BLOCKCHAIN INTEGRATION PROPOSAL AND ROADMAP

Prepared By QNNX Protocol Team

ABSTRACT

This roadmap outlines how to integrate blockchain logging into the QNNX platform to securely record key protocol events using smart contracts and optional IPFS storage. It includes architecture, smart contract design, backend integration steps, and deployment strategy

QNNX

1. Objective

This document provides a **technical roadmap** to implement blockchain logging into the QNNX platform, a protocol-layer system based on the Beckn protocol. The goal is to ensure that important events such as /search, /order, and /confirm are recorded on a decentralized ledger for transparency and traceability.

2. Scope of Integration

The blockchain integration will focus on **event-level logging** and **verifiable audit trails** for selected protocol interactions. It will be limited to:

- Search initiated (/search)
- Order placed (/order)
- Order confirmed (/confirm)

Optional events include:

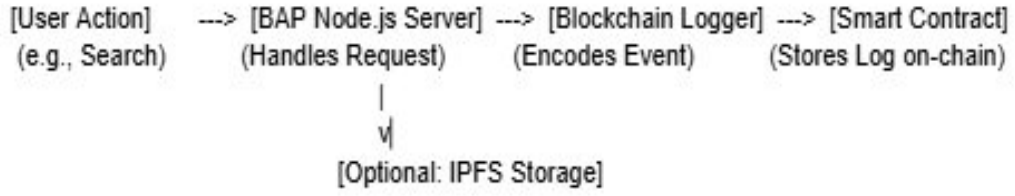
- Order status updates (/status)
- Order cancellation (/cancel)

3. Architecture Overview

Components Involved:

- **Smart Contract:** Logs protocol events on-chain
- **IPFS (Optional):** Stores large payloads off-chain
- **Node.js Service Layer (BAP/BPP):** Triggers contract interactions
- **Blockchain Node/Provider:** e.g., Infura, Alchemy

System Workflow:



4. Technology Stack

Component	Technology
Smart Contracts	Solidity
Blockchain Network	Polygon Mumbai / Ethereum Goerli (testnet)
Deployment Tool	Hardhat / Remix
IPFS Integration	ipfs-http-client
Web3 Library	web3.js or ethers.js
Node.js Runtime	Existing QNNX BAP/BPP services

5. Smart Contract Design

Contract: QnnxLogger.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract QnnxLogger {
    event LogEvent(string eventType, string txId, string userId, uint timestamp);

    function logEvent(string memory eventType, string memory txId, string memory userId)
public {
        emit LogEvent(eventType, txId, userId, block.timestamp);
    }
}
```

Events Logged:

- eventType: e.g., “search”, “order”, “confirm”
- txId: transaction or session ID
- userId: buyer or seller identifier
- timestamp: block time

6. Deployment Steps

A. Smart Contract

1. Write the smart contract using Solidity.
2. Compile the contract using Hardhat or Remix IDE.
3. Deploy to a testnet (Polygon Mumbai or Ethereum Goerli).
4. Save the deployed contract address and ABI for backend integration.

B. Backend Integration (Node.js)

1. Install web3:

```
npm install web3
```

2. Create a utility file BlockchainLogger.js:

```
const Web3 = require('web3');
const abi = require('./QnnxLoggerABI.json');
const contractAddress = '<deployed_address>';
const web3 = new Web3('<Infura/Alchemy_URL>');
const contract = new web3.eth.Contract(abi, contractAddress);

async function logEvent(eventType, txId, userId) {
  const account = web3.eth.accounts.privateKeyToAccount(process.env.PRIVATE_KEY);
  const tx = contract.methods.logEvent(eventType, txId, userId);
  const gas = await tx.estimateGas({ from: account.address });
  const data = tx.encodeABI();

  const signedTx = await web3.eth.accounts.signTransaction({
    to: contractAddress,
    data,
    gas
  }, account.privateKey);
```

```
  return web3.eth.sendSignedTransaction(signedTx.rawTransaction);
}

module.exports = { logEvent };
```

3. From each relevant route (/search, /order, /confirm) in BAP/BPP, call logEvent() with correct parameters.

7. IPFS Integration (Optional)

1. Install IPFS client:

```
npm install ipfs-http-client
```

2. Upload payload to IPFS:

```
const { create } = require('ipfs-http-client');
const ipfs = create({ url: 'https://ipfs.io' });

async function uploadPayload(data) {
   const result = await ipfs.add(JSON.stringify(data));
   return result.path; // Store hash in blockchain
}
```

8. Security Considerations

- Store private keys securely (e.g., in environment variables or vaults).
- Never expose keys in code.
- Use proper gas estimation and retry logic to prevent failed transactions.
- Use testnets for testing; deploy to mainnet only after validation.

9. Testing Checklist

Test smart contract deployment and event logging for /search, /order, and /confirm. Verify logs on a testnet explorer, ensure backend logging works reliably, and confirm private key security. Optionally, test IPFS uploads and transaction retries.

10. Future Improvements

- Add encryption before uploading payloads to IPFS
- Add support for signed messages or verifiable credentials
- Integrate with analytics dashboards (e.g., TheGraph)
- Enforce role-based access in smart contract

11. Conclusion

This document offers a practical and detailed technical roadmap to integrate blockchain into the QNNX platform. By logging select protocol-layer events on a decentralized blockchain, we enhance auditability, trust, and integrity. The outlined steps can be implemented iteratively, starting with testnet deployments and gradually scaling up to production-ready blockchain logging.