idx] = moved;
            allocationIndex[moved] = idx + 1;
        }
        allocationLabels.pop();
        delete allocationIndex[label];
        delete allocationGrams[label];
        _enforceAllocationCap();
        emit AllocationRemoved(label);
    }

    function getAllocations() external view returns (bytes32[] memory labels, uint256[] memory
gramsList, uint256 totalAllocated) {
        uint256 n = allocationLabels.length;
        labels = new bytes32[](n);
        gramsList = new uint256[](n);
        for (uint256 i = 0; i < n; i++) {
            bytes32 l = allocationLabels[i];
            labels[i] = l;
            gramsList[i] = allocationGrams[l];
            totalAllocated += gramsList[i];
        }
    }

    // views

    function getAsset(uint256 id) external view returns (GoldAsset memory) {
        return assets[id];
    }

    function allocatedTotal() public view returns (uint256 sum) {
        uint256 n = allocationLabels.length;
        for (uint256 i = 0; i < n; i++) {
            sum += allocationGrams[allocationLabels[i]];
        }
    }

    // internal helpers

    function _enforceAllocationCap() internal view {
        require(allocatedTotal() <= totalVerifiedGrams, "over allocated");
    }

    function _labelExists(bytes32 label) internal view returns (bool) {
        return allocationIndex[label] != 0;
    }

    function _isCustodianValid(address who) internal view returns (bool) {
        if (address(did) == address(0) || custodianSchema == bytes32(0)) return true;
        return did.isValid(who, custodianSchema);
    }

    function _isAuditorValid(address who) internal view returns (bool) {
        if (address(did) == address(0) || auditorSchema == bytes32(0)) return true;
        return did.isValid(who, auditorSchema);
    }
}
```

2. File scripts/16_deploy_GoldReserveRegistry.js with this content

```
const { ethers, network } = require("hardhat");
const { saveAddress } = require("./helpers/saveAddress");
const { getAddr } = require("./helpers/getAddr");

async function main() {
  const admin = process.env.ADMIN;
  if (!admin) throw new Error("ADMIN missing");

  const didAddr = getAddr(network.name, "DIDRegistry");
  const analyticsAddr = getAddr(network.name, "AXIOMAnalyticsHub");

  const custodianSchema = process.env.SCHEMA_CUSTODIAN || "0x";
  const auditorSchema = process.env.SCHEMA_AUDITOR || "0x";

  const GoldReserveRegistry = await ethers.getContractFactory("GoldReserveRegistry");
  const reg = await GoldReserveRegistry.deploy(admin);
  await reg.waitForDeployment();
  const addr = await reg.getAddress();
  console.log(`GoldReserveRegistry deployed at ${addr}`);

  // wire DID and Analytics if present
  if (didAddr) {
    const tx1 = await reg.setDidRegistry(didAddr, custodianSchema, auditorSchema);
    await tx1.wait();
    console.log(`DidRegistry wired: ${didAddr}`);
  }

  if (analyticsAddr) {
    const tx2 = await reg.setAnalyticsHub(analyticsAddr);
    await tx2.wait();
    console.log(`AnalyticsHub wired: ${analyticsAddr}`);
  }

  // optional role grants
  const custodian = process.env.GOLD_CUSTODIAN;
  if (custodian) {
    const role = await reg.CUSTODIAN_ROLE();
    await (await reg.grantRole(role, custodian)).wait();
    console.log(`Granted CUSTODIAN_ROLE to ${custodian}`);
  }

  const auditor = process.env.GOLD_AUDITOR;
  if (auditor) {
    const role = await reg.AUDITOR_ROLE();
    await (await reg.grantRole(role, auditor)).wait();
    console.log(`Granted AUDITOR_ROLE to ${auditor}`);
  }

  saveAddress(network.name, "GoldReserveRegistry", addr, {
    did: didAddr,
    analytics: analyticsAddr,
    custodianSchema,
    auditorSchema,
    custodian,
    auditor
  });
}
main().catch((e)=>{ console.error(e); process.exit(1); });
```

3. Update package.json by adding this script entry

```
{
  "scripts": {
    "deploy:gold": "hardhat run scripts/16_deploy_GoldReserveRegistry.js --network peaq"
  }
}
```

4. Update .env template by adding these keys

```
SCHEMA_CUSTODIAN=0x435553544f4449414e5f5631
SCHEMA_AUDITOR=0x41554449544f525f5631
GOLD_CUSTODIAN=0xOptionalCustodianAddress
GOLD_AUDITOR=0xOptionalAuditorAddress
```

5. Notes for integration

Place GoldReserveRegistry admin under your multisig and timelock path. Grant CUSTODIAN_ROLE to your vault partner account and AUDITOR_ROLE to your audit partner. Call setAllocation with a label such as 0x41584d5f524553455256455f5631 that represents AXM reserve version one and a grams value that never exceeds total verified grams. Publish audit reports as CIDs through your analytics hub when available.

End of prompt.