OmniCounter.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import { ILayerZeroEndpointV2, MessagingFee, MessagingReceipt, Origin } from
"@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/ILayerZeroEndpointV2.sol";

import { ILayerZeroComposer } from "@layerzerolabs/lz-evm-protocol-
v2/contracts/interfaces/ILayerZeroComposer.sol";

import { OApp } from "../OApp.sol";

import { OptionsBuilder } from "../libs/OptionsBuilder.sol";

import { OAppPreCrimeSimulator } from "../../precrime/OAppPreCrimeSimulator.sol";

library MsgCodec {
    uint8 internal constant VANILLA_TYPE = 1;
    uint8 internal constant COMPOSED_TYPE = 2;
    uint8 internal constant ABA_TYPE = 3;
    uint8 internal constant COMPOSED_ABA_TYPE = 4;

    uint8 internal constant MSG_TYPE_OFFSET = 0;
    uint8 internal constant SRC_EID_OFFSET = 1;
    uint8 internal constant VALUE_OFFSET = 5;

    function encode(uint8 _type, uint32 _srcEid) internal pure returns (bytes memory) {
        return abi.encodePacked(_type, _srcEid);
    }

    function encode(uint8 _type, uint32 _srcEid, uint256 _value) internal pure returns (bytes
memory) {
        return abi.encodePacked(_type, _srcEid, _value);
    }
```

```
    function msgType(bytes calldata _message) internal pure returns (uint8) {
        return uint8(bytes1(_message[MSG_TYPE_OFFSET:SRC_EID_OFFSET]));
    }

    function srcEid(bytes calldata _message) internal pure returns (uint32) {
        return uint32(bytes4(_message[SRC_EID_OFFSET:VALUE_OFFSET]));
    }

    function value(bytes calldata _message) internal pure returns (uint256) {
        return uint256(bytes32(_message[VALUE_OFFSET:]));
    }
}

contract OmniCounter is ILayerZeroComposer, OApp, OAppPreCrimeSimulator {
    using MsgCodec for bytes;
    using OptionsBuilder for bytes;

    uint256 public count;
    uint256 public composedCount;

    address public admin;
    uint32 public eid;

    mapping(uint32 srcEid => mapping(bytes32 sender => uint64 nonce)) private
maxReceivedNonce;
    bool private orderedNonce;

    // for global assertions
    mapping(uint32 srcEid => uint256 count) public inboundCount;
    mapping(uint32 dstEid => uint256 count) public outboundCount;
```

```
    constructor(address _endpoint, address _delegate) OApp(_endpoint, _delegate) {
        admin = msg.sender;
        eid = ILayerZeroEndpointV2(_endpoint).eid();
    }

    modifier onlyAdmin() {
        require(msg.sender == admin, "only admin");
        _;
    }

    // ------------------------------
    // Only Admin
    function setAdmin(address _admin) external onlyAdmin {
        admin = _admin;
    }

    function withdraw(address payable _to, uint256 _amount) external onlyAdmin {
        (bool success, ) = _to.call{ value: _amount }("");
        require(success, "OmniCounter: withdraw failed");
    }

    // ------------------------------
    // Send
    function increment(uint32 _eid, uint8 _type, bytes calldata _options) external payable {
        //      bytes memory options = combineOptions(_eid, _type, _options);
        _lzSend(_eid, MsgCodec.encode(_type, eid), _options, MessagingFee(msg.value, 0),
payable(msg.sender));
        _incrementOutbound(_eid);
    }
```

```
  // this is a broken function to skip incrementing outbound count
  // so that preCrime will fail
  function brokenIncrement(uint32 _eid, uint8 _type, bytes calldata _options) external payable onlyAdmin {
    //      bytes memory options = combineOptions(_eid, _type, _options);
    _lzSend(_eid, MsgCodec.encode(_type, eid), _options, MessagingFee(msg.value, 0), payable(msg.sender));
  }

  function batchIncrement(
    uint32[] calldata _eids,
    uint8[] calldata _types,
    bytes[] calldata _options
  ) external payable {
    require(_eids.length == _options.length && _eids.length == _types.length, "OmniCounter: length mismatch");

    MessagingReceipt memory receipt;
    uint256 providedFee = msg.value;
    for (uint256 i = 0; i < _eids.length; i++) {
      address refundAddress = i == _eids.length - 1 ? msg.sender : address(this);
      uint32 dstEid = _eids[i];
      uint8 msgType = _types[i];
      //        bytes memory options = combineOptions(dstEid, msgType, _options[i]);
      receipt = _lzSend(
        dstEid,
        MsgCodec.encode(msgType, eid),
        _options[i],
        MessagingFee(providedFee, 0),
        payable(refundAddress)
      );
      _incrementOutbound(dstEid);
```

```
            providedFee -= receipt.fee.nativeFee;
        }
    }

    // ------------------------------
    // View
    function quote(
        uint32 _eid,
        uint8 _type,
        bytes calldata _options
    ) public view returns (uint256 nativeFee, uint256 lzTokenFee) {
        //      bytes memory options = combineOptions(_eid, _type, _options);
        MessagingFee memory fee = _quote(_eid, MsgCodec.encode(_type, eid), _options, false);
        return (fee.nativeFee, fee.lzTokenFee);
    }

    // @dev enables preCrime simulator
    // @dev routes the call down from the OAppPreCrimeSimulator, and up to the OApp
    function _lzReceiveSimulate(
        Origin calldata _origin,
        bytes32 _guid,
        bytes calldata _message,
        address _executor,
        bytes calldata _extraData
    ) internal virtual override {
        _lzReceive(_origin, _guid, _message, _executor, _extraData);
    }

    // ------------------------------
    function _lzReceive(
        Origin calldata _origin,
```

```
    bytes32 _guid,
    bytes calldata _message,
    address /*_executor*/,
    bytes calldata /*_extraData*/
  ) internal override {
    _acceptNonce(_origin.srcEid, _origin.sender, _origin.nonce);
    uint8 messageType = _message.msgType();

    if (messageType == MsgCodec.VANILLA_TYPE) {
      count++;

      ////////////////////////////////// IMPORTANT ///////////////////////////////////
      /// if you request for msg.value in the options, you should also encode it
      /// into your message and check the value received at destination (example below).
      /// if not, the executor could potentially provide less msg.value than you requested
      /// leading to unintended behavior. Another option is to assert the executor to be
      /// one that you trust.
      ////////////////////////////////////////////////////////////////////////////////
      require(msg.value >= _message.value(), "OmniCounter: insufficient value");

      _incrementInbound(_origin.srcEid);
    } else if (messageType == MsgCodec.COMPOSED_TYPE || messageType ==
MsgCodec.COMPOSED_ABA_TYPE) {
      count++;
      _incrementInbound(_origin.srcEid);
      endpoint.sendCompose(address(this), _guid, 0, _message);
    } else if (messageType == MsgCodec.ABA_TYPE) {
      count++;
      _incrementInbound(_origin.srcEid);

      // send back to the sender
```

```
        _incrementOutbound(_origin.srcEid);
        bytes memory options =
OptionsBuilder.newOptions().addExecutorLzReceiveOption(200000, 10);
        _lzSend(
          _origin.srcEid,
          MsgCodec.encode(MsgCodec.VANILLA_TYPE, eid, 10),
          options,
          MessagingFee(msg.value, 0),
          payable(address(this))
        );
      } else {
        revert("invalid message type");
      }
    }

    function _incrementInbound(uint32 _srcEid) internal {
      inboundCount[_srcEid]++;
    }

    function _incrementOutbound(uint32 _dstEid) internal {
      outboundCount[_dstEid]++;
    }

    function lzCompose(
      address _oApp,
      bytes32 /*_guid*/,
      bytes calldata _message,
      address,
      bytes calldata
    ) external payable override {
      require(_oApp == address(this), "!oApp");
```

```
    require(msg.sender == address(endpoint), "!endpoint");

    uint8 msgType = _message.msgType();
    if (msgType == MsgCodec.COMPOSED_TYPE) {
      composedCount += 1;
    } else if (msgType == MsgCodec.COMPOSED_ABA_TYPE) {
      composedCount += 1;

      uint32 srcEid = _message.srcEid();
      _incrementOutbound(srcEid);
      bytes memory options =
OptionsBuilder.newOptions().addExecutorLzReceiveOption(200000, 0);
      _lzSend(
        srcEid,
        MsgCodec.encode(MsgCodec.VANILLA_TYPE, eid),
        options,
        MessagingFee(msg.value, 0),
        payable(address(this))
      );
    } else {
      revert("invalid message type");
    }
  }

  // ------------------------------
  // Ordered OApp
  // this demonstrates how to build an app that requires execution nonce ordering
  // normally an app should decide ordered or not on contract construction
  // this is just a demo
  function setOrderedNonce(bool _orderedNonce) external onlyOwner {
    orderedNonce = _orderedNonce;
```

```
    }

    function _acceptNonce(uint32 _srcEid, bytes32 _sender, uint64 _nonce) internal virtual {
        uint64 currentNonce = maxReceivedNonce[_srcEid][_sender];
        if (orderedNonce) {
            require(_nonce == currentNonce + 1, "OApp: invalid nonce");
        }
        // update the max nonce anyway. once the ordered mode is turned on, missing early nonces will be rejected
        if (_nonce > currentNonce) {
            maxReceivedNonce[_srcEid][_sender] = _nonce;
        }
    }

    function nextNonce(uint32 _srcEid, bytes32 _sender) public view virtual override returns (uint64) {
        if (orderedNonce) {
            return maxReceivedNonce[_srcEid][_sender] + 1;
        } else {
            return 0; // path nonce starts from 1. if 0 it means that there is no specific nonce enforcement
        }
    }

    // TODO should override oApp version with added ordered nonce increment
    // a governance function to skip nonce
    function skipInboundNonce(uint32 _srcEid, bytes32 _sender, uint64 _nonce) public virtual onlyOwner {
        endpoint.skip(address(this), _srcEid, _sender, _nonce);
        if (orderedNonce) {
            maxReceivedNonce[_srcEid][_sender]++;
        }
```

```
    }

    function isPeer(uint32 _eid, bytes32 _peer) public view override returns (bool) {
        return peers[_eid] == _peer;
    }

    // @dev Batch send requires overriding this function from OAppSender because the msg.value
contains multiple fees
    function _payNative(uint256 _nativeFee) internal virtual override returns (uint256 nativeFee) {
        if (msg.value < _nativeFee) revert NotEnoughNative(msg.value);
        return _nativeFee;
    }

    // be able to receive ether
    receive() external payable virtual {}

    fallback() external payable {}
}
```

OmniCounterPreCrime.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import { PreCrime, PreCrimePeer } from "../../precrime/PreCrime.sol";
import { InboundPacket } from "../../precrime/libs/Packet.sol";
import { OmniCounter } from "./OmniCounter.sol";

contract OmniCounterPreCrime is PreCrime {
    struct ChainCount {
        uint32 remoteEid;
```

```
        uint256 inboundCount;
        uint256 outboundCount;
    }

    constructor(address _endpoint, address _counter) PreCrime(_endpoint, _counter) {}

    function buildSimulationResult() external view override returns (bytes memory) {
        address payable payableSimulator = payable(simulator);
        OmniCounter counter = OmniCounter(payableSimulator);
        ChainCount[] memory chainCounts = new ChainCount[](preCrimePeers.length);
        for (uint256 i = 0; i < preCrimePeers.length; i++) {
            uint32 remoteEid = preCrimePeers[i].eid;
            chainCounts[i] = ChainCount(remoteEid, counter.inboundCount(remoteEid),
counter.outboundCount(remoteEid));
        }
        return abi.encode(chainCounts);
    }

    function _preCrime(
        InboundPacket[] memory /** _packets */,
        uint32[] memory _eids,
        bytes[] memory _simulations
    ) internal view override {
        uint32 localEid = _getLocalEid();
        ChainCount[] memory localChainCounts;

        // find local chain counts
        for (uint256 i = 0; i < _eids.length; i++) {
            if (_eids[i] == localEid) {
                localChainCounts = abi.decode(_simulations[i], (ChainCount[]));
                break;
```

```
        }
    }

    // local against remote
    for (uint256 i = 0; i < _eids.length; i++) {
        uint32 remoteEid = _eids[i];
        ChainCount[] memory remoteChainCounts = abi.decode(_simulations[i], (ChainCount[]));
        (uint256 _inboundCount, ) = _findChainCounts(localChainCounts, remoteEid);
        (, uint256 _outboundCount) = _findChainCounts(remoteChainCounts, localEid);
        if (_inboundCount > _outboundCount) {
            revert CrimeFound("inboundCount > outboundCount");
        }
    }
}

function _findChainCounts(
    ChainCount[] memory _chainCounts,
    uint32 _remoteEid
) internal pure returns (uint256, uint256) {
    for (uint256 i = 0; i < _chainCounts.length; i++) {
        if (_chainCounts[i].remoteEid == _remoteEid) {
            return (_chainCounts[i].inboundCount, _chainCounts[i].outboundCount);
        }
    }
    return (0, 0);
}

function _getPreCrimePeers(
    InboundPacket[] memory _packets
) internal view override returns (PreCrimePeer[] memory peers) {
    PreCrimePeer[] memory allPeers = preCrimePeers;
```

```
        PreCrimePeer[] memory peersTmp = new PreCrimePeer[](_packets.length);

        int256 cursor = -1;
        for (uint256 i = 0; i < _packets.length; i++) {
            uint32 srcEid = _packets[i].origin.srcEid;

            // push src eid & peer
            int256 index = _indexOf(allPeers, srcEid);
            if (index >= 0 && _indexOf(peersTmp, srcEid) < 0) {
                cursor++;
                peersTmp[uint256(cursor)] = allPeers[uint256(index)];
            }
        }
        // copy to return
        if (cursor >= 0) {
            uint256 len = uint256(cursor) + 1;
            peers = new PreCrimePeer[](len);
            for (uint256 i = 0; i < len; i++) {
                peers[i] = peersTmp[i];
            }
        }
    }

    function _indexOf(PreCrimePeer[] memory _peers, uint32 _eid) internal pure returns (int256) {
        for (uint256 i = 0; i < _peers.length; i++) {
            if (_peers[i].eid == _eid) return int256(i);
        }
        return -1;
    }
}
```

Core.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.26;


import { OApp, Origin, MessagingFee} from "@layerzerolabs/oapp-evm/contracts/oapp/OApp.sol";

import { MessagingReceipt, MessagingParams } from "@layerzerolabs/lz-evm-protocol-v2/contracts/interfaces/ILayerZeroEndpointV2.sol";

import { OAppOptionsType3 } from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OAppOptionsType3.sol";

import { SafeERC20, IERC20 } from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

import { Ownable } from "@openzeppelin/contracts/access/Ownable.sol";

import "./interface/IGojoWrappedUsd.sol";

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


error InvalidCaller(address caller);

error AlreadyExported(uint256 projectId);

error NotProjectOwner(uint256 projectId, address owner);

error NotEnoughIP(uint256 projectId, uint256 ipConsumption, uint256 availableIP);

error InvalidCrosschainCaller(uint32 eid, bytes32 caller);

error InvalidMsgType();

error LzAltTokenUnavailable();


contract GojoCore is OApp, OAppOptionsType3{

    using SafeERC20 for IERC20;


    struct ConstructorParams{

        address endpoint;

        address gojoWrappedUsd;

        address gojoCoreAIAgent;
```

```
}

struct Project{
   string metadata;
   uint32[] aiAgentsUsed;
   address owner;
   uint256 ipConsumption;
   uint32 generationsCount;
   bool isExported;
}

struct GetQuoteParams{
   uint32 dstEid;
   Project project;
   bytes extraSendOptions;
   bytes extraRelayOptions;
}

struct DomainSpecificAiAgent{
   string metadata;
   address agentAddress;
}

address public gojoWrappedUsd;
address public gojoCoreAIAgent;
bytes32 public gojoStoryCoreAddress;
bytes32 public gojoSignHookAddres;
uint16 public constant SEND = 1;
uint16 public constant SEND_ABC = 2;
uint32 public constant STORY_EID = 40315;
uint32 public constant SKALE_EID = 40273;
```

```
    uint32 public constant POLYGON_EID = 40267;

    uint256 public projectIdCount;
    uint32 public aiAgentsCount;

    mapping(uint256 => Project) public projects;
    mapping(uint32 => DomainSpecificAiAgent) public domainSpecificAiAgents;

    constructor(ConstructorParams memory _params) OApp(_params.endpoint, msg.sender)
Ownable(msg.sender) {
        gojoWrappedUsd = _params.gojoWrappedUsd;
        gojoCoreAIAgent = _params.gojoCoreAIAgent;
        projectIdCount = 0;
    }

    event ProjectCreated(uint256 projectId, string metadata, address owner);
    event GenerationAction(uint256 projectId, uint32[] newAiAgentsUsed, uint256
ipConsumption);
    event DomainSpecificAiAgentAdded(DomainSpecificAiAgent[] agent);
    event MessageSent(bytes32 guid, uint32 dstEid, bytes payload);
    event MessageReceived(bytes32 guid, Origin origin, address executor, bytes payload, bytes
extraData);

    modifier onlyGojoStoryCore(uint32 _eid, bytes32 _sender){
        if(_eid != STORY_EID || _sender != gojoStoryCoreAddress) revert
InvalidCrosschainCaller(_eid, _sender);
        _;
    }

    modifier onlyGojoCoreAiAgent(address _sender){
        if(_sender != gojoCoreAIAgent) revert InvalidCaller(_sender);
        _;
```

```
  }

  function setGojoStoryAddress(address _gojoStoryCoreAddress) external onlyOwner {
    gojoStoryCoreAddress = addressToBytes32(_gojoStoryCoreAddress);
    setPeer(STORY_EID, addressToBytes32(_gojoStoryCoreAddress));
  }

  function setGojoSignHook(address _gojoSignHookAddress) external onlyOwner {
    gojoSignHookAddres = addressToBytes32(_gojoSignHookAddress);
    setPeer(POLYGON_EID, addressToBytes32(_gojoSignHookAddress));
  }

  function setGojoWrappedUsd(address _gojoWrappedUsd) external onlyOwner {
    gojoWrappedUsd = _gojoWrappedUsd;
  }

  function _payNative(uint256 _nativeFee) internal virtual override returns (uint256 nativeFee) {
    address nativeErc20 = endpoint.nativeToken();
    if (nativeErc20 == address(0)) revert LzAltTokenUnavailable();

    IERC20(nativeErc20).safeTransferFrom(msg.sender, address(endpoint), _nativeFee);
  }

  function _lzSend(uint32 _dstEid, bytes memory _message, bytes memory _options,
MessagingFee memory _fee, address _refundAddress) internal virtual override returns
(MessagingReceipt memory receipt) {
    _payNative(_fee.nativeFee);
    if (_fee.lzTokenFee > 0) _payLzToken(_fee.lzTokenFee);
    return endpoint.send(
      MessagingParams(_dstEid, _getPeerOrRevert(_dstEid), _message, _options,
_fee.lzTokenFee > 0),
      _refundAddress
```

```
        );
    }

    function createProject(string memory _metadata) external {
        uint256 projectId = projectIdCount;
        projects[projectId] = Project(_metadata, new uint32[](0), msg.sender, 0, 0, false);
        projectIdCount++;
        emit ProjectCreated(projectId, _metadata, msg.sender);
    }

    function registerGeneration(uint256 _projectId, uint32[] memory newAiAgentsUsed, uint256 _ipConsumption) external onlyGojoCoreAiAgent(msg.sender) {
        if(projects[_projectId].isExported) revert AlreadyExported(_projectId);
        Project storage project = projects[_projectId];
        for(uint i = 0; i < newAiAgentsUsed.length; i++) project.aiAgentsUsed.push(newAiAgentsUsed[i]);
        project.ipConsumption += _ipConsumption;
        project.generationsCount++;
        emit GenerationAction(_projectId, newAiAgentsUsed, _ipConsumption);
    }

    function exportProject(uint256 _projectId, bytes calldata _extraSendOptions, bytes calldata _extraRelayOptions, uint256 _skaleFee) external payable {
        if(projects[_projectId].isExported) revert AlreadyExported(_projectId);
        if(projects[_projectId].owner != msg.sender) revert NotProjectOwner(_projectId, msg.sender);

        uint256 _availaleIP = IGojoWrappedUsd(gojoWrappedUsd).balanceOf(msg.sender);
        if(projects[_projectId].ipConsumption > _availaleIP) revert NotEnoughIP(_projectId, projects[_projectId].ipConsumption, _availaleIP);
        IGojoWrappedUsd(gojoWrappedUsd).exportProject(msg.sender, projects[_projectId].ipConsumption);
```

```
    Project storage project = projects[_projectId];
    project.isExported = true;
    _send(projects[_projectId], _extraSendOptions, _extraRelayOptions, _skaleFee);
}

function _send(
    Project memory _project,
    bytes calldata _extraSendOptions,
    bytes calldata _extraRelayOptions,
    uint256 skaleFee
) public payable {
    bytes memory options = combineOptions(POLYGON_EID, SEND_ABC, _extraSendOptions);
    bytes memory _sendData = abi.encode(_project, _extraRelayOptions);
    MessagingReceipt memory receipt = _lzSend(
        POLYGON_EID,
        _sendData,
        options,
        MessagingFee(skaleFee, 0),
        payable(msg.sender)
    );

    emit MessageSent(receipt.guid, POLYGON_EID, _sendData);
}

function _lzReceive(
    Origin calldata _origin,
    bytes32 _guid,
    bytes calldata _payload,
    address _executor,
    bytes calldata _extraData
) internal override  onlyGojoStoryCore(_origin.srcEid, _origin.sender){
```

```
        DomainSpecificAiAgent[] memory agents = abi.decode(_payload,
(DomainSpecificAiAgent[]));
        for(uint i = 0; i < agents.length; i++){
            domainSpecificAiAgents[aiAgentsCount] = agents[i];
            aiAgentsCount++;
        }
        emit DomainSpecificAiAgentAdded(agents);
        emit MessageReceived(_guid, _origin, _executor, _payload, _extraData);
    }

    function getQuote(
        GetQuoteParams[2] memory _params
    ) public view returns (MessagingFee memory totalFee) {
        for (uint i = 0; i < 2; i++) {
            bytes memory payload = abi.encode(_params[i].project, _params[i].extraRelayOptions);
            bytes memory options = this.combineOptionsHelper(_params[i].dstEid, SEND_ABC,
_params[i].extraSendOptions);
            MessagingFee memory fee = _quote(_params[i].dstEid, payload, options, false);
            totalFee.nativeFee += fee.nativeFee;
        }
    }

    function combineOptionsHelper(uint32 _dstEid, uint16 _msgType, bytes calldata
_extraOptions) external view returns (bytes memory) {
        return combineOptions(_dstEid, _msgType, _extraOptions);
    }

    function addressToBytes32(address _address) public pure returns (bytes32) {
        return bytes32(uint256(uint160(_address)));
    }

    function bytes32ToAddress(bytes32 _bytes32) public pure returns (address) {
```

```
        return address(uint160(uint256(_bytes32)));
    }

}
```

ProtocolRelayer.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.26;

import { OApp, MessagingFee, Origin } from "@layerzerolabs/oapp-evm/contracts/oapp/OApp.sol";
import { OAppOptionsType3 } from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OAppOptionsType3.sol";
import { Ownable } from "@openzeppelin/contracts/access/Ownable.sol";

contract GojoProtocolRelayer is OApp, OAppOptionsType3 {

    string public data = "Nothing received yet";
    uint16 public constant SEND = 1;
    uint16 public constant SEND_ABC = 2;
    uint32 public constant STORY_EID = 40315;
    uint32 public constant SKALE_EID = 40273;
    uint32 public constant POLYGON_EID = 40267;

    event MessageRelayed(string message, uint32 dstEid);
    event MessageReceived(string message, uint32 senderEid, bytes32 sender);
    error InvalidMsgType();

    constructor(address _endpoint, address _owner) OApp(_endpoint, _owner) Ownable(msg.sender) {}
```

```
    function quote(
        uint32 _dstEid,
        string memory _message,
        bytes calldata _options,
        bool _payInLzToken
    ) public view returns (MessagingFee memory fee) {
        bytes memory payload = abi.encode(_message);
        fee = _quote(_dstEid, payload, _options, _payInLzToken);
    }

    function combineOptionsHelper(uint32 _dstEid, uint16 _msgType, bytes calldata
_extraOptions) external view returns (bytes memory) {
        return combineOptions(_dstEid, _msgType, _extraOptions);
    }

    function _lzReceive(
        Origin calldata _origin,
        bytes32,
        bytes calldata message,
        address,
        bytes calldata
    ) internal override {
        (string memory _data, bytes memory relayOptions) = abi.decode(message, (string, bytes));
        data = _data;

        string memory _newMessage = "Source chain said HI!";
        uint32 _destinationEid = _origin.srcEid == SKALE_EID ? STORY_EID : SKALE_EID;

        bytes memory _options= this.combineOptionsHelper(_destinationEid, SEND, relayOptions);
        _lzSend(
            _destinationEid,
```

```
            abi.encode(_newMessage),
            _options,
            MessagingFee(msg.value, 0),
            payable(address(this))
        );
        emit MessageRelayed(_newMessage, _origin.srcEid);
    }


    receive() external payable {}

}
```

RoyaltyRelayer.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.26;

import { OApp, MessagingFee, Origin } from "@layerzerolabs/oapp-evm/contracts/oapp/OApp.sol";
import { OAppOptionsType3 } from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OAppOptionsType3.sol";
import { Ownable } from "@openzeppelin/contracts/access/Ownable.sol";

contract GojoRelayer is OApp, OAppOptionsType3 {

    string public data = "Nothing received yet";
    uint16 public constant SEND = 1;
    uint16 public constant SEND_ABC = 2;
    uint32 public constant STORY_EID = 40315;
    uint32 public constant SKALE_EID = 40273;
    uint32 public constant POLYGON_EID = 40267;
```

```
    event MessageRelayed(string message, uint32 dstEid);
    event MessageReceived(string message, uint32 senderEid, bytes32 sender);
    error InvalidMsgType();

    constructor(address _endpoint, address _owner) OApp(_endpoint, _owner)
Ownable(msg.sender) {}

    function quote(
        uint32 _dstEid,
        string memory _message,
        bytes calldata _options,
        bool _payInLzToken
    ) public view returns (MessagingFee memory fee) {
        bytes memory payload = abi.encode(_message);
        fee = _quote(_dstEid, payload, _options, _payInLzToken);
    }

    function combineOptionsHelper(uint32 _dstEid, uint16 _msgType, bytes calldata
_extraOptions) external view returns (bytes memory) {
        return combineOptions(_dstEid, _msgType, _extraOptions);
    }

    function _lzReceive(
        Origin calldata _origin,
        bytes32,
        bytes calldata message,
        address,
        bytes calldata
    ) internal override {
        (string memory _data, bytes memory relayOptions) = abi.decode(message, (string, bytes));
        data = _data;
```

```
        string memory _newMessage = "Source chain said HI!";
        uint32 _destinationEid = _origin.srcEid == SKALE_EID ? STORY_EID : SKALE_EID;

        bytes memory _options= this.combineOptionsHelper(_destinationEid, SEND, relayOptions);
        _lzSend(
            _destinationEid,
            abi.encode(_newMessage),
            _options,
            MessagingFee(msg.value, 0),
            payable(address(this))
        );
        emit MessageRelayed(_newMessage, _origin.srcEid);
    }


    receive() external payable {}

}
```

StoryCore.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

import { OApp, Origin, MessagingFee, MessagingReceipt} from "@layerzerolabs/oapp-evm/contracts/oapp/OApp.sol";
import { OAppOptionsType3 } from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OAppOptionsType3.sol";
import "./interface/IGojoWrappedUsd.sol";
import "./interface/IGojoStoryUsdWrapper.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import { IGroupingModule } from "./interface/story-protocol-core/IGroupingModule.sol";
```

```
import { ILicensingModule } from "./interface/story-protocol-core/ILicensingModule.sol";
import { IIPAssetRegistry } from "./interface/story-protocol-core/IIPAssetRegistry.sol";
import "./interface/INFT.sol";
import "./interface/IMockEvenSplitGroupPool.sol";

error InvalidCaller(address caller);
error NotProjectOwner(uint256 projectId, address owner);
error InvalidCrosschainCaller(uint32 eid, bytes32 caller);

contract GojoStoryCore is OApp, OAppOptionsType3 {

  struct ConstructorParams{
    address endpoint;
    address gojoCoreAIAgent;
    address gojoRelayer;
  }

  struct Project{
    string metadata;
    uint32[] aiAgentsUsed;
    address owner;
    uint256 ipConsumption;
    uint32 generationsCount;
    bool isExported;
  }

  struct Resource {
    string metadata;
    string ipMetadata;
    address creator;
    uint32 aiAgentId;
```

```
    address ipId;
    uint256 tokenId;
  }

  struct CreateAiAgentsInput {
    string metadata;
    string ipMetadata;
  }

  struct DomainSpecificAiAgent{
    string metadata;
    string ipMetadata;
    address resourceGroupAddress;
    address agentAddress;
  }

  address public gojoCoreAIAgent;
  bytes32 public gojoCoreAddress;
  bytes32 public gojoRelayer;
  address public gojoStoryUsdWrapperAddress;
  IIPAssetRegistry public constant IP_ASSET_REGISTRY =
IIPAssetRegistry(0x1a9d0d28a0422F26D31Be72Edc6f13ea4371E11B);
  IGroupingModule public constant GROUPING_MODULE =
IGroupingModule(0x26Eb59B900FD158396931d2349Fd6B08f0390e76);
  ILicensingModule public constant LICENSING_MODULE =
ILicensingModule(0xd81fd78f557b457b4350cB95D20b547bFEb4D857);
  IMockEvenSplitGroupPool public constant SPLIT_POOL =
IMockEvenSplitGroupPool(0x69e0D5123bc0539a87a9dDcE82E803575e35cbb4);
  address public constant
PIL_LICENSE_TEMPLATE=0x0752f61E59fD2D39193a74610F1bd9a6Ade2E3f9;
  uint256 public constant NON_TRANSFERRABLE_COMMERCIAL_USE_LICENSE = 2;
  uint32 public constant STORY_EID = 40315;
  uint32 public constant SKALE_EID = 40273;
```

```
    uint32 public constant POLYGON_EID = 40267;

    uint256 public resourceIdCount;
    uint32 public aiAgentsCount;
    uint32 public exportedProjectsCount;

    INFT public immutable gojoAiAgentNft;
    INFT public immutable gojoResourceNft;

    mapping(uint256 => Resource) public resources;
    mapping(uint32 => DomainSpecificAiAgent) public domainSpecificAiAgents;
    mapping(uint32 => Project) public exportedProjects;
    mapping(uint32 => uint256) public aiAgentsRevenue;

    constructor(ConstructorParams memory _params) OApp(_params.endpoint, msg.sender)
Ownable(msg.sender) {
        gojoCoreAIAgent = _params.gojoCoreAIAgent;
        resourceIdCount = 0;
        gojoRelayer = addressToBytes32(_params.gojoRelayer);
    }

    event ResourceUploaded(uint256 resourceId, string metadata, string ipMetadata, address
owner, address registeredIp, uint32 aiAgentId, uint256 assetTokenId);
    event DomainSpecificAiAgentAdded(uint32 aiAgentId, DomainSpecificAiAgent agent);
    event MessageSent(bytes32 guid, uint32 dstEid, bytes payload, MessagingFee fee, uint64
nonce);
    event MessageReceived(bytes32 guid, Origin origin, address executor, bytes payload, bytes
extraData);

    modifier onlyGojoRelayer(uint32 _eid, bytes32 _sender){
        if(_eid != POLYGON_EID || _sender != gojoRelayer) revert InvalidCrosschainCaller(_eid,
_sender);
        _;
```

```
    }

    function setGojoCoreAddress(address _gojoCoreAddress) external onlyOwner {
        gojoCoreAddress = addressToBytes32(_gojoCoreAddress);
        setPeer(STORY_EID, addressToBytes32(_gojoCoreAddress));
    }

    function setGojoStoryUsdWrapperAddress(address _gojoStoryUsdWrapperAddress) external onlyOwner {
        gojoStoryUsdWrapperAddress = _gojoStoryUsdWrapperAddress;
    }

    function registerAiAgentIp(string memory metadata, string memory ipMetadata) external returns(address groupId, address aiAgentId) {
        groupId = GROUPING_MODULE.registerGroup(address(SPLIT_POOL));
        uint256 tokenId = gojoAiAgentNft.safeMint(address(this), metadata);

        aiAgentId = IP_ASSET_REGISTRY.register(block.chainid, address(gojoAiAgentNft), tokenId);
        LICENSING_MODULE.attachLicenseTerms(aiAgentId, PIL_LICENSE_TEMPLATE, NON_TRANSFERRABLE_COMMERCIAL_USE_LICENSE);

        address[] memory parentIpIds = new address[](1);
        parentIpIds[0] = aiAgentId;
        uint256[] memory licenseTermIds = new uint256[](1);
        licenseTermIds[0] = NON_TRANSFERRABLE_COMMERCIAL_USE_LICENSE;

        domainSpecificAiAgents[aiAgentsCount] = DomainSpecificAiAgent(metadata, ipMetadata, groupId, aiAgentId);
        LICENSING_MODULE.registerDerivative(groupId, parentIpIds, licenseTermIds, PIL_LICENSE_TEMPLATE, "");
        gojoAiAgentNft.safeTransferFrom(address(this), msg.sender, tokenId);
        emit DomainSpecificAiAgentAdded(aiAgentsCount, domainSpecificAiAgents[aiAgentsCount]);
```

```
        aiAgentsCount++;
    }

    function createResource(uint32 aiAgentId, string memory metadata, string memory ipMetadata) external {
        uint256 tokenId = gojoResourceNft.safeMint(address(this), metadata);
        address groupId = domainSpecificAiAgents[aiAgentId].resourceGroupAddress;

        address resourceId = IP_ASSET_REGISTRY.register(block.chainid, address(gojoResourceNft), tokenId);
        LICENSING_MODULE.attachLicenseTerms(resourceId, PIL_LICENSE_TEMPLATE, NON_TRANSFERRABLE_COMMERCIAL_USE_LICENSE);

        address[] memory ipIds = new address[](1);
        ipIds[0] = resourceId;
        GROUPING_MODULE.addIp(groupId, ipIds);

        gojoResourceNft.safeTransferFrom(address(this), msg.sender, tokenId);
        resources[resourceIdCount] = Resource(metadata, ipMetadata, msg.sender, aiAgentId, resourceId, tokenId);
        emit ResourceUploaded(resourceIdCount, metadata, ipMetadata, msg.sender, resourceId, aiAgentId, tokenId);
        resourceIdCount++;
    }

    function combineOptionsHelper(uint32 _dstEid, uint16 _msgType, bytes calldata _extraOptions) external view returns (bytes memory) {
        return combineOptions(_dstEid, _msgType, _extraOptions);
    }

    function _lzReceive(
        Origin calldata _origin,
        bytes32 _guid,
```

```
        bytes calldata _payload,
        address _executor,
        bytes calldata _extraData
    ) internal override  onlyGojoRelayer(_origin.srcEid, _origin.sender){
        (Project memory project) = abi.decode(_payload, (Project));

        IGojoStoryUsdWrapper(gojoStoryUsdWrapperAddress).unwrap(project.ipConsumption);
        exportedProjects[exportedProjectsCount] = project;

        uint256 aiAgentsUsed = project.aiAgentsUsed.length;
        uint256 revenuePerAgent = project.ipConsumption / aiAgentsUsed;

        for(uint i = 0; i < aiAgentsUsed; i++) aiAgentsRevenue[project.aiAgentsUsed[i]] +=
revenuePerAgent;

        emit MessageReceived(_guid, _origin, _executor, _payload, _extraData);
        exportedProjectsCount++;
    }

    function addressToBytes32(address _address) public pure returns (bytes32) {
        return bytes32(uint256(uint160(_address)));
    }

    function bytes32ToAddress(bytes32 _bytes32) public pure returns (address) {
        return address(uint160(uint256(_bytes32)));
    }

    function onERC721Received(address, address, uint256, bytes calldata) external pure returns
(bytes4) {
        return this.onERC721Received.selector;
    }
```

}

StoryUSDWrapper.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

import { OApp, Origin, MessagingFee, MessagingReceipt} from "@layerzerolabs/oapp-evm/contracts/oapp/OApp.sol";
import { OAppOptionsType3 } from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OAppOptionsType3.sol";

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

error InvalidCrosschainCaller(uint32 eid, bytes32 caller);
error NotEnoughAllowance(uint256 allowance, uint256 amount);
error UnAuthorizedCaller(address caller);

contract GojoStoryUsdWrapper is OApp, OAppOptionsType3 {

    bytes32 public gojoWrappedUsdAddress;
    bytes32 public gojoRelayer;
    address public constant STORY_USD = 0x91f6F05B08c16769d3c85867548615d270C42fC7;
    uint16 public constant SEND = 1;
    uint16 public constant SEND_ABC = 2;
    uint32 public constant STORY_EID = 40315;
    uint32 public constant SKALE_EID = 40273;
    uint32 public constant POLYGON_EID = 40267;
```

```
    constructor(address _endpoint) OApp(_endpoint, msg.sender) Ownable(msg.sender) {


    }


    event MessageSent(bytes32 guid, uint256 amount);

    event MessageReceived(bytes32 guid, Origin origin, address executor, bytes payload, bytes
extraData);


    modifier onlyGojoRelayer(uint32 _eid, bytes32 _sender){

        if(_eid != POLYGON_EID || _sender != gojoRelayer) revert InvalidCrosschainCaller(_eid,
_sender);

        _;

    }


    function setGojoWrappedUsdAddress(address _gojoWrappedUsdAddress) external
onlyOwner {

        gojoWrappedUsdAddress = addressToBytes32(_gojoWrappedUsdAddress);

        setPeer(SKALE_EID, addressToBytes32(_gojoWrappedUsdAddress));

    }


    function setGojoRelayer(address _gojoRelayer) external onlyOwner {

        gojoRelayer = addressToBytes32(_gojoRelayer);

        setPeer(STORY_EID, gojoRelayer);

    }


    function bridgeToSKALE(uint256 _amount, bytes calldata _extraSendOptions, bytes calldata
_extraRelayOptions) external payable {

        uint256 allowance = IERC20(STORY_USD).allowance(msg.sender, address(this));

        if(allowance < _amount) revert NotEnoughAllowance(allowance, _amount);

        IERC20(STORY_USD).transferFrom(msg.sender, address(this), _amount);

        _send(_amount, _extraSendOptions, _extraRelayOptions);

    }
```

```
function _send(
    uint256 amount,
    bytes calldata _extraSendOptions,
    bytes calldata _extraRelayOptions
) internal {
    bytes memory options = combineOptions(POLYGON_EID, SEND_ABC, _extraSendOptions);

    MessagingReceipt memory receipt=_lzSend(
        POLYGON_EID,
        abi.encode(msg.sender, amount, _extraRelayOptions),
        options,
        MessagingFee(msg.value, 0),
        payable(msg.sender)
    );

    emit MessageSent(receipt.guid, amount);
}

function _lzReceive(
    Origin calldata _origin,
    bytes32 _guid,
    bytes calldata _payload,
    address _executor,
    bytes calldata _extraData
) internal override  onlyGojoRelayer(_origin.srcEid, _origin.sender){
    (address receiver, uint256 amount) = abi.decode(_payload, (address, uint256));

    IERC20(STORY_USD).transferFrom(address(this), receiver, amount);
    emit MessageReceived(_guid, _origin, _executor, _payload, _extraData);
}
```

```
    function quote(
        uint256 tokenAmount,
        bytes calldata _extraSendOptions,
        bytes calldata _extraRelayOptions,
        bool _payInLzToken
    ) public view returns (MessagingFee memory fee) {
        bytes memory payload = abi.encode(msg.sender, tokenAmount, _extraRelayOptions);
        bytes memory options = combineOptions(POLYGON_EID, SEND_ABC, _extraSendOptions);
        fee = _quote(POLYGON_EID, payload, options, _payInLzToken);
    }

    function addressToBytes32(address _address) public pure returns (bytes32) {
        return bytes32(uint256(uint160(_address)));
    }

    function bytes32ToAddress(bytes32 _bytes32) public pure returns (address) {
        return address(uint160(uint256(_bytes32)));
    }

}
```

WrappedStoryUsd.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

import { OApp, Origin, MessagingFee, MessagingReceipt} from "@layerzerolabs/oapp-evm/contracts/oapp/OApp.sol";

import { OAppOptionsType3 } from "@layerzerolabs/oapp-evm/contracts/oapp/libs/OAppOptionsType3.sol";
```

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

error NotGojoCore(address caller);
error InvalidCrosschainCaller(uint32 eid, bytes32 caller);
error NotEnoughBalance(uint256 balance, uint256 amount);

contract GojoWrappedStoryUSD is ERC20, OApp, OAppOptionsType3 {

    address public gojoCoreAddress;
    bytes32 public gojoRelayer;
    uint16 public constant SEND = 1;
    uint16 public constant SEND_ABC = 2;
    uint32 public constant STORY_EID = 40315;
    uint32 public constant SKALE_EID = 40273;
    uint32 public constant POLYGON_EID = 40267;

    constructor(string memory name, string memory symbol, address _endpoint) ERC20(name,
symbol) OApp(_endpoint, msg.sender) Ownable(msg.sender) {}

    event MessageSent(bytes32 guid, uint256 amount);
    event MessageReceived(bytes32 guid, Origin origin, address executor, bytes payload, bytes
extraData);

    modifier onlyGojoRelayer(uint32 _eid, bytes32 _sender){
        if(_eid != POLYGON_EID || _sender != gojoRelayer) revert InvalidCrosschainCaller(_eid,
_sender);
        _;
    }

    function setGojoCoreAddress(address _gojoCoreAddress) external onlyOwner {
        gojoCoreAddress = _gojoCoreAddress;
```

```
    }

    function setGojoRelayer(address _gojoRelayer) external onlyOwner {
        gojoRelayer = addressToBytes32(_gojoRelayer);
        setPeer(POLYGON_EID, gojoRelayer);
    }

    function exportProject(address from, uint256 amount) external {
        if(msg.sender != gojoCoreAddress) revert NotGojoCore(msg.sender);
        _burn(from, amount);
    }

    function bridgeToStory(uint256 _amount, bytes calldata _extraSendOptions, bytes calldata
_extraRelayOptions) external payable {
        if(balanceOf(msg.sender) < _amount) revert NotEnoughBalance(balanceOf(msg.sender),
_amount);
        _burn(msg.sender, _amount);
        _send(_amount, _extraSendOptions, _extraRelayOptions);
    }

    function _send(
        uint256 amount,
        bytes calldata _extraSendOptions,
        bytes calldata _extraRelayOptions
    ) internal {
        bytes memory options = combineOptions(POLYGON_EID, SEND_ABC, _extraSendOptions);

        MessagingReceipt memory receipt=_lzSend(
            POLYGON_EID,
            abi.encode(msg.sender, amount, _extraRelayOptions),
            options,
            MessagingFee(msg.value, 0),
```

```
            payable(msg.sender)
        );

        emit MessageSent(receipt.guid, amount);
    }

    function _lzReceive(
        Origin calldata _origin,
        bytes32 _guid,
        bytes calldata _payload,
        address _executor,
        bytes calldata _extraData
    ) internal override  onlyGojoRelayer(_origin.srcEid, _origin.sender) {
        (address receiver, uint256 amount) = abi.decode(_payload, (address, uint256));
        _mint(receiver, amount);
        emit MessageReceived(_guid, _origin, _executor, _payload, _extraData);
    }

    function quote(
        uint256 tokenAmount,
        bytes calldata _extraSendOptions,
        bytes calldata _extraRelayOptions,
        bool _payInLzToken
    ) public view returns (MessagingFee memory fee) {
        bytes memory payload = abi.encode(msg.sender, tokenAmount, _extraRelayOptions);
        bytes memory options = combineOptions(POLYGON_EID, SEND_ABC, _extraSendOptions);
        fee = _quote(POLYGON_EID, payload, options, _payInLzToken);
    }

    function combineOptionsHelper(uint32 _dstEid, uint16 _msgType, bytes calldata
_extraOptions) external view returns (bytes memory) {
```

```
        return combineOptions(_dstEid, _msgType, _extraOptions);
    }


    function addressToBytes32(address _address) public pure returns (bytes32) {
        return bytes32(uint256(uint160(_address)));
    }

    function bytes32ToAddress(bytes32 _bytes32) public pure returns (address) {
        return address(uint160(uint256(_bytes32)));
    }


}
```