Soroban Payment Contract Deployment Guide

Prerequisites

1. **Install Rust and Soroban CLI**

```bash
# Install Rust
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

# Add wasm target
rustup target add wasm32-unknown-unknown

# Install Soroban CLI
cargo install --locked stellar-cli --features opt
```

2. **Verify Installation**

```bash
stellar --version
```

Project Structure

```
payment-contract/
├── Cargo.toml
├── src/
│   └── lib.rs
├── tests/
│   └── test.rs
└── README.md
```

Contract Features

Core Functionality

- **Multi-Address Support**: Accepts 3 authorized addresses for payments
- **Business Management**: Register and manage businesses
- **Fee Handling**: Configurable fee percentages with automatic distribution
- **Payment History**: Track all payments per address
- **XLM Support**: Native Stellar Lumens payments

- **Token Support**: Custom token payments via token contracts
- **Authorization**: Multi-signature support with authorized addresses

Key Functions

1. **initialize(owner, default_fee_percentage)**
   - Initialize contract with owner and default fee structure
2. **register_business(name, owner, fee_recipient, fee_percentage)**
   - Register a new business for payment processing
3. **create_payment_request(amount, business_name, description, denomination, authorized_addresses, requester, custom_fee_percentage)**
   - Create new payment request with multiple authorized payers
4. **execute_xlm_payment(payment_id, payer)**
   - Execute XLM payment from authorized address
5. **execute_payment(payment_id, payer, token_address)**
   - Execute token payment from authorized address
6. **get_payment_request(payment_id)**
   - Retrieve payment request details
7. **cancel_payment_request(payment_id, caller)**
   - Cancel pending payment request

Deployment Steps

1. Build the Contract

```bash
# Create new project
mkdir payment-contract
cd payment-contract

# Initialize with the provided code
# Copy lib.rs and Cargo.toml to appropriate locations

# Build the contract
stellar contract build
```

2. Deploy to Testnet

bash

```
# Configure testnet network
stellar network add --global testnet \
  --rpc-url https://soroban-testnet.stellar.org:443 \
  --network-passphrase "Test SDF Network ; September 2015"

# Generate test identity
stellar keys generate alice --network testnet

# Fund the account
stellar keys fund alice --network testnet

# Deploy contract
stellar contract deploy \
  --wasm target/wasm32-unknown-unknown/release/payment_contract.wasm \
  --source alice \
  --network testnet
```

3. Initialize Contract

bash

```
# Initialize with owner and 2.5% default fee (250 basis points)
stellar contract invoke \
  --id <CONTRACT_ID> \
  --source alice \
  --network testnet \
  -- \
  initialize \
  --owner <OWNER_ADDRESS> \
  --default_fee_percentage 250
```

4. Register Business

bash

```
stellar contract invoke \
  --id <CONTRACT_ID> \
  --source alice \
  --network testnet \
  -- \
  register_business \
  --business_name "Example Store" \
  --business_owner <BUSINESS_OWNER_ADDRESS> \
  --fee_recipient <FEE_RECIPIENT_ADDRESS> \
  --fee_percentage 300
```

Integration with Frontend

JavaScript Integration Example

```
javascript
```

```javascript
import {
  Contract,
  SorobanRpc,
  TransactionBuilder,
  Networks,
  BASE_FEE
} from '@stellar/stellar-sdk';

const contractAddress = 'YOUR_CONTRACT_ADDRESS';
const rpcUrl = 'https://soroban-testnet.stellar.org:443';

const server = new SorobanRpc.Server(rpcUrl);
const contract = new Contract(contractAddress);

// Create payment request
async function createPaymentRequest(params) {
  const account = await server.getAccount(params.requester);

  const operation = contract.call(
    'create_payment_request',
    params.amount,
    params.businessName,
    params.description,
    params.denomination,
    params.authorizedAddresses,
    params.requester,
    params.customFeePercentage
  );

  const transaction = new TransactionBuilder(account, {
    fee: BASE_FEE,
    networkPassphrase: Networks.TESTNET,
  })
    .addOperation(operation)
    .setTimeout(300)
    .build();

  // Sign and submit transaction
  // Return payment ID
}

// Execute XLM payment
async function executeXLMPayment(paymentId, payer) {
  const account = await server.getAccount(payer);
  const operation = contract.call(
    'execute_xlm_payment',
    paymentId,
    payer
  );

  const transaction = new TransactionBuilder(account, {
    fee: BASE_FEE,
    networkPassphrase: Networks.TESTNET,
  })
    .addOperation(operation)
    .setTimeout(300)
    .build();

  // Sign and submit transaction
}
```

Testing

Unit Tests

bash

```
# Run tests
cargo test

# Run with logs
cargo test -- --nocapture
```

Integration Testing

bash

```
# Deploy to testnet and run integration tests
stellar contract invoke \
  --id <CONTRACT_ID> \
  --source alice \
  --network testnet \
  -- \
  get_payment_request \
  --payment_id 1234567890
```

Security Considerations

1. **Authorization**: All payment methods require proper authorization
2. **Balance Checks**: Insufficient balance checks prevent failed transactions
3. **Fee Validation**: Fee percentages are capped at 100% (10000 basis points)
4. **Business Validation**: Only active businesses can process payments
5. **Payment Status**: Prevents double-spending and unauthorized modifications

Gas Optimization

- Uses `panic_with_error!` for efficient error handling
- Minimal storage reads/writes
- Batch operations where possible
- Optimized data structures

Monitoring and Maintenance

1. **Event Logging**: Contract logs all major operations
2. **Payment History**: Tracks all payments per address
3. **Business Management**: Activate/deactivate businesses
4. **Fee Updates**: Modify fee structures as needed

Production Deployment

For mainnet deployment:

```bash
# Add mainnet network
stellar network add --global mainnet \
  --rpc-url https://soroban-mainnet.stellar.org:443 \
  --network-passphrase "Public Global Stellar Network ; September 2015"

# Deploy to mainnet
stellar contract deploy \
  --wasm target/wasm32-unknown-unknown/release/payment_contract.wasm \
  --source <MAINNET_ACCOUNT> \
  --network mainnet
```

Support and Documentation

- Soroban Documentation
- Stellar SDK Documentation

- Contract Examples

Version History

- v0.1.0: Initial contract with basic payment functionality
- Support for XLM and token payments
- Multi-signature authorization
- Business management features
- Fee handling and distribution