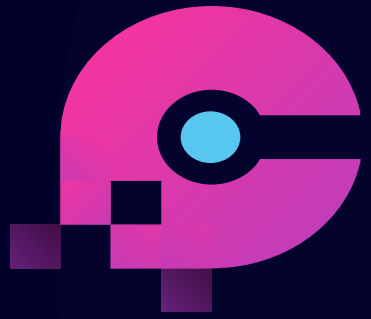


Foti to Monke

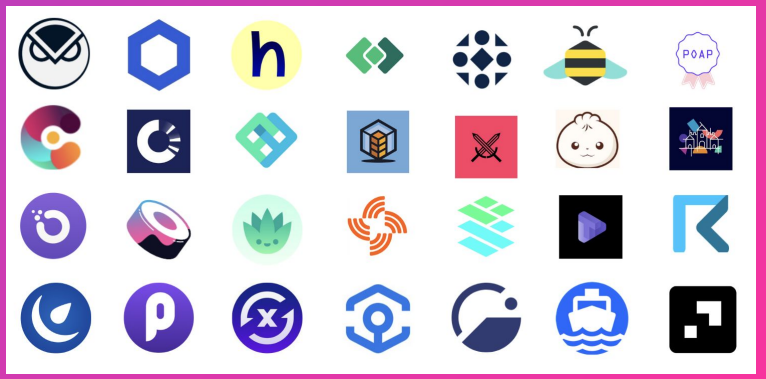
Project 3 – Bridge Revolution
Faster, and Stable, Low transaction Fees

Collaborators: Whitney Turner, Christy Dain, Racim Badsi, Samuel Simon, Dean Kim

Why xDai ?

- Fast transaction times (5 seconds) & low transaction fees
- Digital cash. A stable chain is ideal for real world value exchange where 1 xDai = 1 US Dollar.
- A stable token for transactions & gas fees.
- A green, energy-efficient and ecologically aware blockchain network.
- Permissionless delegated Proof-Of-Stake based consensus with public POSDAO.
- STAKE token for community consensus participation and incentives.
- Wide-ranging Community Support (see xDai Validator Organizations).
- Extreme usability with tools like Burner Wallet & Burner Wallet 2.
- Growing ecosystem designed to support stable person-to-person transactions, micro transactions, conference currencies, community currencies, DeFi, NFTs, DAOs, games and more.
- Full-featured BlockScout Explorer.
- On-chain, decentralized Random Number Generator.
- Smart Contract, DApp & toolset compatibility with other Ethereum-based chains like Ethereum, Ethereum Classic, BSC and others.

xDai
Stable Chian

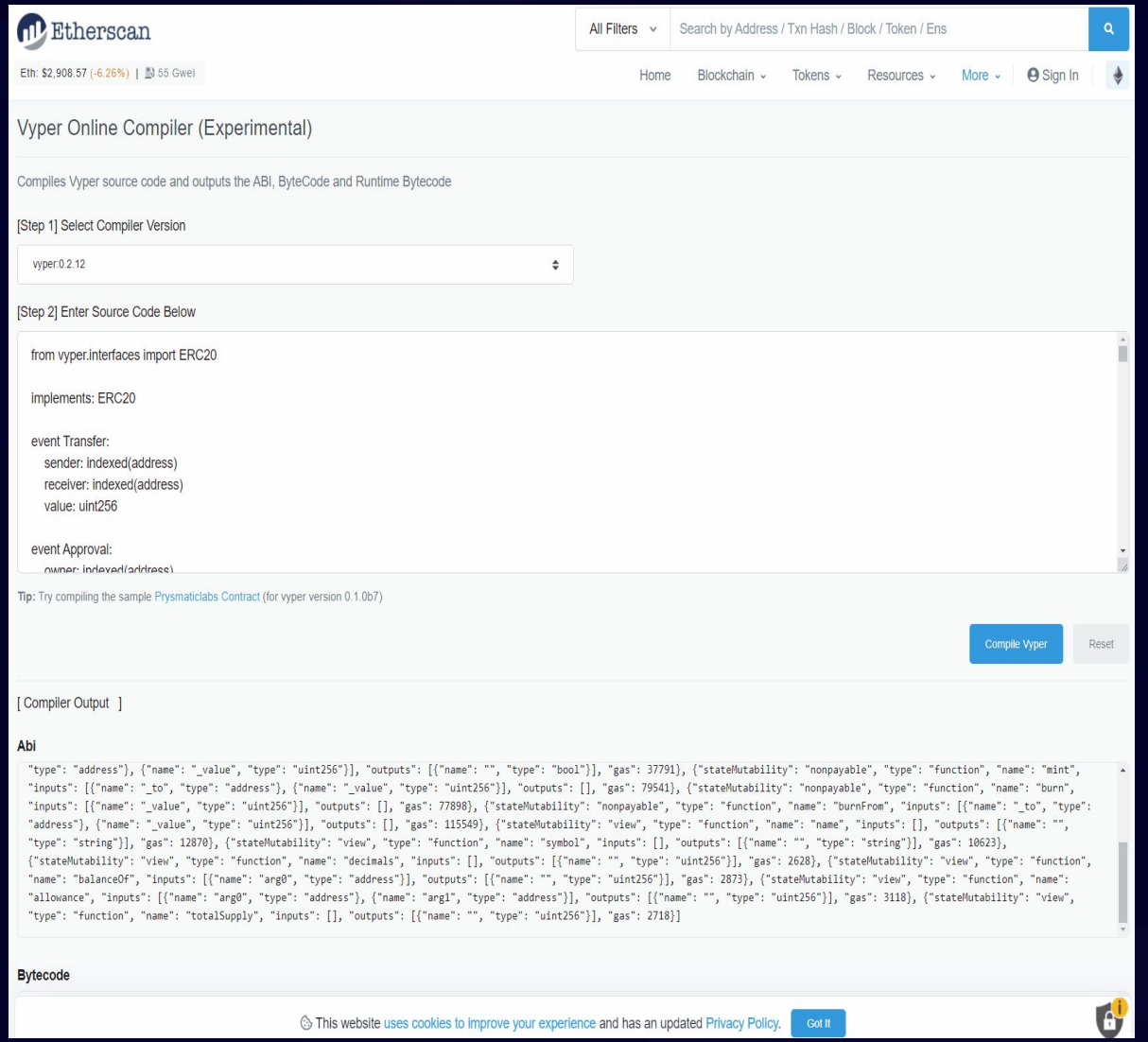


The xDai chain is a stable payments EVM (Ethereum Virtual Machine) blockchain designed for fast and inexpensive transactions. The chain uses a unique dual-token model; xDai is a stable token used for transactions, payments, and fees, and STAKE is a governance token used to support the underlying POSDAO Proof-of-Stake consensus.

The Tools Used to Compile, Deploy and Interact with The Contract

- Online Vyper Compiler (EtherScan)
- MyEtherWallet (Deploy and Interact)
- BlockScout (easy access to the contract details)
- Vyper Language

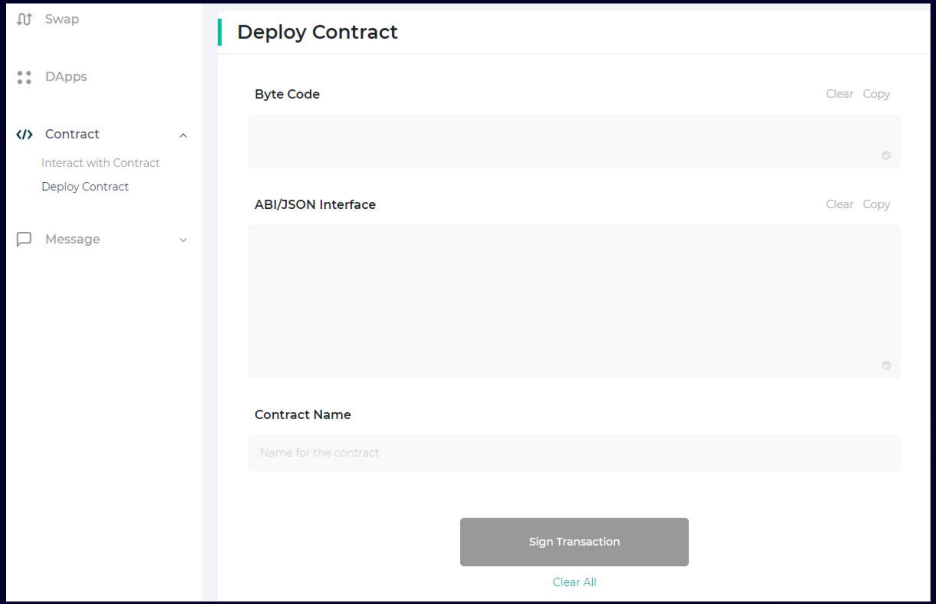
Compiling ERC 20 FOTI and MONKE contract

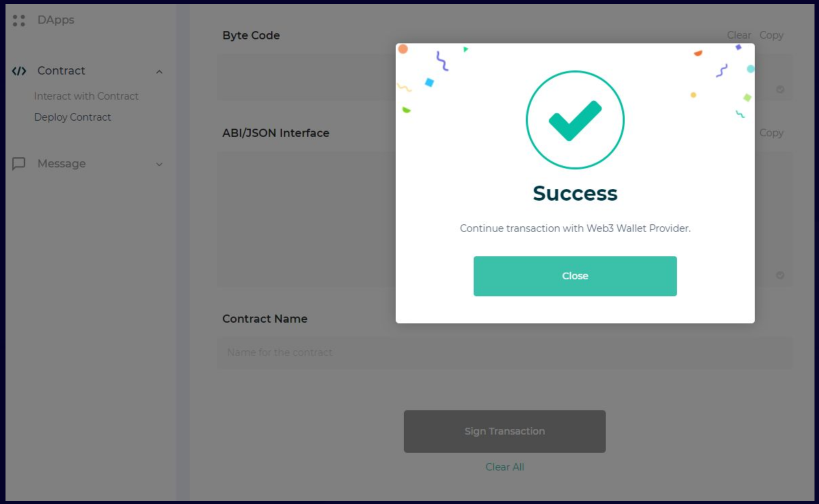


FOTI Contract

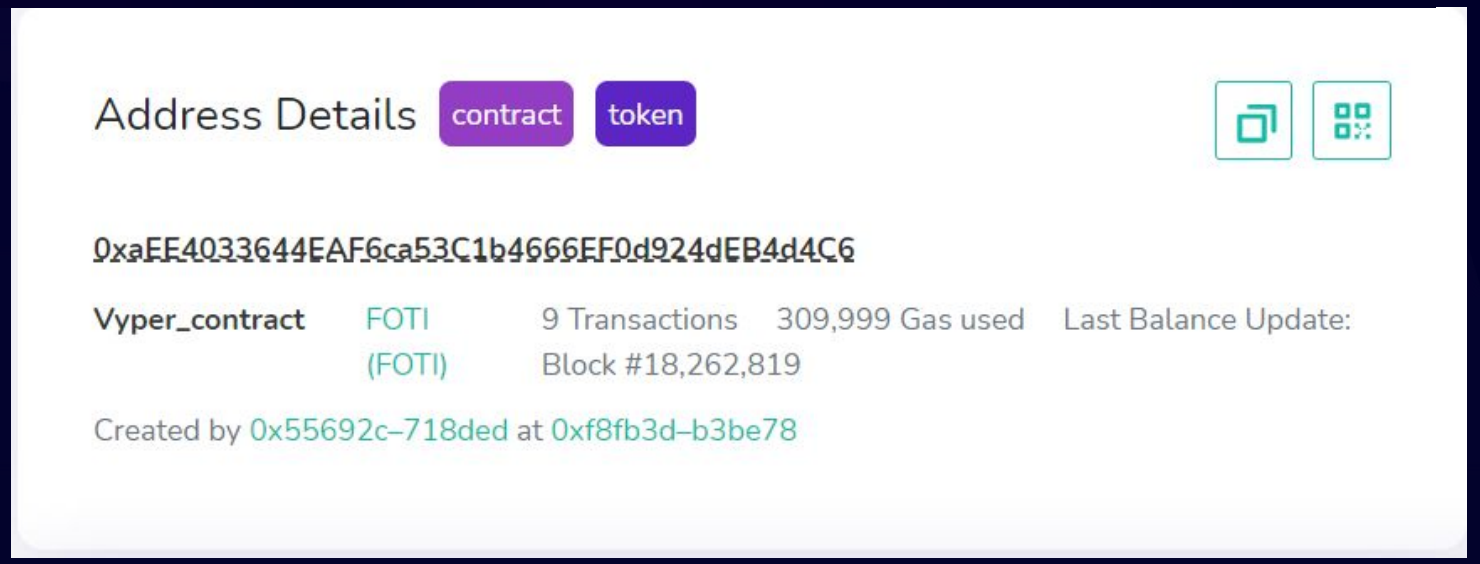
Deploying the ERC 20 FOTI and Monke contract

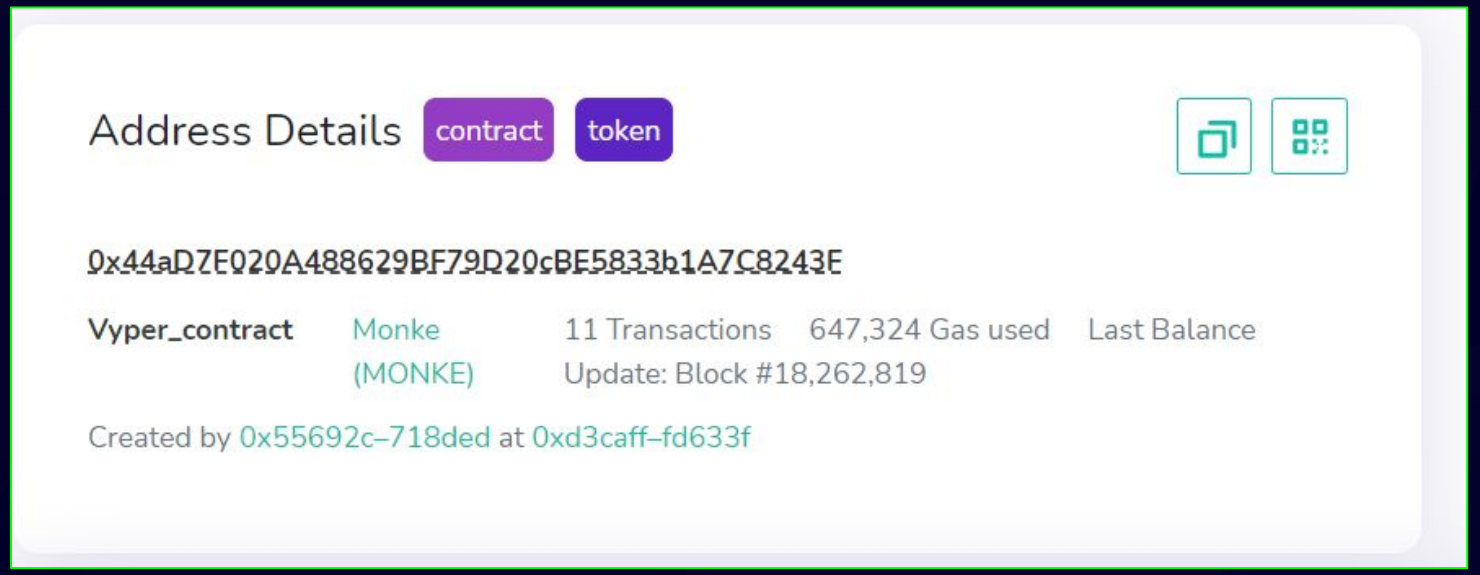
- Copy the Abi and Bytcode from the compiled contract and paste them in the V5.MyEtherWallet deployer and sign transaction





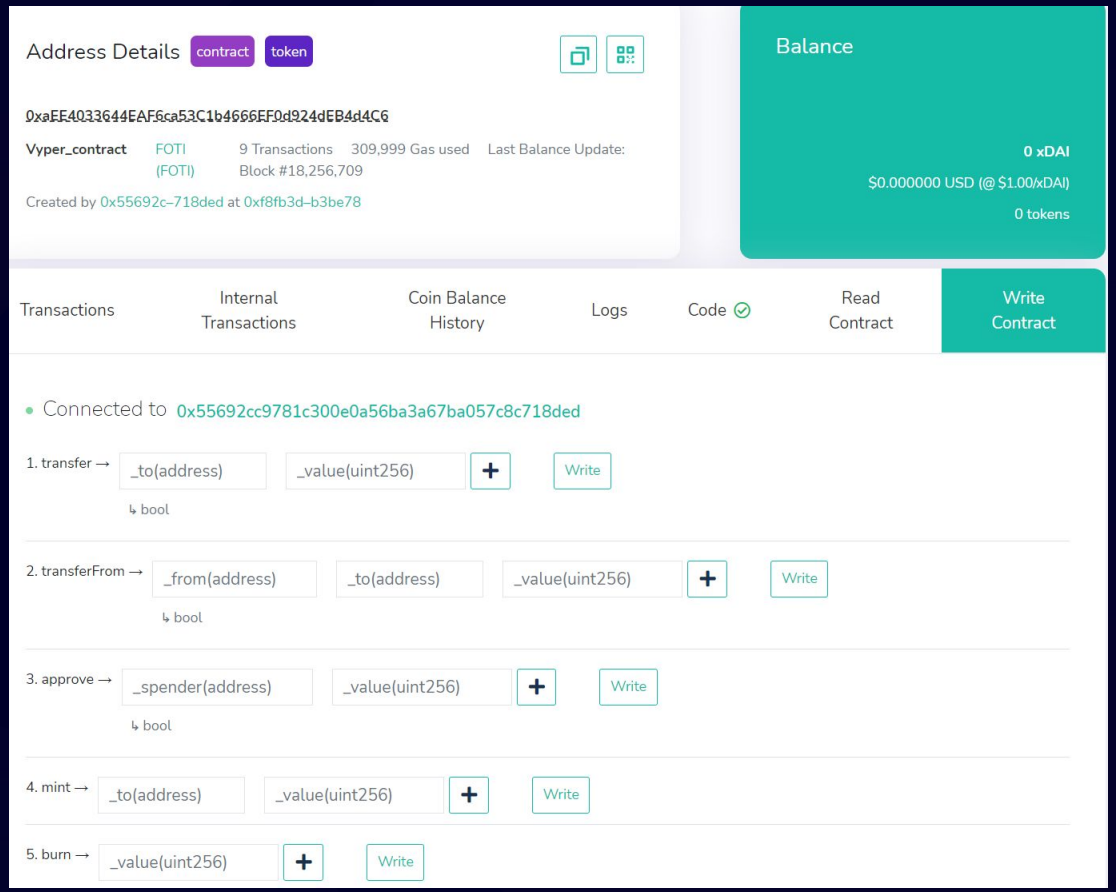
FOTI and Monke contract Addresses



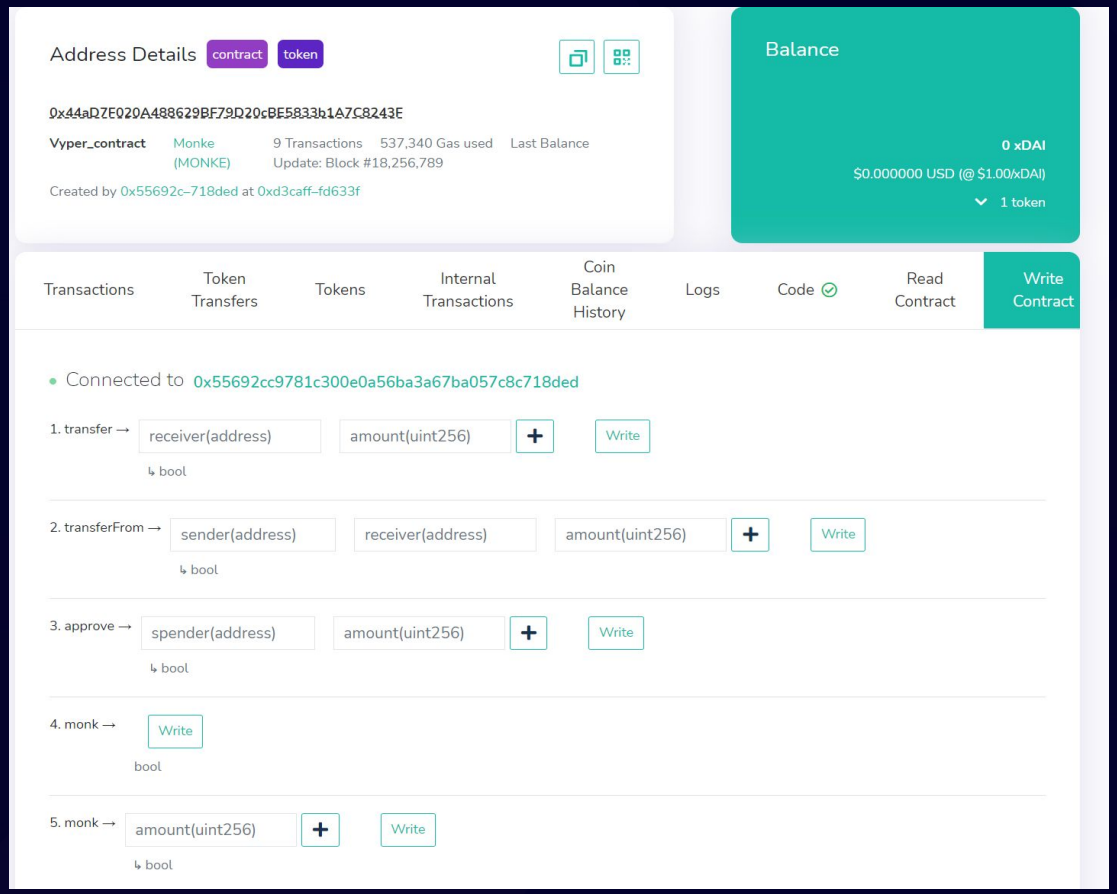


BlockScouts

- Once deployed, The contracts can be accessed from metamask through blockscouts



FOTI Contract

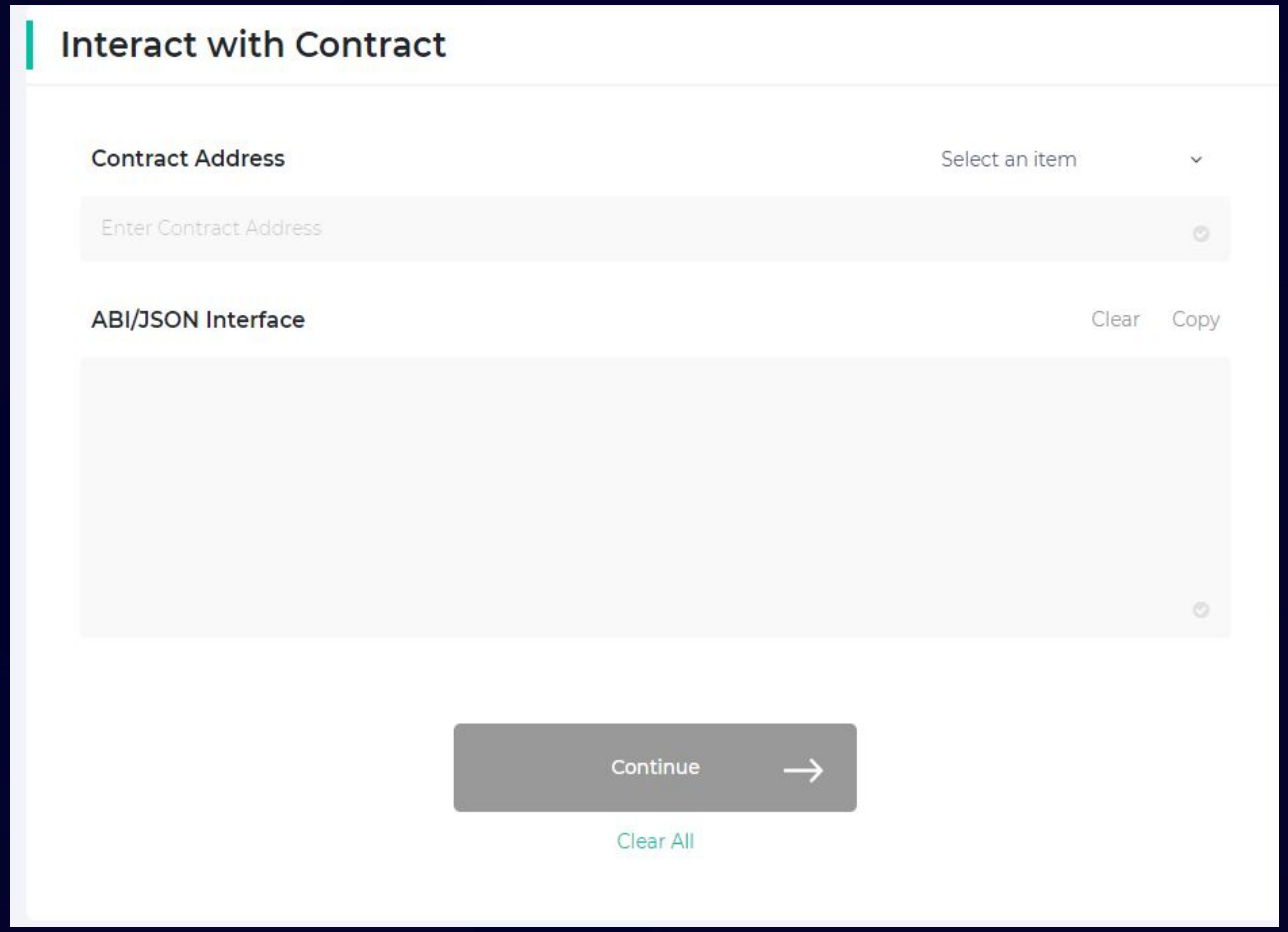


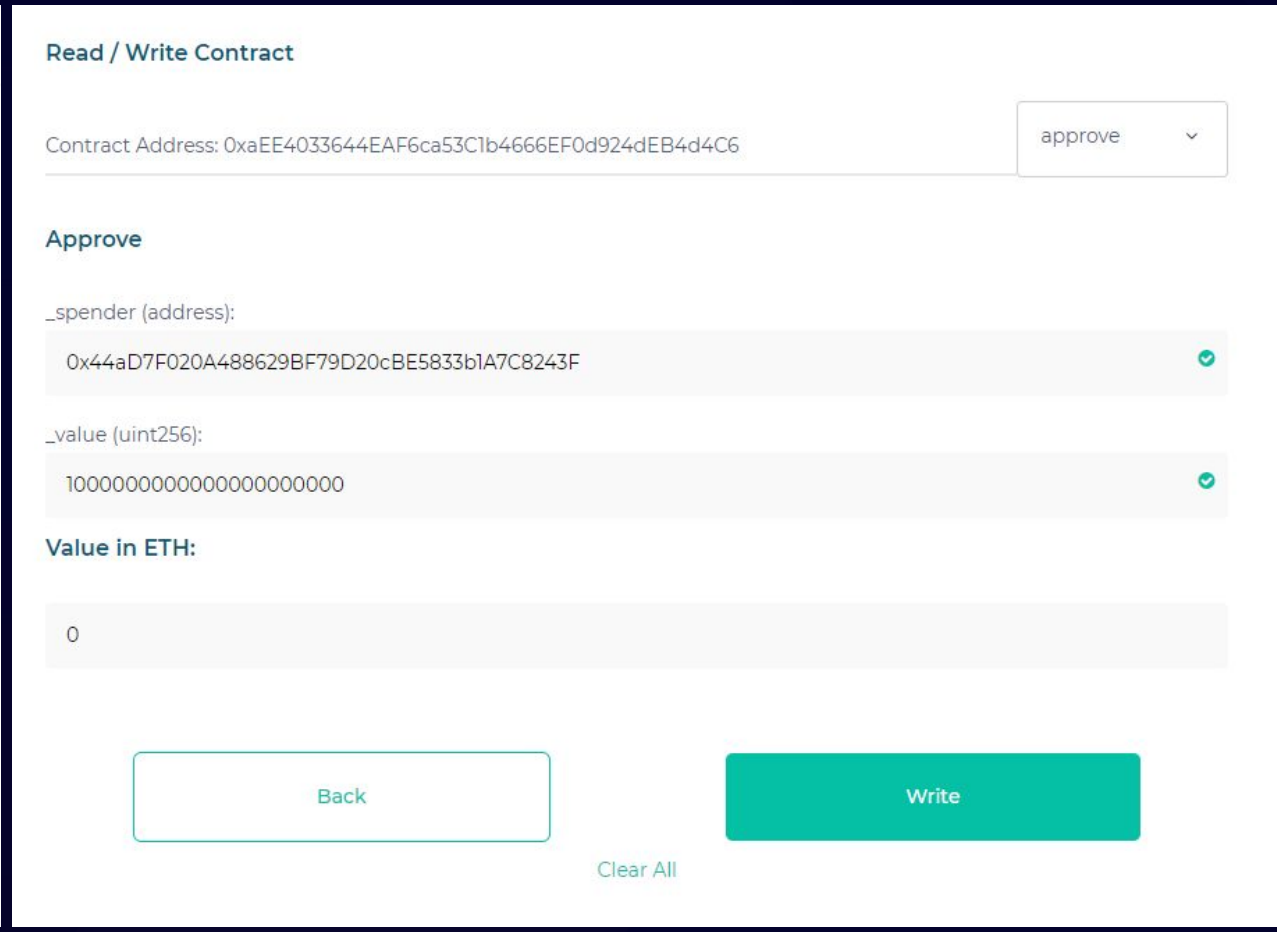
Monke Contract

Interacting with the contracts through MyEhterWallet (Approve)

You are first asked to input the Contract address and the ABI

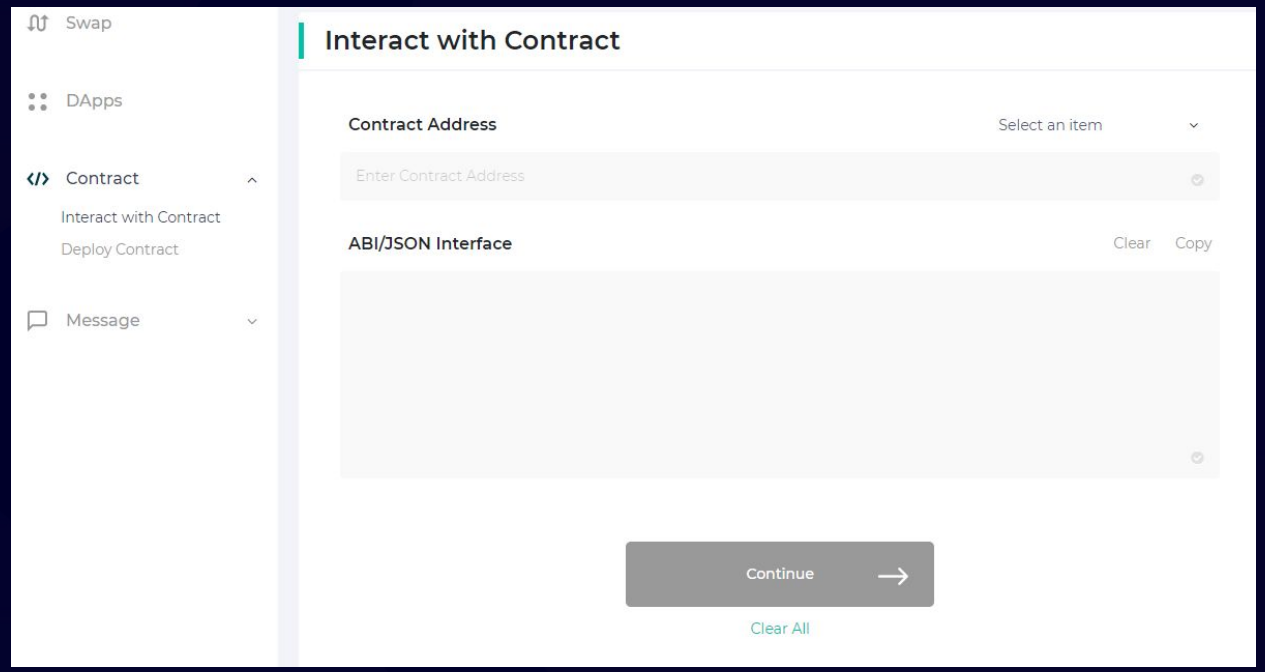
Using MyEtherWallet we can MONKE in FOTI Contract



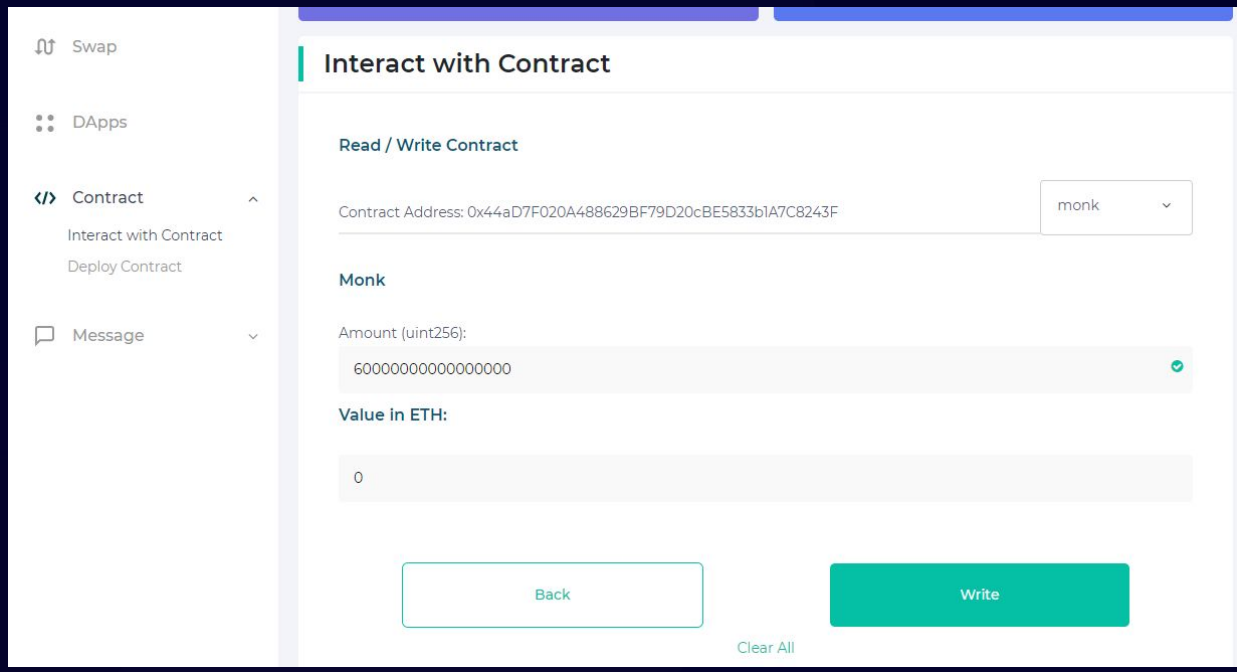


Interacting with the contracts through MyEhterWallet (MONK)

You are first asked to input the Contract address and the ABI

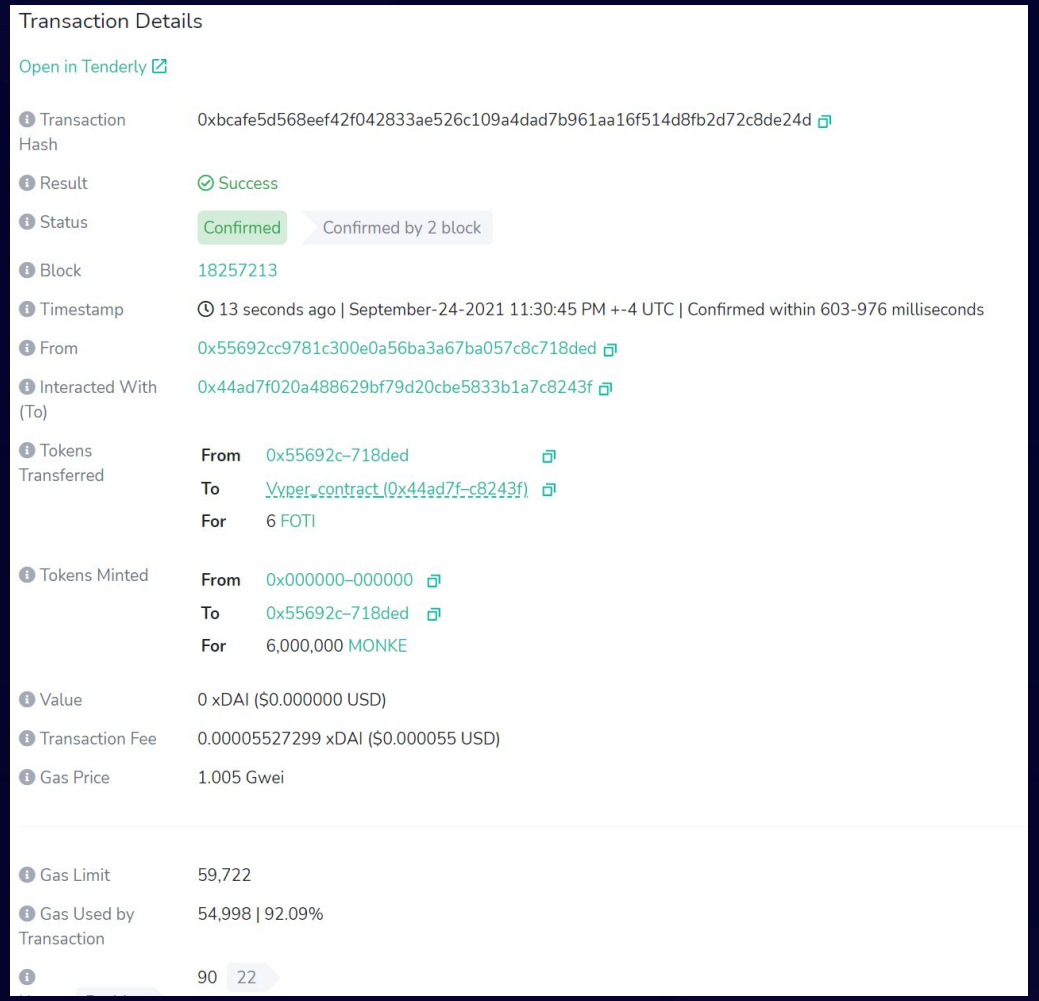


Using MyEtherWallet we can now monk our FOTI



MONK

Interacting with the contracts through MyEhterWallet



Transact 0.6FOTI for 600,000 MONKE

```
import React, { Component } from 'react'
import Monke from '../src/abis/monke.json'
import Navbar from './Navbar'
import Main from './Main'
import '../src/styles/Global.css'
import 'tailwindcss/tailwind.css'
import { Web3ReactProvider } from '@web3-react/core'
import Web3 from 'web3'
function getLibrary(provider) {
  return new Web3(provider)
}
function tailwind ({ Component, pageProps }) {
  return (
    <Web3ReactProvider getLibrary={getLibrary}>
      <Component {...pageProps} />
    </Web3ReactProvider>
  )
}
export default MyApp
class MyApp extends Component {
  async componentWillMount() {
    await this.loadWeb3()
    await this.loadBlockchainData()
  }
}
  async ; loadBlockchainData(); {
    const web3 = window.web3
    const accounts = await web3.eth.getAccounts()
    this.setState({ account: accounts[0] })
    const networkId = await web3.eth.net.getId()
    // Load MonkeToken
    const MonkeData = Monke.networks[networkId]
    if(MonkeData) {
      const mMnke = new web3.eth.Contract(Monke.abi, MonkeData.address)
      this.setState({ Monke })
      let MonkeBalance = await Monke.methods.balanceOf(this.state.account).call()
      this.setState({ MonkeBalance: MonkeBalance.toString() })
    } else {
      window.alert('Monke contract not deployed to detected network.')
    }
  }
```

Dapp.js

The Dapp.js is function here is to create a User interface where the Foti tokens can be exchanged for large amount of Monke

We do this by importing the abis from the foti, monke and Token farm contracts.

```
43    async ; loadWeb3() ; {
44      if (window.ethereum) {
45        window.web3 = new Web3(window.ethereum)
46        await window.ethereum.enable()
47      }
48      else if (window.web3) {
49        window.web3 = new Web3(window.web3.currentProvider)
50      }
51      else {
52        window.alert('Non-Ethereum browser detected. You should consider trying MetaMask!')
53      }
54    }
55    stakeTokens = (amount) => {
56      this.setState({ loading: true })
57      this.state.Foti.methods.approve(this.state.monke._address, amount).send({ from: this.state.account }).on('transactionHash', (hash) => {
58        this.state.monke.methods.stakeTokens(amount).send({ from: this.state.account }).on('transactionHash', (hash) => {
59          this.setState({ loading: false })
60        })
61      })
62    }
63    unstakeTokens = (amount) => {
64      this.setState({ loading: true })
65      this.state.monke.methods.unstakeTokens().send({ from: this.state.account }).on('transactionHash', (hash) => {
66        this.setState({ loading: false })
67      })
68    }
69    constructor(props) ; {
70      super(props)
71      this.state = {
72        account: '0x0',
73        Monke: {},
74        Foti: {},
75        FotiBalance: '0',
76        monkeBalance: '0',
77        stakingBalance: '0',
78        loading: true
79      }
80    }
```

Dapp.js

Here in the Dapp we are creating the circumstances in which Foti can be staked by Monke for exchange/conversion

Dapp.js continue....

```
81    render() ; {
82      let content
83      if(this.state.loading) {
84        content = <p id="loader" className="text-center">Loading...</p>
85      } else {
86        content = <Main
87          monkeBalance={this.state.monkeBalance}
88        />
89      }
90      return (
91        <div>
92          <Navbar account={this.state.account} />
93          <div className="container-fluid mt-5">
94            <div className="row">
95              <main role="main" className="col-lg-12 ml-auto mr-auto" style={{ maxWidth: '600px' }}>
96                <div className="content mr-auto ml-auto">
97                  <a
98                    href=""
99                    target="_blank"
100                   rel="noopener noreferrer"
101                 >
102                 </a>
103                 {content}
104               </div>
105             </main>
106         </div>
107       </div>
108       </div>
109     );
110   }
```

Dapp.js

We are importing the final touches on the interface here so that the website is compiled and looking like a normal website

Index.js

We are using Web3 injected connector here to connect your MetaMask wallet to the UI via react button

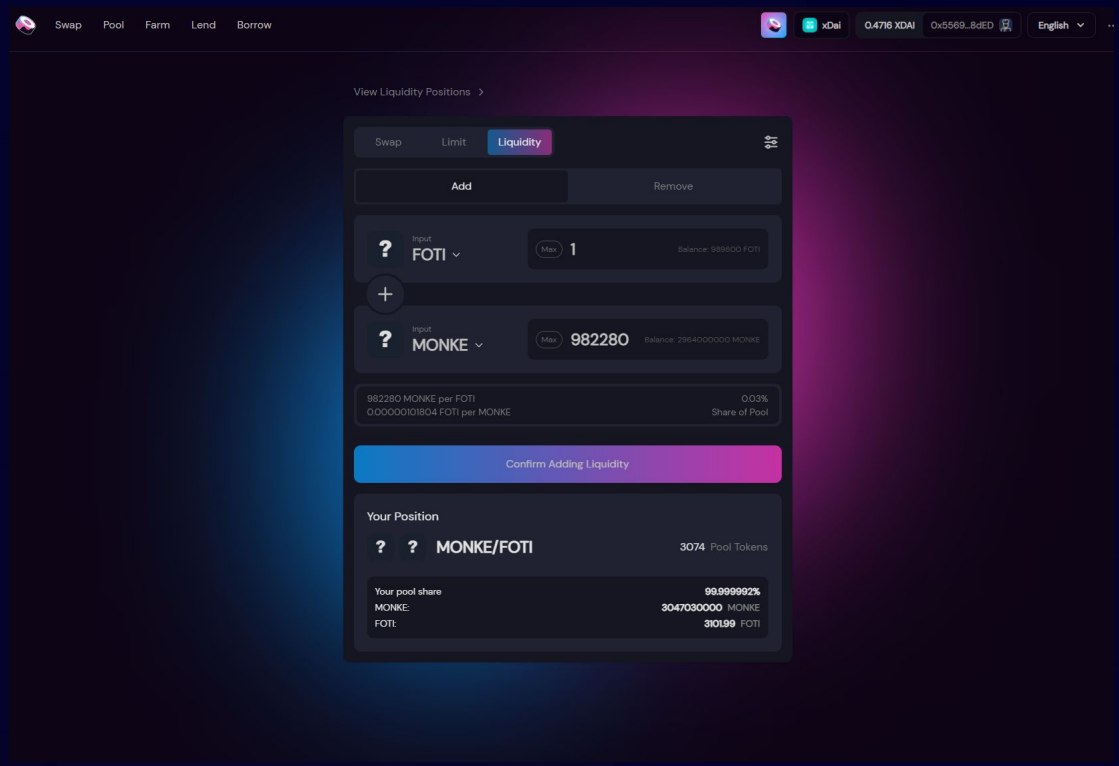
```
import React from 'react';
import ReactDOM from 'react-dom';
import { useWeb3React } from "@web3-react/core";
import Dapp from '../src/Dapp.js';
import * as serviceWorker from './serviceWorker';
import { injected } from "../src/connector.js";
export default function Home() {
    const { active, account, library, connector, activate, deactivate } = useWeb3React()
  async function connect() {
    try {
      await activate(injected)
    } catch (ex) {
      console.log(ex)
    }
}
  async function disconnect() {
    try {
      deactivate()
    } catch (ex) {
      console.log(ex)
    }
  }
  return (
    <div className="flex flex-col items
      <button onClick={connect} classNa
      {active ? <span>Connected with <b
      <button onClick={disconnect} clas
    </div>
  )
}
ReactDOM.render(<Dapp />, document.getElementById('root'));
// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

```
import { InjectedConnector } from '@web3-react/injected-connector'
export const injected = new InjectedConnector({
  supportedChainIds: [1, 3, 4, 5, 42, 100],
})
```

Creating the interface for the exchange

```
interface UniswapExchangeInterface():
    # Public Variables
    def tokenAddress() -> address: 0xaEE4033644EAF6ca53C1b4666EF0d924dEB4d4C6
    def factoryAddress() -> address: 0x44aD7F020A488629BF79D20cBE5833b1A7C8243F
    # Providing Liquidity
    def addLiquidity(min_liquidity: uint256, max_tokens: uint256, deadline: timestamp) -> uin
    def removeLiquidity(amount: uint256, min_eth: uint256(wei), min_tokens: uint256, deadline
    # Trading
    def ethToTokenSwapInput(min_tokens: uint256, deadline: timestamp) -> uint256: modifying
    def ethToTokenTransferInput(min_tokens: uint256, deadline: timestamp, recipient: address)
    def ethToTokenSwapOutput(tokens_bought: uint256, deadline: timestamp) -> uint256(wei): mo
    def ethToTokenTransferOutput(tokens_bought: uint256, deadline: timestamp, recipient: addr
    def tokenToEthSwapInput(tokens_sold: uint256, min_eth: uint256(wei), deadline: timestamp)
    def tokenToEthTransferInput(tokens_sold: uint256, min_eth: uint256(wei), deadline: timest
    def tokenToEthSwapOutput(eth_bought: uint256(wei), max_tokens: uint256, deadline: timesta
    def tokenToEthTransferOutput(eth_bought: uint256(wei), max_tokens: uint256, deadline: tim
    def tokenToTokenSwapInput(tokens_sold: uint256, min_tokens_bought: uint256, min_eth_bough
    def tokenToTokenTransferInput(tokens_sold: uint256, min_tokens_bought: uint256, min_eth_b
    def tokenToTokenSwapOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_eth_sold
    def tokenToTokenTransferOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_eth_
    def tokenToExchangeSwapInput(tokens_sold: uint256, min_tokens_bought: uint256, min_eth_bo
    def tokenToExchangeTransferInput(tokens_sold: uint256, min_tokens_bought: uint256, min_et
    def tokenToExchangeSwapOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_eth_s
    def tokenToExchangeTransferOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_e
    # Get Price
    def getEthToTokenInputPrice(eth_sold: uint256(wei)) -> uint256: constant
    def getEthToTokenOutputPrice(tokens_bought: uint256) -> uint256(wei): constant
    def getTokenToEthInputPrice(tokens_sold: uint256) -> uint256(wei): constant
    def getTokenToEthOutputPrice(eth_bought: uint256(wei)) -> uint256: constant
    # Pool Token ERC20 Compatibility
    def balanceOf() -> address: constant
    def allowance(_owner : address, _spender : address) -> uint256: constant
    def transfer(_to : address, _value : uint256) -> bool: modifying
    def transferFrom(_from : address, _to : address, _value : uint256) -> bool: modifying
    def approve(_spender : address, _value : uint256) -> bool: modifying
    # Setup
    def setup(token_addr: address): modifying
```

We need to import the abi from this token exchange contract in order to have the swap function in our interface



https://app.sushi.com/swap

Thank You

And See You Next Time