Omzo.io tech docs

Omzo.io tech docs **1**

1. Overview 2
    - What Omzo.io Offers: 2
    - Technical Advantages: 3
2. Protocol Architecture 3
    - Bitcoin Blockchain (Settlement Layer) 3
    - Midl Network (EVM Integration Layer) 3
    - Runes Protocol (Asset Layer) 4
    - Omzo Smart Contracts (Application Layer) 4
    - Interfaces and Developer Tools (User Layer) 4
3. Midl Network Integration 5
    - Transaction Intents and Lifecycle 5
    - Key Transaction Parameters: 5
    - Wallet Integration and Address Derivation 6
    - Scalability Considerations 6
4. Runes Protocol 6
    - Rune Creation and Minting 6
    - Rune Integration into Midl Ecosystem 7
    - Asset Management and Trading 7
    - Security and Transparency 7
5. Smart Contracts 8
    - Core Smart Contract Functionalities 8
    - Contract Deployment & Management 8
    - Security and Best Practices 9
6. Developers Guide 9
    - Environment Setup 9
    - Using Midl.js Library 10
    - DApp Deployment Examples 10
    - Integrating Frontend Applications 11
    - Best Practices for Developers 11
7. Frontend and Wallet Integration 12
    - Wallet Integration with Leather Wallet 12
    - Midl.js Connectors 12
    - Transaction Management and Intent Execution 13
    - User-Friendly UI and UX Design 13
8. API Reference 14
    - Authentication and Connection 14
    - Asset Operations 14

Liquidity Pool Management 14
Swaps and Transactions 15
Midl Network Operations 15
Data Queries 16
Error Handling and Responses 16
Documentation and Support 17
9. Security and Audits 17
Smart Contract Security 17
Intent-Based Transaction Security (Midl) 17
Asset Security with Runes 18
Wallet and Frontend Security 18
Incident Response and Monitoring 18
10. Example Use-Cases & Tutorials 19
Tutorial 1: Swapping Bitcoin ⇿ Runes 19
Tutorial 2: Minting a Custom Rune 19
Tutorial 3: Adding Liquidity to a Pool 19
Practical Examples for Developers 20
Troubleshooting and FAQ 20

1. Overview

Omzo.io is a decentralized finance (DeFi) platform utilizing Bitcoin's secure blockchain combined with Ethereum-compatible smart contract functionality through Midl technology. The platform enables users and developers to seamlessly create, trade, and manage fungible digital assets (Runes) directly on Bitcoin.

What Omzo.io Offers:

- **Bitcoin-Native DeFi:** Omzo.io brings the sophisticated financial capabilities traditionally associated with Ethereum directly onto Bitcoin, leveraging the inherent security and decentralization of the Bitcoin blockchain.

- **Ethereum Compatibility via Midl:** Using Midl's innovative intent-based transactions, Omzo integrates Ethereum Virtual Machine (EVM) capabilities onto Bitcoin, allowing smart contracts and decentralized applications (DApps) to operate securely and transparently.

- **Runes Integration:** Omzo facilitates the creation and trading of Bitcoin-native tokens (Runes), offering robust DeFi asset management, liquidity provisioning, and secure token swaps without third-party

intermediaries.

Technical Advantages:

- **Security and Decentralization:** Omzo inherits Bitcoin's robust security model, reducing risks associated with traditional DeFi platforms.
- **Scalable Smart Contracts:** Omzo's use of Midl technology ensures efficient execution and verification of Ethereum-compatible smart contracts directly on Bitcoin.
- **Seamless User Experience:** The intuitive user interfaces and comprehensive developer tools simplify interaction and adoption, even for users unfamiliar with blockchain technology.

This documentation provides technical details, implementation guidelines, API references, and practical examples for developers and integrators aiming to leverage Omzo.io's full capabilities for decentralized financial innovation.

---

2. Protocol Architecture

Omzo.io's protocol architecture effectively combines Bitcoin's blockchain with Ethereum-compatible smart contracts via Midl, creating a unified and robust environment for decentralized finance.

Bitcoin Blockchain (Settlement Layer)

At the foundational layer, Bitcoin acts as the immutable ledger and settlement layer for all transactions within Omzo.io. Every transaction—from token creation to swaps—is recorded securely and transparently on Bitcoin's blockchain, providing unmatched decentralization, security, and reliability.

Midl Network (EVM Integration Layer)

Midl serves as the critical integration layer that bridges Bitcoin's transaction network with Ethereum-style smart contracts:

- **Transaction Intents:**
  Midl introduces the innovative concept of "intents," Ethereum-compatible transaction requests explicitly linked to confirmed Bitcoin transactions. Validators execute these intents only after verifying associated Bitcoin

funding transactions, ensuring atomicity, security, and transaction integrity.

- **BTC Transactions and EVM Execution:**
  Each Ethereum-compatible smart contract execution (intent) must have corresponding Bitcoin funding transactions, confirmed by the Bitcoin network, before execution within Midl's EVM environment.

Runes Protocol (Asset Layer)

Runes represent fungible digital assets native to Bitcoin's blockchain. Within Omzo, Runes function equivalently to Ethereum's ERC-20 tokens, supporting seamless integration with Ethereum-compatible smart contracts via Midl:

- **Asset Creation:** Users easily mint Runes specifying parameters like supply, divisibility, and minting capabilities.

- **Asset Management:** Runes can be securely transferred, swapped, and integrated into decentralized applications, providing robust financial flexibility within the Bitcoin ecosystem.

Omzo Smart Contracts (Application Layer)

Omzo's smart contracts handle core functionalities such as:

- **Token Swaps:** Secure, decentralized exchanges between Bitcoin and Runes assets.

- **Liquidity Pools:** Automated market making with transparent asset pricing and decentralized liquidity management.

- **Token Issuance:** Secure creation and management of custom digital tokens within the Bitcoin network.

Interfaces and Developer Tools (User Layer)

Omzo.io provides robust frontend integrations, comprehensive API references, and practical SDKs:

- **Wallet Integration:** Support for popular Bitcoin wallets, ensuring ease of use, secure asset management, and seamless blockchain interactions.

- **Developer APIs:** Extensive APIs and hooks provided via the Midl.js library simplify application development, asset management, and integration

processes.

By layering these components cohesively, Omzo.io delivers a secure, scalable, and comprehensive platform for decentralized finance directly within the Bitcoin ecosystem.

3. Midl Network Integration

The Midl Network provides Omzo.io with Ethereum-compatible smart-contract capabilities on the Bitcoin blockchain, enabling a new level of decentralized finance functionality previously unavailable on Bitcoin.

Transaction Intents and Lifecycle

Midl's integration revolves around its unique **Transaction Intents**, Ethereum-compatible transaction requests explicitly linked to Bitcoin transactions:

- **Intent Creation:**
  When a user initiates an action (e.g., token swap or liquidity provision), Omzo creates an intent containing Ethereum-compatible instructions.

- **Bitcoin Funding Transaction:**
  Users must sign and broadcast a corresponding Bitcoin transaction containing required assets (BTC or Runes). This funding transaction ensures each intent is securely backed by verified Bitcoin assets.

- **Validation and Execution:**
  After the funding transaction receives at least one block confirmation, Midl validators execute the intent, applying Ethereum-compatible logic securely and transparently on Bitcoin.

Key Transaction Parameters:

- **btcAddressType:** Defines Bitcoin address type (0 = Taproot, 1 = Segwit).

- **btcTxHash:** Bitcoin transaction hash linking funding transaction to the intent.

- **PublicKey:** Receiver of transferred assets, ensuring transparency and security.

Example Midl transaction (pseudo-code):

javascript

```
contract.swapToken(
  amount, recipient,
  { btcAddressType: 0, btcTxHash: "98093bf...f606" }
);
```

Wallet Integration and Address Derivation

Midl requires users to manage assets securely through Bitcoin wallets. Wallets like Leather Wallet derive private keys using standard Bitcoin derivation paths (e.g., m/44'/1'/0'/0/0), ensuring compatibility and security.

Omzo frontend applications use specialized wallet connectors provided by Midl.js, simplifying user authentication, asset management, and secure transactions.

Scalability Considerations

Midl addresses Bitcoin's longer block times compared to Ethereum by batching multiple intents. This mechanism optimizes transaction throughput, significantly improving scalability and reducing network congestion, maintaining a smooth and efficient user experience.

---

4. Runes Protocol

Runes are Bitcoin-native fungible assets forming the foundation of Omzo.io's asset ecosystem, providing capabilities analogous to Ethereum's ERC-20 tokens but native to Bitcoin's blockchain.

Rune Creation and Minting

Omzo.io offers straightforward asset creation through its integrated **Rune Etcher** tool:

- **Rune Naming & Symbol:** Users choose a clear token name and single-letter ticker symbol.
- **Supply and Premine:** Creators specify the initial asset supply, automatically deposited into their wallet upon minting.

- **Divisibility:** Default divisibility set to eight decimal places, matching Bitcoin's precision.

- **Mintability:** Optional mintability toggle allowing additional token issuance if required by the project.

Rune Etcher automates the creation process, transparently recording asset parameters on the Bitcoin blockchain.

Rune Integration into Midl Ecosystem

Before trading or liquidity provisioning, Runes must be integrated into Omzo's Midl ecosystem:

- **Adding Rune to TSS Vault:** Users perform a minimal transfer (edict) to Midl's secure Threshold Signature Scheme (TSS) Vault, registering the asset securely within the ecosystem.

- **Rune ID and EVM Address:** Each Rune receives a unique Rune ID (on Bitcoin) and corresponding Ethereum-compatible address (on Midl). These identifiers ensure compatibility with smart contracts, wallets, and decentralized applications.

Example of Rune integration (pseudo-code):

javascript
```
useEdictRune(runeId, amount, multisigAddress);
```

Asset Management and Trading

Once integrated, Runes seamlessly interact with Ethereum-compatible smart contracts running on Omzo.io:

- **Liquidity Pools:** Users deposit Runes and BTC to provide liquidity, earning fees proportionally.

- **Decentralized Swaps:** Users trade Bitcoin and Runes directly within decentralized markets without intermediaries.

- **Cross-Asset Transactions:** Runes integrate with other Ethereum-compatible assets via Midl, enabling cross-chain asset interactions and diverse DeFi strategies.

Security and Transparency

The Runes protocol benefits directly from Bitcoin's secure blockchain infrastructure, providing transparent asset ownership, secure transactions, and robust protection against fraud or malicious manipulation. Users verify Rune transactions transparently via dedicated Omzo and Midl blockchain explorers, enhancing overall trust and usability.

5. Smart Contracts

Omzo.io utilizes Ethereum-compatible smart contracts via Midl to enable decentralized trading, liquidity management, and asset issuance directly on Bitcoin. These smart contracts are built with security, transparency, and efficiency in mind, leveraging established best practices from the Ethereum ecosystem, adapted specifically for Omzo's unique infrastructure.

Core Smart Contract Functionalities

Omzo's smart contracts provide key DeFi functions, including:

- **Token Swaps:** Facilitates decentralized asset exchanges between Bitcoin and Runes without intermediaries. Transactions are atomic, meaning swaps fully execute or not at all, ensuring transaction integrity and user security.

- **Liquidity Pools:** Automatically manage market liquidity using AMM logic (constant-product formula). Users deposit asset pairs into pools, receiving liquidity-provider tokens representing their share. The contracts transparently distribute collected fees back to liquidity providers proportionally.

- **Rune Asset Issuance:** Smart contracts securely handle Rune creation and management. Users interact directly with contracts to mint, issue, and manage custom Bitcoin-native assets, specifying initial supply, divisibility, and minting options clearly and transparently.

Contract Deployment & Management

Developers deploy smart contracts seamlessly using Midl's deployment framework, compatible with popular Ethereum tools (e.g., Hardhat):

Example of a basic smart contract deployment using Midl (hardhat.config.ts):

typescript

```
import "@midl-xyz/hardhat-deploy";

async function main() {
  await hre.midl.initialize();
  const deployer = await hre.midl.getAddress();
  await hre.midl.deploy(
    "OmzoLiquidityPool",
    { args: [deployer, "0xAddress"] },
    {}
  );
  await hre.midl.execute();
}
```

Security and Best Practices

Smart contracts within Omzo adhere to rigorous security standards, including:

- **Regular Audits:** Continuous third-party code audits to proactively identify and resolve potential vulnerabilities.
- **Transparent Codebases:** Fully open-source smart contracts with comprehensive inline documentation, enabling straightforward community and security expert verification.
- **Standardized Libraries:** Utilization of battle-tested Solidity libraries and EVM-compatible patterns to mitigate common vulnerabilities, including reentrancy, front-running, and integer overflow attacks.

---

6. Developers Guide

Omzo.io offers comprehensive resources for developers aiming to build decentralized applications (DApps), integrating Bitcoin-native decentralized finance functionalities via Midl technology.

Environment Setup

Developers typically begin working within a Regtest environment for efficient testing:

- **Wallet Setup:**
  Omzo recommends the Leather Wallet integration for asset management. Developers quickly configure this wallet to interact with the Midl Regtest environment.

- **Connecting to Midl Network:** Configure wallet settings for seamless interaction:

    - Bitcoin API URL: https://mempool.regtest.midl.xyz/api

    - Stacks API URL: https://api.hiro.so

Using Midl.js Library

Omzo simplifies frontend and backend development using the Midl.js library, providing useful hooks and methods:

- **Creating Transaction Intents:**

javascript

```
import { useAddTxIntention, useFinalizeTxIntentions } from 'midl-js';

// Example intent creation:
const intent = useAddTxIntention({
  btcAddressType: 0,
  btcTxHash: 'transaction_hash'
});
```

- **Finalizing and Broadcasting Intents:**

javascript

```
useFinalizeTxIntentions(intentArray);
```

DApp Deployment Examples

Step-by-step DApp deployment example with Hardhat:

1. **Initialize Project:**
   Set up project and dependencies (hardhat + @midl-xyz/hardhat-deploy).

2. **Contract Deployment Script:**

typescript

```
async function deployContracts() {
  await hre.midl.initialize();
  const deployer = await hre.midl.getAddress();
  await hre.midl.deploy(
    "TokenSwap",
    { args: [deployer] },
    {}
  );
  await hre.midl.execute();
}
deployContracts();
```

Integrating Frontend Applications

Omzo's frontend integration is straightforward with Midl's provided React hooks:

- **Wallet Connectors:**
  Utilize Midl.js connectors for seamless Bitcoin wallet integration:

javascript

```
import { LeatherConnector } from 'midl-js/connectors';

const wallet = new LeatherConnector();
await wallet.connect();
```

- **Interacting with Runes (Assets):**
  Simplified Rune transactions using hooks:

javascript

```
import { useEdictRune } from 'midl-js/hooks';

useEdictRune(runeId, amount, multisigAddress);
```

Best Practices for Developers

- Use Omzo-provided standard libraries for common DeFi operations.
- Conduct rigorous testing in Regtest before deploying to mainnet.
- Maintain comprehensive documentation and version control for transparency and collaborative security review.

Omzo.io provides an intuitive development ecosystem with extensive documentation, examples, and support channels, empowering developers to build secure, efficient, and innovative DeFi applications directly on Bitcoin.

7. Frontend and Wallet Integration

Omzo.io simplifies user interactions with decentralized financial services on Bitcoin by providing robust frontend solutions and seamless wallet integrations. By utilizing Midl technology, Omzo ensures users experience intuitive, secure, and familiar interactions, significantly reducing the complexity typically associated with blockchain-based applications.

Wallet Integration with Leather Wallet

Omzo primarily recommends the Leather Wallet for secure management of Bitcoin-native assets (Runes):

- **Setup Leather Wallet:**
    - Download Leather Wallet extension.
    - Add Midl Network using the provided settings:
        - Network Name: Midl Regtest
        - Bitcoin API URL: https://mempool.regtest.midl.xyz/api
        - Stacks API URL: https://api.hiro.so
- **Managing Assets:**

- Users easily deposit, withdraw, and manage Runes and Bitcoin assets directly within their wallets.

- Clear asset balance updates and transaction history enhance transparency and security.

Midl.js Connectors

Frontend developers utilize Midl.js connectors, simplifying user wallet interactions:

Example: Connect Leather Wallet with frontend:

javascript

```
import { LeatherConnector } from 'midl-js/connectors';

const wallet = new LeatherConnector();

async function connectWallet() {
  const connected = await wallet.connect();
  if (connected) {
    const userAddress = await wallet.getAddress();
    console.log('Connected Address:', userAddress);
  } else {
    console.error('Wallet connection failed');
  }
}
```

Transaction Management and Intent Execution

Frontend interfaces easily manage transactions through Midl.js hooks:

- **Creating Transaction Intents:**
  Users initiate actions like swaps and liquidity management through frontend interactions. Midl.js handles intent creation and transaction execution.

javascript

```
import { useAddTxIntention, useFinalizeTxIntentions } from 'midl-js/hooks';

// Create Intent
const intent = useAddTxIntention({
```

```
  btcAddressType: 0,
  btcTxHash: 'btc_transaction_hash'
});

// Finalize Intent
await useFinalizeTxIntentions([intent]);
```

User-Friendly UI and UX Design

Omzo frontend emphasizes ease-of-use:

- Clear interfaces for token swapping, liquidity management, and Rune creation.
- Step-by-step transaction confirmations and detailed error handling.
- Transparent transaction monitoring through built-in explorers.

---

8. API Reference

Omzo.io provides a comprehensive API reference detailing core protocol interactions, asset management functionalities, and Midl-specific operations, enabling developers to integrate efficiently with Omzo's decentralized financial ecosystem.

Authentication and Connection

- **Connect Wallet**
    - Endpoint: `connectWallet(walletConnector)`
    - Usage: Securely connects user wallet for transactions.

Asset Operations

- **Mint New Rune**
    - Endpoint: `mintRune(name, symbol, supply, divisibility, mintable)`

    - Usage: Creates and deploys a new Rune asset on Bitcoin.

- **Rune Transfer**
    - Endpoint: transferRune(runeId, amount, destination)
    - Usage: Transfers specified amount of Rune tokens to another user or address.

Liquidity Pool Management

- **Add Liquidity**
    - Endpoint: addLiquidity(tokenA, tokenB, amountA, amountB)
    - Usage: Deposits specified asset pair into liquidity pool, returning LP tokens.
- **Remove Liquidity**
    - Endpoint: removeLiquidity(poolId, lpTokenAmount)
    - Usage: Withdraws liquidity from specified pool, returning assets proportionally.

Swaps and Transactions

- **Execute Swap**
    - Endpoint: swapTokens(inputToken, outputToken, inputAmount)
    - Usage: Executes decentralized asset swap via Omzo's AMM mechanism.
- **Transaction Status**
    - Endpoint: getTransactionStatus(txHash)
    - Usage: Checks status and confirmation details for given transaction hash.

Midl Network Operations

- **Create Intent**
    - Endpoint: createIntent(contractCall, btcAddressType, btcTxHash)
    - Usage: Creates Ethereum-compatible transaction intent linked to Bitcoin funding transaction.
- **Finalize Intents**
    - Endpoint: finalizeIntents(intentArray)
    - Usage: Validates and executes batch of intents, securely finalizing transaction execution.

Data Queries

- **Rune Balances**
    - Endpoint: getRuneBalances(address)
    - Usage: Retrieves detailed balances for all Runes held at given address.
- **Liquidity Pool Data**
    - Endpoint: getLiquidityPoolInfo(poolId)
    - Usage: Fetches detailed liquidity pool information including total liquidity, asset balances, and LP token distribution.

Error Handling and Responses

All API endpoints return standardized JSON objects containing clear success/error indicators, detailed error codes, and human-readable descriptions to assist frontend handling and user troubleshooting.

Example successful response:

json

```
{
  "success": true,
  "data": {
    "txHash": "0xtransactionhash",
```

```
    "status": "confirmed"
  }
}
```

Example error response:

json

```
{
  "success": false,
  "error": {
    "code": 403,
    "message": "Insufficient liquidity for requested swap."
  }
}
```

Documentation and Support

Complete API reference documentation, interactive examples, and direct developer support channels ensure effective and smooth integration experiences, enabling robust decentralized application development on Omzo.io.

9. Security and Audits

Security is paramount in Omzo.io's decentralized finance ecosystem. Built upon Bitcoin's proven security and Midl's Ethereum-compatible smart contracts, Omzo employs robust measures to safeguard user assets, data integrity, and smart contract reliability.

Smart Contract Security

Omzo follows industry best practices for smart contract security:

- **Regular Code Audits:**
    - All Omzo smart contracts undergo rigorous, periodic security audits by reputable blockchain security firms.
    - Audit results are publicly available, fostering transparency and trust within the Omzo community.
- **Secure Development Practices:**

    - Adoption of secure coding standards from established Ethereum projects (e.g., Uniswap).

    - Usage of widely-accepted Solidity libraries to mitigate common vulnerabilities (e.g., OpenZeppelin).

Intent-Based Transaction Security (Midl)

Midl's intent-based transactions significantly enhance Omzo's security:

- **Atomic Execution:**
  Transactions either fully execute or revert entirely, avoiding partial failures and protecting user funds.

- **Bitcoin Confirmation Requirement:**
  Midl validators only execute intents after corresponding Bitcoin transactions receive confirmed blocks, preventing double-spending and transaction spoofing.

Asset Security with Runes

Runes benefit directly from Bitcoin's secure blockchain:

- Transparent ownership, transaction traceability, and immutability ensured by Bitcoin's public ledger.

- Integration with Threshold Signature Scheme (TSS) vaults for secure Rune management and transfers.

Wallet and Frontend Security

Omzo integrates with secure wallets (Leather Wallet), ensuring users control private keys securely. Frontend applications:

- Employ secure wallet connection mechanisms (e.g., Midl.js connectors).

- Clearly communicate transaction details and confirmation states to prevent user errors or phishing attacks.

Incident Response and Monitoring

- **Real-Time Monitoring:**
    - Omzo employs continuous monitoring tools to detect anomalies, irregular transactions, and potential threats promptly.
- **Rapid Incident Response:**
    - Dedicated security teams and clear escalation procedures allow rapid response to potential vulnerabilities, minimizing risks and damages.
    - Transparent communication channels inform users immediately of security incidents or concerns, maintaining user trust.

---

10. Example Use-Cases & Tutorials

This section provides practical, step-by-step examples demonstrating Omzo.io's primary decentralized finance functionalities, helping developers and users quickly understand and adopt the platform.

Tutorial 1: Swapping Bitcoin ↔ Runes

- **Step-by-Step Process:**
    1. Connect Leather Wallet to Omzo Swap dApp.
    2. Choose Bitcoin as the input asset, desired Rune as the output.
    3. Specify swap amount; confirm the transaction intent.
    4. Sign and broadcast associated Bitcoin funding transaction.
    5. Wait for Bitcoin confirmation; Midl validators execute the intent.
    6. Verify updated balances in wallet and Omzo frontend.

Tutorial 2: Minting a Custom Rune

- **Rune Creation Using Rune Etcher:**

1. Connect wallet to Omzo's Rune Etcher tool.
2. Enter token details: name, symbol, divisibility, supply, mintability.
3. Confirm Rune creation parameters.
4. Sign and confirm creation transaction via wallet.
5. Verify token availability in wallet and Omzo's frontend.

Tutorial 3: Adding Liquidity to a Pool

- **Providing Liquidity:**
    1. Navigate to Omzo liquidity management page.
    2. Select assets (Bitcoin and a specific Rune) for liquidity pool.
    3. Input desired amounts for each asset.
    4. Confirm intent creation, fund via Bitcoin transaction.
    5. Verify liquidity-provider tokens received representing pool ownership.
    6. Track earned fees and manage liquidity through Omzo frontend.

Practical Examples for Developers

- **Deploying Custom Smart Contracts (DApps):**
    - Detailed examples showing contract deployment on Midl using popular tools like Hardhat.
    - Example code snippets, deployment scripts, and best practices.
- **Frontend Integration Examples:**
    - Examples demonstrating wallet integrations, transaction intents, and secure frontend interactions using Midl.js.

Troubleshooting and FAQ

- Clear solutions to common challenges developers face during integration:
    - Transaction intent failures.
    - Rune asset visibility issues.
    - Wallet integration troubleshooting.
- Community support channels (Telegram, Discord, GitHub) for direct developer assistance.