Decentralized Cryptocurrency Exchange

Final Project

Contributors

Victor Freire

Vijaya Reddy

Yan Domingos

Fernanda O’Malley

Diana Cubides

FinTech BootCamp - University of Toronto

Decentralized Cryptocurrency Exchange

Topics to be covered

- Project Summary/Objectives
- Dependencies
- Token Smart Contract
- CryptoCurrency Exchange Contract
- Front-end development
- Next steps

OBJECTIVE:

Develop a Blockchain App that allows to code and deploy smart contracts to run a decentralized currency exchange that allows to trade cryptocurrency.

How to use the exchange:

Step 1: deposit funds

Step 2: make order

Step 3: fill order

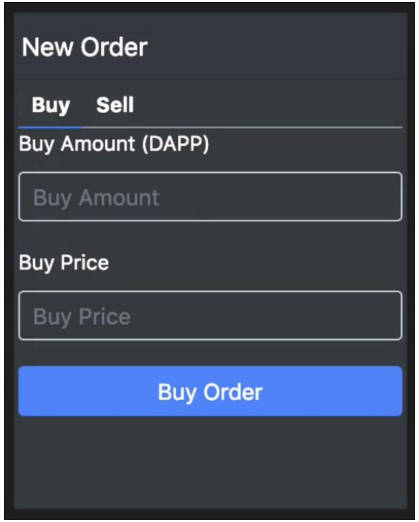
Step 4: withdraw funds

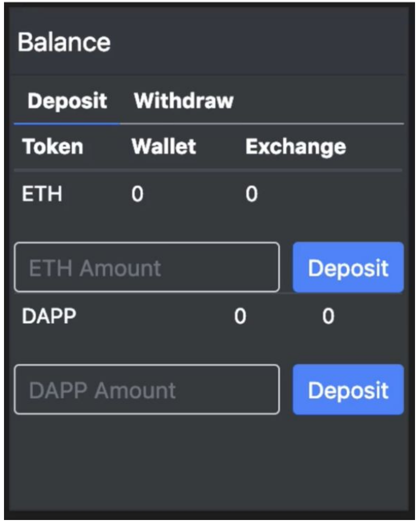
Market: ETH/To Token

All orders will be limit orders - one creates order and someone else needs to match that order.

Sell orders or buy orders

Fill orders: maker, taker, fees (which go to account which deployed the contract)





Token	Wallet	Exchange
ETH	0	0
DAPP	0	0

SMART CONTRACTS

- **Token Smart Contract:** token built, following ERC standards
- **CryptoCurrency Exchange Contract:** Built to have the capability of deposit and withdraw funds (tokens or Ether), manage orders and charge fees.

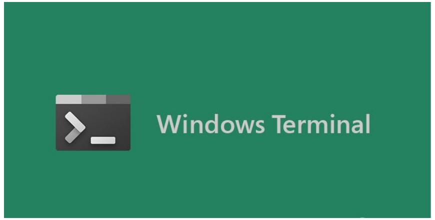
Solidity used as programming language, SafeMath imported

Truffle allows to test the contract as it is being built, to identify errors and make sure once it is deployed to the blockchain is fully functional

Parallel coding in Truffle for testing purposes

Dependencies:

JavaScript



Windows Terminal

Solidity

TRUFFLE

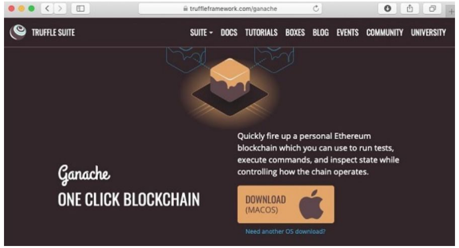
ubuntu®

Downloads

Python

MetaMask

React



Ganache

ONE CLICK BLOCKCHAIN

Quickly fire up a personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates.

BABEL

Dependency : Back-End

Chrome : is used to add Metamask extension which interacts with the blockchain.

Metamask : It is an Ethereum wallet, which will allow us to hold crypto currency and also interact with De-apps and build them. Will change our browser into blockchain browser.

Sublime Text : Is a text editor used to write the code. From the Package Control, Installed the Ethereum Package, which gives the syntax highlighting for the Solidity program, to build the Smart Contracts.

Environment : Back-End

Ganache : Used as a Local blockchain
React : is used for building UI and handling the view layer, usually for single-page app
Node version manager : nvm, helps to install the required version.(here we have used 10.19.0)
Truffle : is a framework that helps in developing Ethereum smart contracts, test and publish on blockchain.
Ubuntu : Using Ubuntu we can Open project in Sublime text and also after running application will update any changes made to the code.

Dependency back-end / front-end

appexcharts : candlestick charts
babel : ES6 features inside of truffle project, javascript compiler
bootstrap : theming the project
chai : testing smart contracts
dotenv : reading environment variables
moment : enables us to work with time in javascript
openzeppelin-solidity : solidity smart contracts and enables to use math
react : to create the react app, building the UI
redux : app state management of the front-end (react)
solidity : build smart contracts that run on Ethereum
web3 : makes client side application talk to the blockchain

Dependency : Front-End

Redux DevTools : Frontend, will stores data in our React app on the frontend. It will allow us to inspect what is happening inside of Redux , as a data store.

Infura : Connects to Ethereum blockchain without having to run our node. The end-point url can be used to connect to the Ethereum when the project is deployed to the public Ethereum network.

Heroku : Is the hosting provider to deploy the project. Deploy the App to production.

Building Smart Contracts

For this project, two contracts need to be designed and deployed:

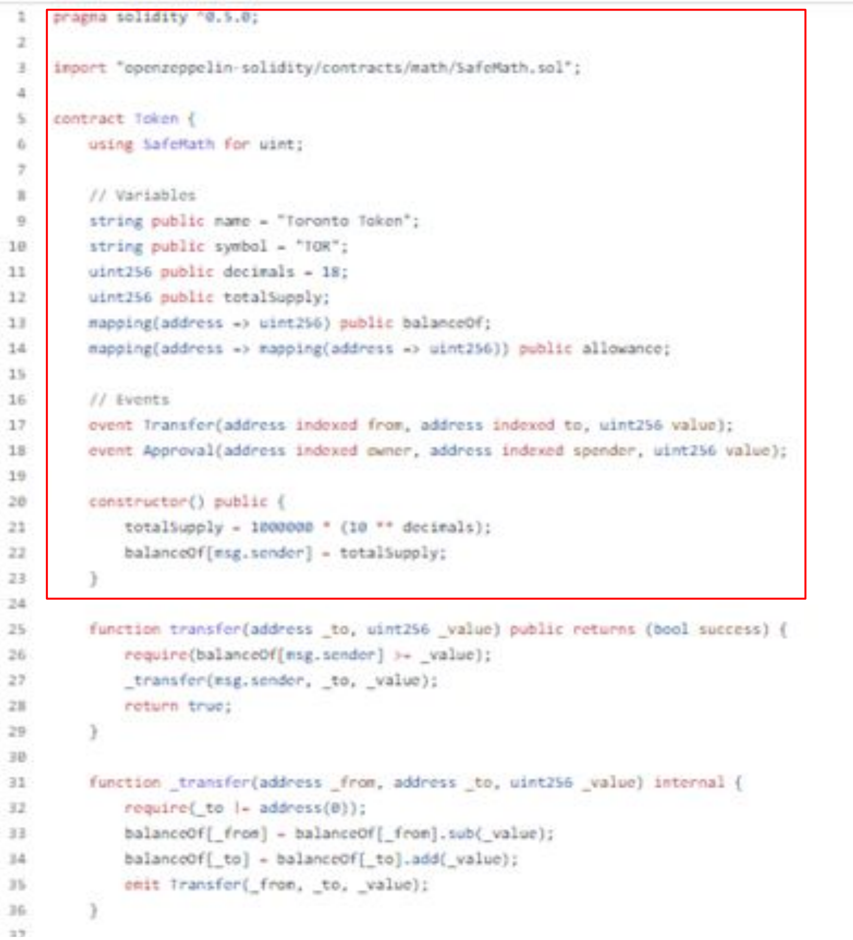
1. **Token Smart Contract**: Following ERC20 Standards

2. **CryptoCurrency Exchange Contract:** Built to have the capability of deposit and withdraw funds (tokens or Ether), manage orders and charge fees.

Token Smart Contract: “Toronto Token”

- ERC20 standards followed:

✓ name, symbol, decimals, supply
✓ Feature to track balances. It adds credibility to the smart contract
✓ Total supply assigned to deployer of contract
✓ Event (who is transfering, who is being transferred to and how much)



```
pragma solidity ^0.5.0;

import "openzeppelin-solidity/contracts/math/SafeMath.sol";

contract Token {
    using SafeMath for uint;

    // Variables
    string public name = "Toronto Token";
    string public symbol = "TOR";
    uint256 public decimals = 18;
    uint256 public totalSupply;
    mapping(address => uint256) public balanceOf;
    mapping(address => mapping(address => uint256)) public allowance;

    // Events
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    constructor() public {
        totalSupply = 1000000 * (10 ** decimals);
        balanceOf[msg.sender] = totalSupply;
    }

    function transfer(address _to, uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);
        _transfer(msg.sender, _to, _value);
        return true;
    }

    function _transfer(address _from, address _to, uint256 _value) internal {
        require(_to != address(0));
        balanceOf[_from] = balanceOf[_from].sub(_value);
        balanceOf[_to] = balanceOf[_to].add(_value);
        emit Transfer(_from, _to, _value);
    }
```

Token Smart Contract: “Toronto Token”

- ERC20 standards followed (continued):

✓ Approve function, which permits tokens to be added to an allowance to be later exchanged. These tokens are approved for a *spender (3rd party arbitrary address)* and are approved by msg.sender (deployer of contract)

✓ Allowance mapped on earlier stage of code

✓ Transfer function:

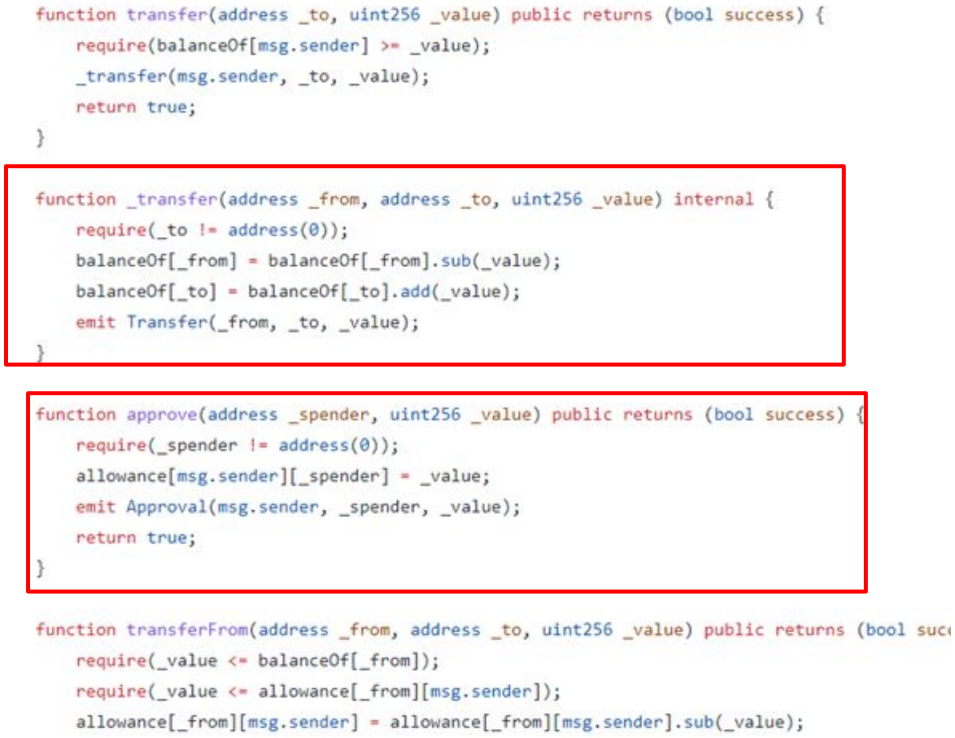
➢ Two functions coded (one public, one internal)

➢ Internal function only visible inside the smart contract

➢ Internal transfer function enables the transfer from and to arbitrary accounts

➢ Internal function transfer adjusts new value after transfer for to and from addresses

➢ Prevents transactions with invalid addresses



```
function transfer(address _to, uint256 _value) public returns (bool success) {
    require(balanceOf[msg.sender] >= _value);
    _transfer(msg.sender, _to, _value);
    return true;
}

function _transfer(address _from, address _to, uint256 _value) internal {
    require(_to != address(0));
    balanceOf[_from] = balanceOf[_from].sub(_value);
    balanceOf[_to] = balanceOf[_to].add(_value);
    emit Transfer(_from, _to, _value);
}

function approve(address _spender, uint256 _value) public returns (bool success) {
    require(_spender != address(0));
    allowance[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}

function transferFrom(address _from, address _to, uint256 _value) public returns (bool suc
    require(_value <= balanceOf[_from]);
    require(_value <= allowance[_from][msg.sender]);
    allowance[_from][msg.sender] = allowance[_from][msg.sender].sub(_value);
```

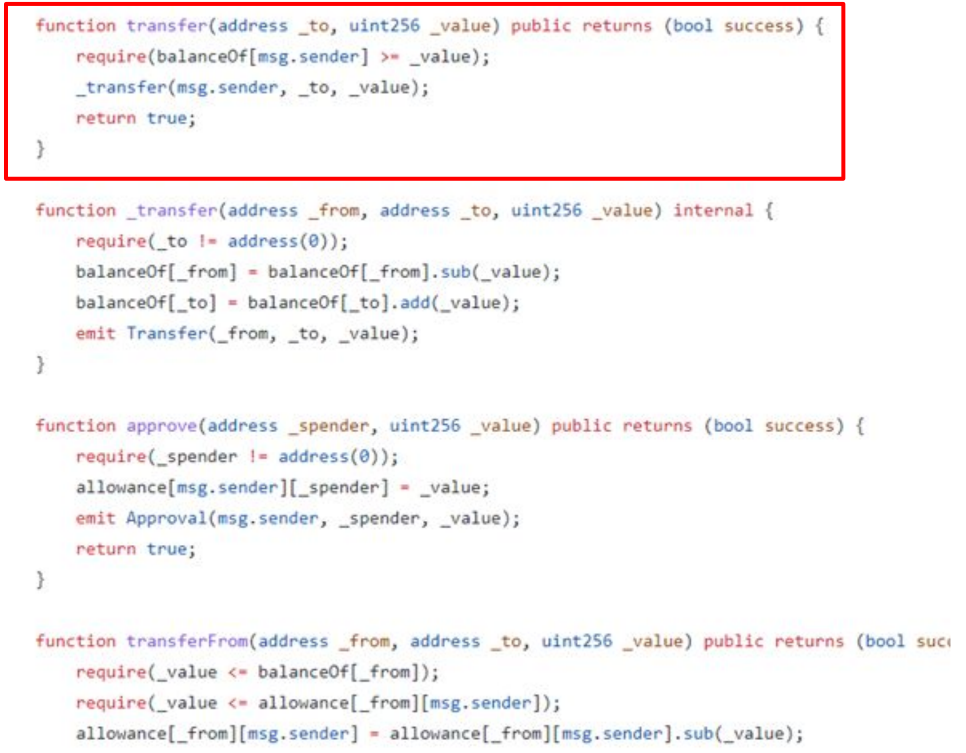
Token Smart Contract: “Toronto Token”

- ERC20 standards followed (continued):

  ✓ Transfer function:

  ➢ Two functions coded (one public, one internal)

  ➢ Public function passes values from msg.sender to arbitrary _to address relying on internal transfer function



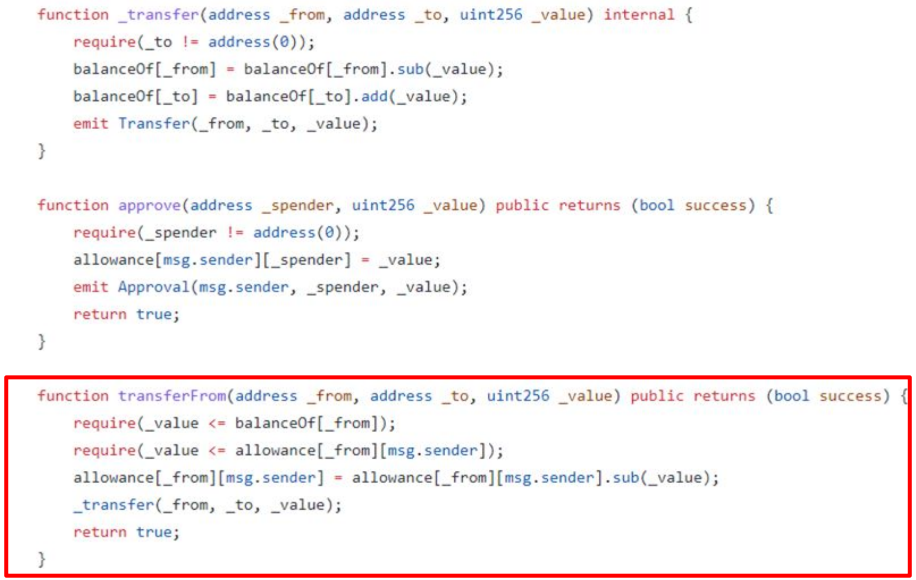
```
function transfer(address _to, uint256 _value) public returns (bool success) {
    require(balanceOf[msg.sender] >= _value);
    _transfer(msg.sender, _to, _value);
    return true;
}

function _transfer(address _from, address _to, uint256 _value) internal {
    require(_to != address(0));
    balanceOf[_from] = balanceOf[_from].sub(_value);
    balanceOf[_to] = balanceOf[_to].add(_value);
    emit Transfer(_from, _to, _value);
}

function approve(address _spender, uint256 _value) public returns (bool success) {
    require(_spender != address(0));
    allowance[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}

function transferFrom(address _from, address _to, uint256 _value) public returns (bool suc
    require(_value <= balanceOf[_from]);
    require(_value <= allowance[_from][msg.sender]);
    allowance[_from][msg.sender] = allowance[_from][msg.sender].sub(_value);
```

Token Smart Contract: “Toronto Token”

- ERC20 standards followed (continued):

✓ Transfer _from function, which allows to complete transfer tokens in the exchange from arbitrary accounts, creating the basis of the decentralized exchange.

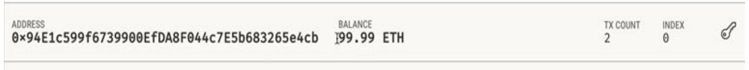
- ➢ Require statement to validate _from address has enough balance to complete transaction
- ➢ Require statement to certifies allowance has been approved from msg.sender
- ➢ Modifies new balance for msg.sender as tokens are consumed
- ➢ Executes the base for Decentralized Exchange by finalizing transfer _from _to,



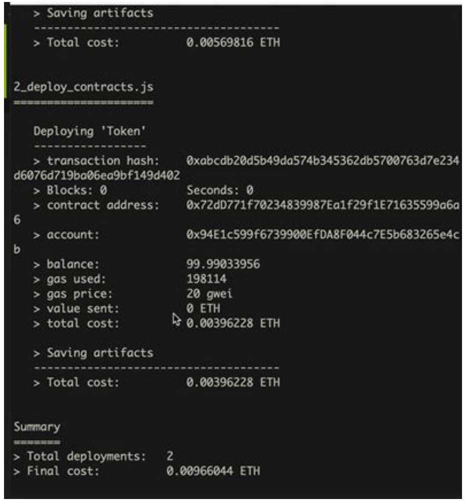
```
function _transfer(address _from, address _to, uint256 _value) internal {
    require(_to != address(0));
    balanceOf[_from] = balanceOf[_from].sub(_value);
    balanceOf[_to] = balanceOf[_to].add(_value);
    emit Transfer(_from, _to, _value);
}

function approve(address _spender, uint256 _value) public returns (bool success) {
    require(_spender != address(0));
    allowance[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}

function transferFrom(address _from, address _to, uint256 _value) public returns (bool success) {
    require(_value <= balanceOf[_from]);
    require(_value <= allowance[_from][msg.sender]);
    allowance[_from][msg.sender] = allowance[_from][msg.sender].sub(_value);
    _transfer(_from, _to, _value);
    return true;
}
```

Token Smart Contract: “Toronto Token”

- Deploying token and contract into the Blockchain uses gas

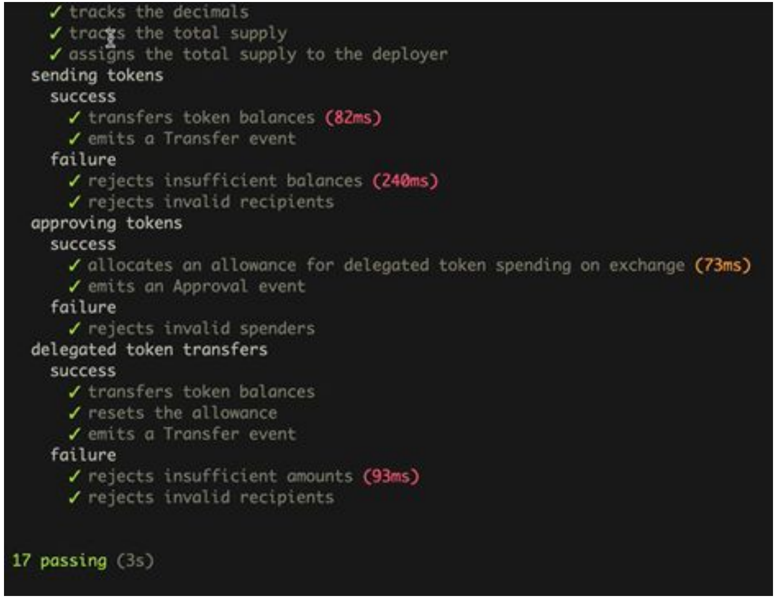


```
ADDRESS                                      BALANCE                                TX COUNT   INDEX
0×94E1c599f6739900EfDA8F044c7E5b683265e4cb   99.99 ETH                              2          0
```



```
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00569816 ETH


2_deploy_contracts.js
=====================

   Deploying 'Token'
   -----------------
   > transaction hash:    0xabcdb20d5b49da574b345362db5700763d7e234
d6076d719ba06ea9bf149d402
   > Blocks: 0            Seconds: 0
   > contract address:    0x72dD771f70234839987Ea1f29f1E71635599a6a
6
   > account:             0x94E1c599f6739900EfDA8F044c7E5b683265e4c
b
   > balance:             99.99033956
   > gas used:            198114
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00396228 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00396228 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.00966044 ETH
```

- All steps of the Toronto Token Smart Contract have been tested in Truffle
- After a succesful set of tests, a cryptocurrency that can be run on Ethereum has been created!



```
    ✓ tracks the decimals
    ✓ tracks the total supply
    ✓ assigns the total supply to the deployer
  sending tokens
    success
      ✓ transfers token balances (82ms)
      ✓ emits a Transfer event
    failure
      ✓ rejects insufficient balances (240ms)
      ✓ rejects invalid recipients
  approving tokens
    success
      ✓ allocates an allowance for delegated token spending on exchange (73ms)
      ✓ emits an Approval event
    failure
      ✓ rejects invalid spenders
  delegated token transfers
    success
      ✓ transfers token balances
      ✓ resets the allowance
      ✓ emits a Transfer event
    failure
      ✓ rejects insufficient amounts (93ms)
      ✓ rejects invalid recipients


17 passing (3s)
```

Building Smart Contracts

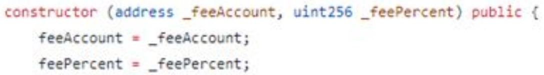
For this project, two contracts need to be designed and deployed:

1. **Token Smart Contract**: Following ERC20 Standards

2. **CryptoCurrency Exchange Contract:** Built to have the capability of deposit and withdraw funds (tokens or Ether), manage orders and charge fees.

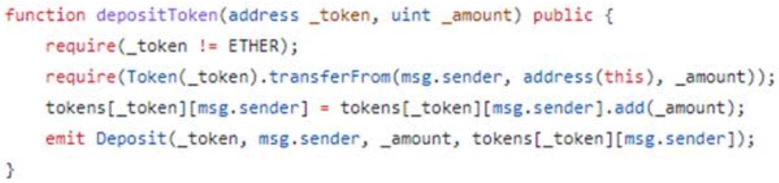
Building the Exchange: Deposits

- Set fee account and % through constructor



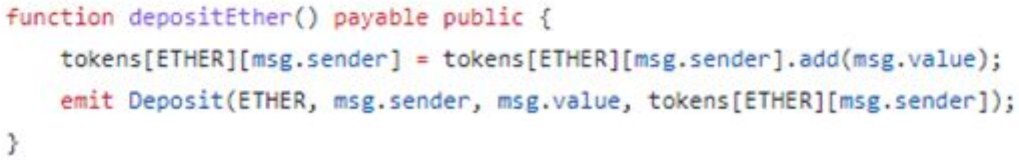
```
constructor (address _feeAccount, uint256 _feePercent) public {
    feeAccount = _feeAccount;
    feePercent = _feePercent;
```

- Allow Token deposits, by importing Toronto Token contract and creating deposit function
  - ✓ Designed exclusively for Tokens, won't accept ETHER (separate function)



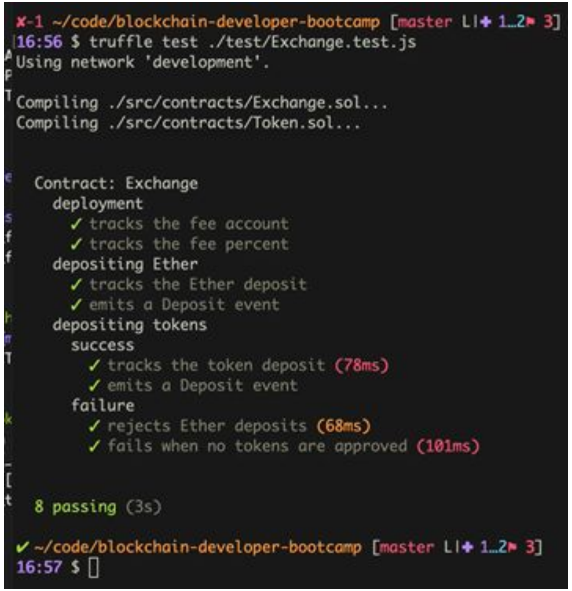
```
function depositToken(address _token, uint _amount) public {
    require(_token != ETHER);
    require(Token(_token).transferFrom(msg.sender, address(this), _amount));
    tokens[_token][msg.sender] = tokens[_token][msg.sender].add(_amount);
    emit Deposit(_token, msg.sender, _amount, tokens[_token][msg.sender]);
}
```

- Allow ETHER deposits
  - ✓ Needs payable modifier
  - ✓ Needs msg.value to identify amount to be transfered



```
function depositEther() payable public {
    tokens[ETHER][msg.sender] = tokens[ETHER][msg.sender].add(msg.value);
    emit Deposit(ETHER, msg.sender, msg.value, tokens[ETHER][msg.sender]);
}
```

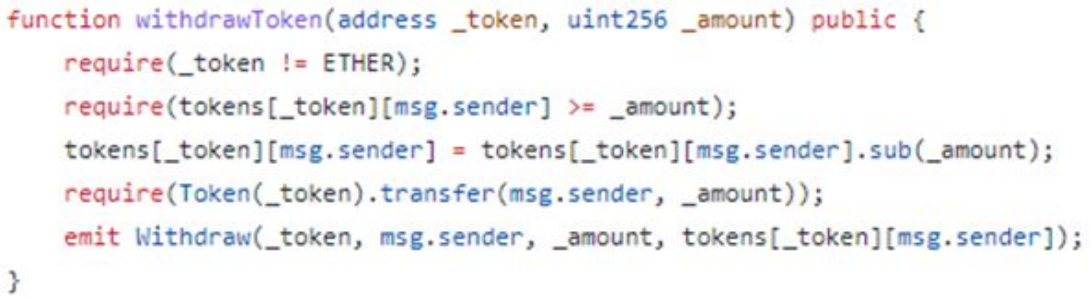
Tested on Truffle… it works!



```
✘-1 ~/code/blockchain-developer-bootcamp [master L|✚ 1…2⚑ 3]
16:56 $ truffle test ./test/Exchange.test.js
Using network 'development'.

Compiling ./src/contracts/Exchange.sol...
Compiling ./src/contracts/Token.sol...


  Contract: Exchange
    deployment
      ✓ tracks the fee account
      ✓ tracks the fee percent
    depositing Ether
      ✓ tracks the Ether deposit
      ✓ emits a Deposit event
    depositing tokens
      success
        ✓ tracks the token deposit (78ms)
        ✓ emits a Deposit event
      failure
        ✓ rejects Ether deposits (68ms)
        ✓ fails when no tokens are approved (101ms)


  8 passing (3s)

✔ ~/code/blockchain-developer-bootcamp [master L|✚ 1…2⚑ 3]
16:57 $ ▯
```

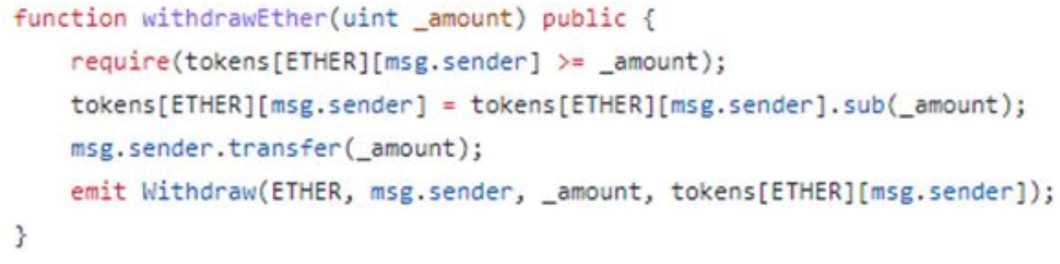
Building the Exchange: Withdraw

- Allow Token withdrawals



```
function withdrawToken(address _token, uint256 _amount) public {
    require(_token != ETHER);
    require(tokens[_token][msg.sender] >= _amount);
    tokens[_token][msg.sender] = tokens[_token][msg.sender].sub(_amount);
    require(Token(_token).transfer(msg.sender, _amount));
    emit Withdraw(_token, msg.sender, _amount, tokens[_token][msg.sender]);
}
```

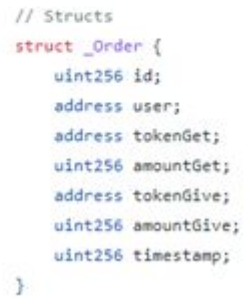
- Allow ETHER withdraws



```
function withdrawEther(uint _amount) public {
    require(tokens[ETHER][msg.sender] >= _amount);
    tokens[ETHER][msg.sender] = tokens[ETHER][msg.sender].sub(_amount);
    msg.sender.transfer(_amount);
    emit Withdraw(ETHER, msg.sender, _amount, tokens[ETHER][msg.sender]);
}
```

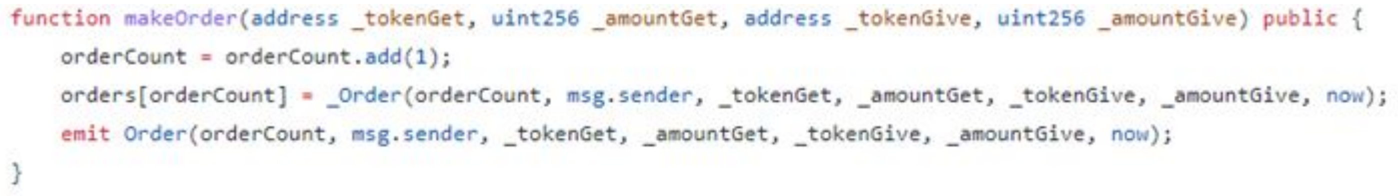
Building the Exchange: Managing Orders

- After depositing tokens or ETHER, users can make different selections in the exchange
- To model the order, contract uses *struct* where different attributes are given to the order



```
// Structs
struct _Order {
    uint256 id;
    address user;
    address tokenGet;
    uint256 amountGet;
    address tokenGive;
    uint256 amountGive;
    uint256 timestamp;
}
```

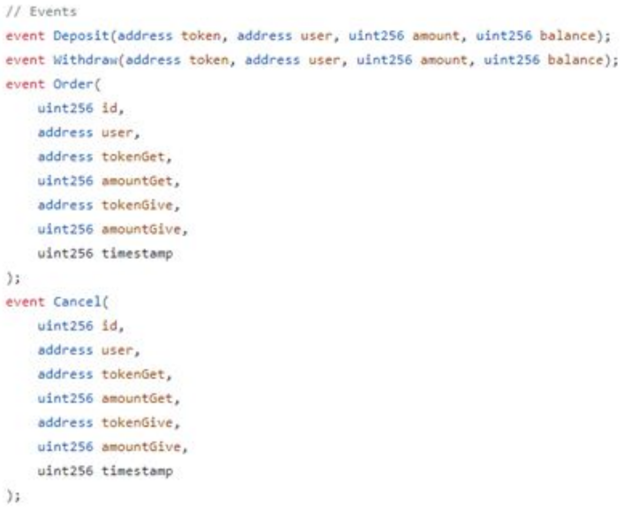
- Order is stored through mapping and is created in the blockchain through function makeOrder



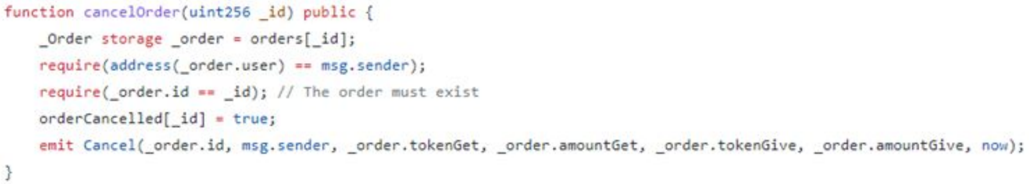
```
function makeOrder(address _tokenGet, uint256 _amountGet, address _tokenGive, uint256 _amountGive) public {
    orderCount = orderCount.add(1);
    orders[orderCount] = _Order(orderCount, msg.sender, _tokenGet, _amountGet, _tokenGive, _amountGive, now);
    emit Order(orderCount, msg.sender, _tokenGet, _amountGet, _tokenGive, _amountGive, now);
}
```

Building the Exchange: Managing Orders

- Cancel Order: As the orders go into the blockchain, they can not be deleted. For this reason it is important to keep a specific record of all canceled orders (event).



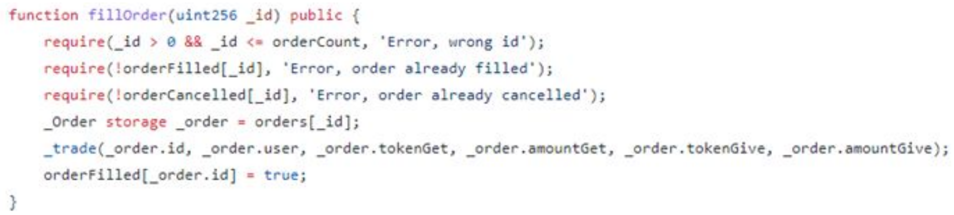
```
// Events
event Deposit(address token, address user, uint256 amount, uint256 balance);
event Withdraw(address token, address user, uint256 amount, uint256 balance);
event Order(
    uint256 id,
    address user,
    address tokenGet,
    uint256 amountGet,
    address tokenGive,
    uint256 amountGive,
    uint256 timestamp
);
event Cancel(
    uint256 id,
    address user,
    address tokenGet,
    uint256 amountGet,
    address tokenGive,
    uint256 amountGive,
    uint256 timestamp
);
```



```
function cancelOrder(uint256 _id) public {
    _Order storage _order = orders[_id];
    require(address(_order.user) == msg.sender);
    require(_order.id == _id); // The order must exist
    orderCancelled[_id] = true;
    emit Cancel(_order.id, msg.sender, _order.tokenGet, _order.amountGet, _order.tokenGive, _order.amountGive, now);
}
```

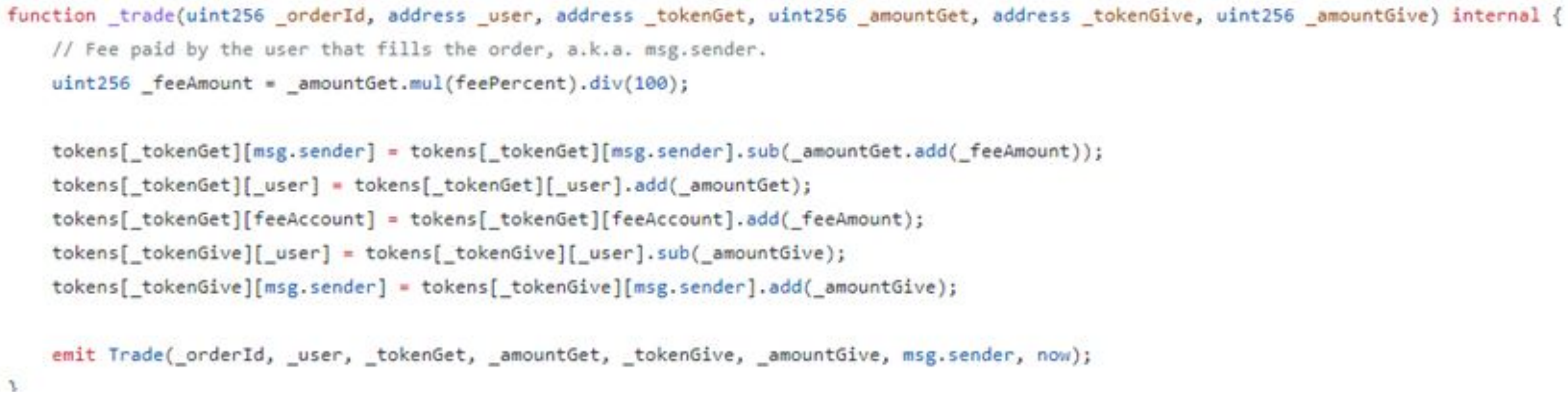
Building the Exchange: Trades

- Function fillOrder to fetch and execute the order, will mark the order as filled .



```
function fillOrder(uint256 _id) public {
    require(_id > 0 && _id <= orderCount, 'Error, wrong id');
    require(!orderFilled[_id], 'Error, order already filled');
    require(!orderCancelled[_id], 'Error, order already cancelled');
    _Order storage _order = orders[_id];
    _trade(_order.id, _order.user, _order.tokenGet, _order.amountGet, _order.tokenGive, _order.amountGive);
    orderFilled[_order.id] = true;
}
```

- Function “_trade” will execute the trade, charge the fees and emit the trade event



```
function _trade(uint256 _orderId, address _user, address _tokenGet, uint256 _amountGet, address _tokenGive, uint256 _amountGive) internal {
    // Fee paid by the user that fills the order, a.k.a. msg.sender.
    uint256 _feeAmount = _amountGet.mul(feePercent).div(100);

    tokens[_tokenGet][msg.sender] = tokens[_tokenGet][msg.sender].sub(_amountGet.add(_feeAmount));
    tokens[_tokenGet][_user] = tokens[_tokenGet][_user].add(_amountGet);
    tokens[_tokenGet][feeAccount] = tokens[_tokenGet][feeAccount].add(_feeAmount);
    tokens[_tokenGive][_user] = tokens[_tokenGive][_user].sub(_amountGive);
    tokens[_tokenGive][msg.sender] = tokens[_tokenGive][msg.sender].add(_amountGive);

    emit Trade(_orderId, _user, _tokenGet, _amountGet, _tokenGive, _amountGive, msg.sender, now);
}
```

Front-end development and next steps

For this project our main focus was to build a good and solid back-end, with all the codes and functionalities working well.

Due to the lack of time and complexity, we were unable to finish the front end . We will continue to work on this project to have all the parts done.

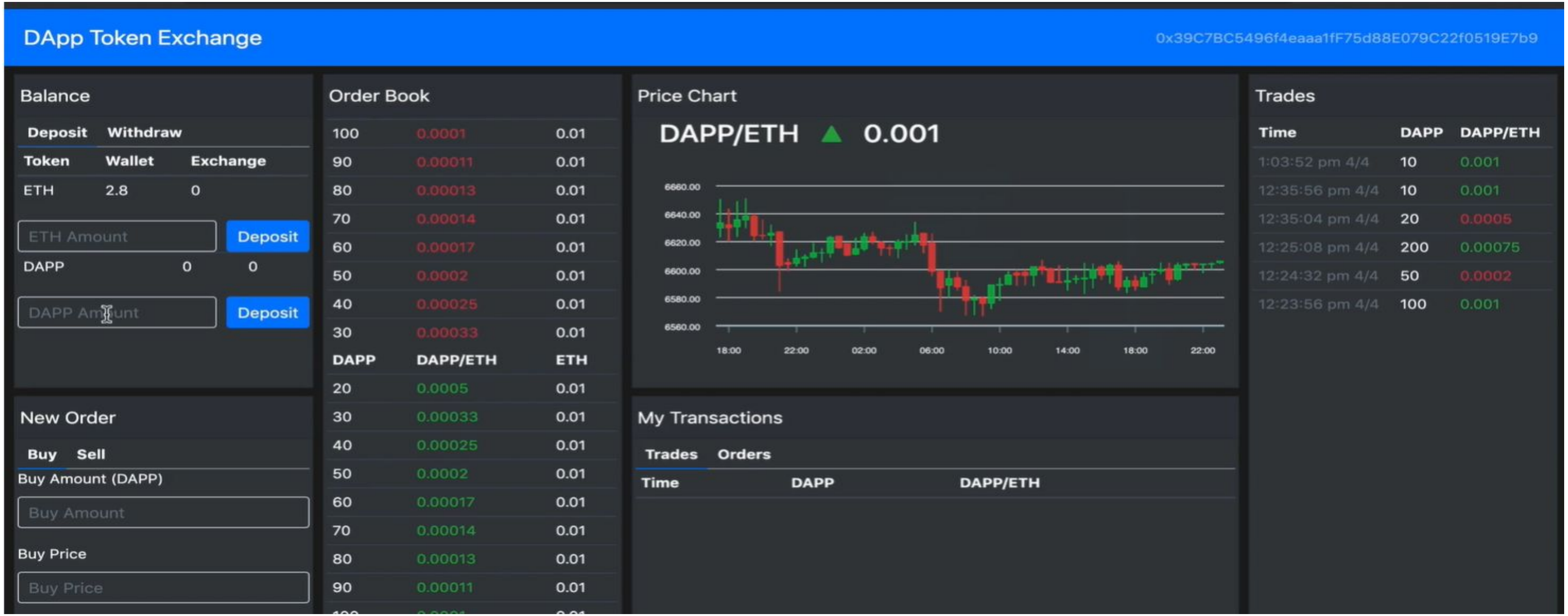
Those are some of the libraries we will use to build the interface of our Decentralized Exchange App :

- App
- Bootstrap
- Service worker
- HTML
- CSS
- Flexbox
- Json
- Reselect

Decentralized Exchange

How the finished product will look

- This is our goal on how our front end will finally look like once its done.



DApp Token Exchange

0x39C7BC5496f4eaaa1fF75d88E079C22f0519E7b9

Balance

Deposit Withdraw

Token	Wallet	Exchange
ETH	2.8	0

ETH Amount — Deposit

DAPP	0	0

DAPP Amount — Deposit

New Order

Buy Sell

Buy Amount (DAPP)

Buy Amount

Buy Price

Buy Price

Order Book

100	0.0001	0.01
90	0.00011	0.01
80	0.00013	0.01
70	0.00014	0.01
60	0.00017	0.01
50	0.0002	0.01
40	0.00025	0.01
30	0.00033	0.01
DAPP	DAPP/ETH	ETH
20	0.0005	0.01
30	0.00033	0.01
40	0.00025	0.01
50	0.0002	0.01
60	0.00017	0.01
70	0.00014	0.01
80	0.00013	0.01
90	0.00011	0.01

Price Chart

DAPP/ETH ▲ 0.001

My Transactions

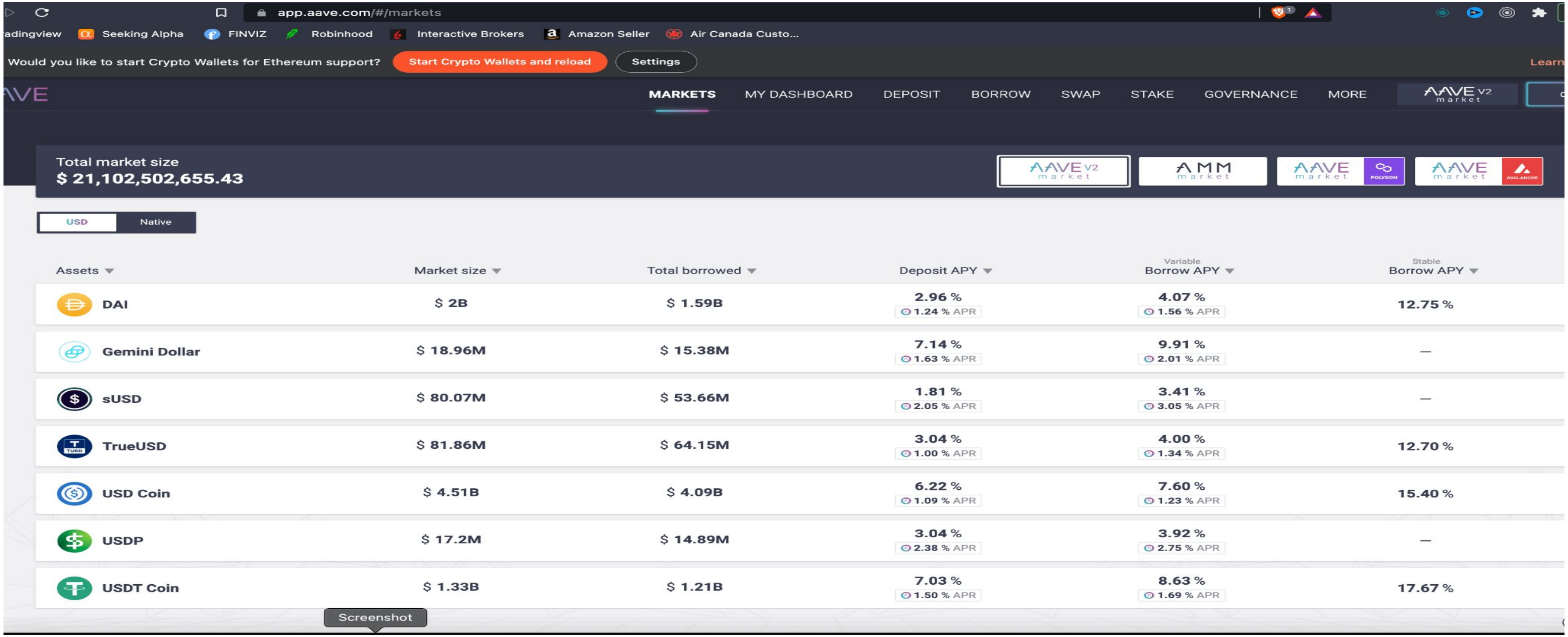
Trades Orders

Time	DAPP	DAPP/ETH

Trades

Time	DAPP	DAPP/ETH
1:03:52 pm 4/4	10	0.001
12:35:56 pm 4/4	10	0.001
12:35:04 pm 4/4	20	0.0005
12:25:08 pm 4/4	200	0.00075
12:24:32 pm 4/4	50	0.0002
12:23:56 pm 4/4	100	0.001

Some DEXes and DEFI examples.



Would you like to start Crypto Wallets for Ethereum support?

Total market size
$ 21,102,502,655.43

Assets	Market size	Total borrowed	Deposit APY	Variable Borrow APY	Stable Borrow APY
DAI	$ 2B	$ 1.59B	2.96 % 1.24 % APR	4.07 % 1.56 % APR	12.75 %
Gemini Dollar	$ 18.96M	$ 15.38M	7.14 % 1.63 % APR	9.91 % 2.01 % APR	—
sUSD	$ 80.07M	$ 53.66M	1.81 % 2.05 % APR	3.41 % 3.05 % APR	—
TrueUSD	$ 81.86M	$ 64.15M	3.04 % 1.00 % APR	4.00 % 1.34 % APR	12.70 %
USD Coin	$ 4.51B	$ 4.09B	6.22 % 1.09 % APR	7.60 % 1.23 % APR	15.40 %
USDP	$ 17.2M	$ 14.89M	3.04 % 2.38 % APR	3.92 % 2.75 % APR	—
USDT Coin	$ 1.33B	$ 1.21B	7.03 % 1.50 % APR	8.63 % 1.69 % APR	17.67 %