Multi-Chain Payment Platform - Deployment and Integration Guide

Overview

This guide covers the deployment and integration of the multi-chain payment platform that supports both Stellar (XLM) and Ethereum L2 (USDC) payments.

Architecture Components

1. Frontend Interface

- **Technology**: React with Tailwind CSS
- **Features**:
  - JSON-based payment data input
  - Multi-chain wallet integration
  - Modal-based address selection
  - Real-time payment processing

2. Stellar Soroban Smart Contract

- **Network**: Stellar Blockchain
- **Language**: Rust
- **Features**:
  - XLM native payments
  - Token payments (USDC on Stellar)
  - Multi-address authorization
  - Fee management
  - Payment tracking

3. Ethereum L2 Smart Contract

- **Network**: Avalanche C-Chain
- **Language**: Solidity
- **Features**:
  - USDC ERC-20 payments
  - Batch payment processing
  - Business configuration

- Refund functionality
- Emergency controls

Prerequisites

Development Environment

bash

```bash
# Node.js and npm
node --version  # v18+
npm --version   # v9+

# Stellar CLI (for Soroban)
curl -L https://github.com/stellar/stellar-cli/releases/latest/download/stellar-cli-x86_64-unknown-linux-gnu.tar.gz
sudo mv stellar /usr/local/bin/

# Hardhat (for Ethereum deployment)
npm install -g hardhat
```

Network Configuration

Stellar Testnet

bash

```bash
# Configure Stellar CLI for testnet
stellar config network --network testnet --rpc-url https://soroban-testnet.stellar.org:443 --network-passphrase "T
```

Avalanche Fuji Testnet

javascript

```javascript
// hardhat.config.js
module.exports = {
  networks: {
    fuji: {
      url: 'https://api.avax-test.network/ext/bc/C/rpc',
      chainId: 43113,
      accounts: ['YOUR_PRIVATE_KEY']
    }
  }
};
```

Deployment Steps

1. Deploy Soroban Contract (Stellar)

bash

```bash
# Build the contract
stellar contract build

# Deploy to testnet
stellar contract deploy \
  --wasm target/wasm32-unknown-unknown/release/payment_contract.wasm \
  --source-account YOUR_STELLAR_ACCOUNT \
  --network testnet

# Initialize contract
stellar contract invoke \
  --id CONTRACT_ID \
  --source-account YOUR_STELLAR_ACCOUNT \
  --network testnet \
  -- initialize \
  --admin YOUR_STELLAR_ACCOUNT \
  --authorized_addresses '["ADDR1", "ADDR2", "ADDR3"]'
```

2. Deploy Ethereum Contract (Avalanche)

bash

```bash
# Install dependencies
npm install @openzeppelin/contracts hardhat @nomiclabs/hardhat-ethers

# Deploy contract
npx hardhat run scripts/deploy.js --network fuji
```

Deployment Script (deploy.js)

javascript

```
const { ethers } = require("hardhat");

async function main() {
  // USDC token address on Avalanche Fuji
  const USDC_ADDRESS = "0x5425890298aed601595a70AB815c96711a31Bc65";

  const USDCPaymentProcessor = await ethers.getContractFactory("USDCPaymentProcessor");
  const contract = await USDCPaymentProcessor.deploy(USDC_ADDRESS);

  await contract.deployed();
  console.log("Contract deployed to:", contract.address);

  // Configure authorized addresses
  const authorizedAddresses = [
    "0x742d35Cc6634C0532925a3b8D400a6ff5E3b0e4b",
    "0x8ba1f109551bD432803012645Hac136c22C3BA6",
    "0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984"
  ];

  for (const addr of authorizedAddresses) {
    await contract.setAuthorizedAddress(addr, true);
    console.log(`Authorized address: ${addr}`);
  }
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

3. Frontend Integration

Wallet Integration Setup

javascript

```
// Install required packages
npm install @stellar/wallet-kit @stellar/stellar-sdk @avalabs/avalanche-wallet-sdk

// Stellar Wallet Kit integration
import { WalletKit } from '@stellar/wallet-kit';

const walletKit = new WalletKit({
  network: 'testnet',
  selectedWallet: 'freighter'
});

// Web3 integration for Avalanche
import { ethers } from 'ethers';

const provider = new ethers.providers.JsonRpcProvider(
  'https://api.avax-test.network/ext/bc/C/rpc'
);
```

Payment Processing Functions

javascript

```javascript
// Stellar XLM Payment
async function processXLMPayment(address, amount, paymentData) {
  try {
    const { publicKey } = await walletKit.getPublicKey();

    // Build transaction
    const transaction = new StellarSdk.TransactionBuilder(account, {
      fee: await server.fetchBaseFee(),
      networkPassphrase: StellarSdk.Networks.TESTNET
    })
    .addOperation(StellarSdk.Operation.payment({
      destination: address,
      asset: StellarSdk.Asset.native(),
      amount: amount.toString()
    }))
    .setTimeout(180)
    .build();

    // Sign and submit
    const signedTransaction = await walletKit.sign(transaction);
    const result = await server.submitTransaction(signedTransaction);

    return result;
  } catch (error) {
    console.error('XLM payment failed:', error);
    throw error;
  }
}

// USDC Payment on Avalanche
async function processUSDCPayment(address, amount, paymentData) {
  try {
    const provider = new ethers.providers.Web3Provider(window.ethereum);
    const signer = provider.getSigner();

    // Contract instance
    const contract = new ethers.Contract(
      CONTRACT_ADDRESS,
      CONTRACT_ABI,
      signer
    );

    // Process payment
    const tx = await contract.processPayment(
      address,
      ethers.utils.parseUnits(amount.toString(), 6), // USDC has 6 decimals
      paymentData.businessName,
      paymentData.customerName,
      paymentData.orderId
    );

    const receipt = await tx.wait();
    return receipt;
  } catch (error) {
    console.error('USDC payment failed:', error);
    throw error;
  }
}
```

JSON Configuration Format

Payment Data Structure

json

```json
{
  "amount": 150.00,
  "currency": "USD",
  "stellarAddresses": [
    "GCKFBEIYTKP5RDBZ7QVRHKK5GFTYUXD5WFJE3DFXDGF3HDVYGRRHIKMR",
    "GADTMGF3XDZXGQJZF7VQXHWSYKQRQ3VBKXMDGRQXFKWYLXZM4QKJHKFM",
    "GAKBPBDMKQRQXFKWYLXZM4QKJHKFMGADTMGF3XDZXGQJZF7VQXHWSYKQ"
  ],
  "usdcAddresses": [
    "0x742d35Cc6634C0532925a3b8D400a6ff5E3b0e4b",
    "0x8ba1f109551bD432803012645Hac136c22C3BA6",
    "0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984"
  ],
  "customerName": "John Doe",
  "businessName": "TechStore Solutions",
  "description": "Premium Software License",
  "orderId": "ORD-2025-001",
  "metadata": {
    "productId": "SOFT-001",
    "licenseType": "Enterprise",
    "validityPeriod": "1 year"
  }
}
```

Security Considerations

1. Address Validation

javascript

```javascript
// Stellar address validation
function isValidStellarAddress(address) {
  try {
    StellarSdk.StrKey.decodeEd25519PublicKey(address);
    return true;
  } catch (error) {
    return false;
  }
}

// Ethereum address validation
function isValidEthereumAddress(address) {
  return ethers.utils.isAddress(address);
}
```

2. Amount Validation

javascript

```javascript
function validatePaymentAmount(amount, currency) {
  if (typeof amount !== 'number' || amount <= 0) {
    throw new Error('Invalid payment amount');
  }

  if (currency === 'USD' && amount > 10000) {
    throw new Error('Amount exceeds maximum limit');
  }

  return true;
}
```

3. Transaction Monitoring

javascript

```
// Monitor Stellar transactions
async function monitorStellarPayment(txHash) {
  try {
    const transaction = await server.transactions()
      .transaction(txHash)
      .call();

    return {
      status: transaction.successful ? 'completed' : 'failed',
      hash: transaction.hash,
      timestamp: transaction.created_at
    };
  } catch (error) {
    return { status: 'pending' };
  }
}

// Monitor Ethereum transactions
async function monitorEthereumPayment(txHash) {
  try {
    const receipt = await provider.getTransactionReceipt(txHash);

    return {
      status: receipt.status === 1 ? 'completed' : 'failed',
      hash: receipt.transactionHash,
      blockNumber: receipt.blockNumber,
      gasUsed: receipt.gasUsed.toString()
    };
  } catch (error) {
    return { status: 'pending' };
  }
}
```

Testing

Unit Tests for Contracts

bash

```bash
# Stellar Soroban tests
stellar test

# Ethereum contract tests
npx hardhat test
```

Integration Tests

javascript

```javascript
// Test payment flow
describe('Payment Integration', () => {
  it('should process XLM payment successfully', async () => {
    const paymentData = {
      amount: 100,
      recipient: 'GCKFBEIYTKP5RDBZ7QVRHKK5GFTYUXD5WFJE3DFXDGF3HDVYGRRHIKMR',
      businessName: 'Test Business',
      customerName: 'Test Customer',
      orderId: 'TEST-001'
    };

    const result = await processXLMPayment(
      paymentData.recipient,
      paymentData.amount,
      paymentData
    );

    expect(result.successful).toBe(true);
  });

  it('should process USDC payment successfully', async () => {
    const paymentData = {
      amount: 100,
      recipient: '0x742d35Cc6634C0532925a3b8D400a6ff5E3b0e4b',
      businessName: 'Test Business',
      customerName: 'Test Customer',
      orderId: 'TEST-002'
    };

    const result = await processUSDCPayment(
      paymentData.recipient,
      paymentData.amount,
      paymentData
    );

    expect(result.status).toBe(1);
  });
});
```

Monitoring and Analytics

Transaction Tracking

javascript

```javascript
// Create transaction tracking system
class PaymentTracker {
  constructor() {
    this.transactions = new Map();
  }

  trackPayment(paymentId, network, txHash) {
    this.transactions.set(paymentId, {
      network,
      txHash,
      timestamp: Date.now(),
      status: 'pending'
    });
  }

  updatePaymentStatus(paymentId, status) {
    const payment = this.transactions.get(paymentId);
    if (payment) {
      payment.status = status;
      payment.updatedAt = Date.now();
    }
  }

  getPaymentHistory() {
    return Array.from(this.transactions.values());
  }
}
```

Error Handling and Logging

javascript

```
class PaymentLogger {
  static logPaymentAttempt(paymentData) {
    console.log(`Payment attempt: ${JSON.stringify(paymentData)}`);
  }

  static logPaymentSuccess(paymentId, txHash) {
    console.log(`Payment successful: ${paymentId} - ${txHash}`);
  }

  static logPaymentError(paymentId, error) {
    console.error(`Payment failed: ${paymentId} - ${error.message}`);
  }
}
```

Maintenance and Updates

Contract Upgrades

- Use proxy patterns for upgradeable contracts
- Implement proper migration scripts
- Test upgrades thoroughly on testnets

Performance Optimization

- Monitor gas usage on Ethereum
- Optimize Stellar transaction fees
- Implement caching for frequently accessed data

Security Updates

- Regular security audits
- Monitor for vulnerabilities
- Update dependencies regularly

Support and Troubleshooting

Common Issues

1. **Wallet Connection Failures**: Ensure proper wallet extension installation
2. **Transaction Failures**: Check account balances and network connectivity
3. **Address Validation Errors**: Verify address format and network compatibility

Debug Tools

- Stellar Laboratory: https://laboratory.stellar.org/
- Avalanche Explorer: https://testnet.snowtrace.io/
- Browser developer tools for frontend debugging

This comprehensive guide provides everything needed to deploy and integrate the multi-chain payment platform successfully.