Blockchain Learning Group DApp Fundamentals

Sep 07, 2018

Contents

An immersive, hands on bootcamp in Blockchain technology, with a focus on decentralized application (DApp) development. From the basics of Blockchain technology to cutting edge smart contract development patterns and deployment techniques. Additionally, a deep dive into utilizing the Ethereum platform to build powerful and impactful applications.

- Participants leave with a strong working knowledge foundation of full DApp development.

Note: For further information please navigate to The Blockchain Learning Group

---

CHAPTER 1

Contents

1.1 Course Prerequisites

1.1.1 1.0 Course Resources

Note: Familiarity beforehand recommended.

1. Docker
2. ReactJS
3. Solidity
4. Truffle Framework
5. Web3JS
6. Ganache-cli
7. Material UI
8. Remix
9. Metamask

1.1.2 2.0 Machine Specs

Attention: Participants are required to bring their own laptops.

1. 4GB of memory and some disk space(4GB+) recommended.

2. Operating System: Ubuntu 16.04+ preferred, Mac and Windows 7+ OK(Mac preferred).

1.1.3 3.0 Machine Setup

3.1 Text Editor

1. Install the Sublime text editor
   - Download the editor here: https://www.sublimetext.com/3
   - Complete the installer steps
2. Install Sublime Package Control
   - Open the editor
   - `ctrl+shift+p` or `cmd+shift+p` (Mac)
   - Select install package control
3. Install the Ethereum package
   - `ctrl+shift+p` or `cmd+shift+p` (Mac)
   - Select install package
   - Search for and selct Ethereum

3.2 Browser

1. Google Chrome
   - Install the Google Chrome browser here.
   - Version > 55.0.0. Check in address bar: `chrome://version/`

OR

2. Brave
   - Install the browser here

3.3 Metamask

1. Install the chrome plugin, Metamask here
2. Once installed, share your address via the BLG slack channel.
   - Metamask extension for Brave Browser may also be enabled within their `default shield settings >> extensions`

3.4 Development Dependencies

1. Local Dockerized Environment
   - Follow the instructions here to configure your environment

[Windows users ONLY]

1. Git client

- Install git for windows here
- And to enable usage within windows command prompt execute the following within a prompt: `set PATH=%PATH%;"C:\Program Files\Git\cmd`
- Confirm git is configured correctly simply run: `git`

[MAC users ONLY]

1. Xcode

- You can find Xcode in the App Store: Xcode

1.2 Blockchain Fundamentals

1.2.1 1. ethstats.net

1.2.2 2. etherscan.io

1.2.3 3. ethernodes.org

1.2.4 4. Hash Function

- Run and attach into the container

```
$ docker run -dit --name=blg-env blockchainlg/dapp-dev-env
$ docker exec -it blg-env bash
# python3
>>> from sha3 import keccak_256
>>> keccak_256(bytes(1)).hexdigest()
bc36789e7a1e281436464229828f817d6612f7b477d66591ff96a9e064bcc98a

>>> keccak_256(bytes(2)).hexdigest()
54a8c0ab653c15bfb48b47fd011ba2b9617af01cb45cab344acd57c924d56798
```

1.2.5 5. Mining Script

- From within the docker container

```
# cd /blg
blg# python3 proof_of_work_mining.py 1
blg# python3 proof_of_work_mining.py 10
blg# python3 proof_of_work_mining.py 1000
blg# python3 proof_of_work_mining.py 100000
```

Note: Mainnet difficulty as of block `6035113` was `3,550,379,886,051,685` seen here

1.2.6 6. Bitcoin 51% Attack Cost

1.2.7 7. Remix

1.2.8 8. DappDeveloper.sol

1.2.9 9. Exceed Block Gas Limit

Add the below to DappDeveloper.sol

```
uint256 value_;

function reachGasLimit() {
  for (uint256 i = 0; i < 10**18; i++) {
      value_ = i;
      value_ = i + 1;
      value_ = i + 2;
  }
}
```

1.2.10 10. Voting Exercise

1.2.11 11. Token Exercise

1.2.12 Bonus

1. Deploy your token to a public Test Net(Kovan, Rinkeby, Ropsten)!

- Ensure Metamask is installed, enabled and unlocked
- Ensure Metamask is connected to Kovan via the drop down in the top left corner
- Within remix under the `run` tab switch from `Javascript VM` to `injected web3`
- Refresh the browser
- Now re-deploy and the contract will be sent from your Metamask account.

Note:

- A Metamask window will pop-up for you to confirm the transaction
- Also **SAVE** the address the token was deployed at! You may need it later :)

---

2. Sync an Ethereum node of your own

Note: Look to setup a node locally or via Azure. Azure is a nice option to begin with as a node locally can be quite heavy and resource intensive.

- Getting Started With Azure
- Sync a Parity node to Kovan

  - Instructions to deploy to Azure here
  - Parity Homepage
- Sync a Geth node to Rinkeby
  - Instructions here
  - Geth Homepage

1.3 Introduction to DApp Development

View Completed Wallet Demo

1.3.1 Stage 1: Dev Enviroment Setup and Application Bootstrap

---

Note: Begin instructions in a fresh terminal instance. Not within any existing window manager, ie. screen or tmux.

Video Tutorial

Important:

Replace <USERNAME> in ALL instructions below with your username. This is your machine's active user and can likely be foun

- Linux: `adam@ubuntu-box:~$`, `<USERNAME>` == `adam`
- Mac: `mac-box:~ adam1$`, `<USERNAME>` == `adam1`
- windows: `c:\users\adam2>`, `<USERNAME>` == `adam2`
- docker-machine: `adam3@DESKTOP-109 MINGW64`, `<USERNAME>` == `adam3`

---

1. Make a blg directory on your desktop

Important: This can be done by simply right-clicking on your desktop and creating a new folder named `blg`.

If you wish to do so from the command line the commands are as follows:

Linux, Mac and Docker Machine

```
cd ~/Desktop && mkdir blg
```

Windows

```
cd c:\Users\<USERNAME>\desktop && MD blg
```

- *Example output:*

```
adam@adam:/$ cd ~/Desktop && mkdir blg
adam@adam:~/Desktop$
```

- Now change into this directory from the command line

Linux, Mac and Docker Machine

```
cd ~/Desktop/blg
```

Windows

```
cd c:\Users\<USERNAME>\desktop\blg
```

2. Clone the wallet template

Important: Make **SURE** you are within the `blg` directory before cloning the repo!

```
git clone https://github.com/Blockchain-Learning-Group/wallet-template.git
cd wallet-template
git checkout tags/2.0
```

- *Example output:*

```
adam@adam:~/Desktop/blg$ git clone https://github.com/Blockchain-Learning-Group/
↪wallet-template.git
Cloning into 'wallet-template'...
[..]
Unpacking objects: 100% (30/30), done.
Checking connectivity... done.

adam@adam:~/Desktop/blg$ cd wallet-template

adam@adam:~/Desktop/blg/wallet-template$ git checkout tags/2.0
Note: checking out 'tags/2.0'.
[...]
HEAD is now at 16aa5a3...
adam@adam:~/Desktop/blg/wallet-template$
```

3. Run your docker container

Important:

- Make sure that the path immediately following the `-v` flag is correct! ie. `/home/adam/Desktop/blg`
- This path must exist on your host and the `blg` directory must contain the `wallet-template` repo.
- Also, take extra care and ensure that the path is correct for your OS.

Attention:

- If you previously ran the container to confirm the prerequisites were completed then first stop and remove this test container.

```
docker stop blg-env && docker rm blg-env
```

- *Example output:*

```
adam@adam:~/Desktop/blg$ docker stop blg-env && docker rm blg-env
blg-env
blg-env
adam@adam:~/Desktop/blg$
```

Linux

```
docker run -dit -p 3000:3000 -p 8545:8545 -v /home/<USERNAME>/Desktop/blg:/blg --
↪name=blg-env blockchainlg/dapp-dev-env
```

Mac

```
docker run -dit -p 3000:3000 -p 8545:8545 -v /Users/<USERNAME>/Desktop/blg:/blg --
↪name=blg-env blockchainlg/dapp-dev-env
```

Windows - If you have not already, follow the steps here to share your C drive with docker.

```
docker run -dit -p 3000:3000 -p 8545:8545 -v c:/Users/<USERNAME>/desktop/blg:/blg --
↪name=blg-env blockchainlg/dapp-dev-env
```

Docker Machine

```
docker run -dit -p 3000:3000 -p 8545:8545 -v /c/Users/<USERNAME>/Desktop/blg:/blg --
↪name=blg-env blockchainlg/dapp-dev-env
```

- *Example output:*

```
adam@adam:~$ docker run -dit -p 3000:3000 -p 8545:8545 -v /home/adam/Desktop/blg:/blg␣
↪--name=blg-env blockchainlg/dapp-dev-env
1bb232a56e6868e2bc4dbeaf86405ec3ed892090809fcab1823cab38e8337dc1
adam@adam:~$
```

Attention: Common Error:

```
adam@adam:~/Desktop/blg$ docker run -dit -p 3000:3000 -p 8545:8545 -v /home/adam/
↪Desktop/blg:/blg --name=blg-env blockchainlg/dapp-dev-env
docker: Error response from daemon: Conflict. The container name "/blg-env" is␣
↪already in use by container
↪"9c52f3787e28c64b197e22ec509fb2a73cd5066543ec6345956e11b6e69ba41c". You have to␣
↪remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.
```

Solution:

```
docker stop blg-env && docker rm blg-env
```

- *Examle Output*

```
adam@adam:~/Desktop/blg$ docker stop blg-env && docker rm blg-env
blg-env
blg-env
adam@adam:~/Desktop/blg$
```

4. Attach into the container

Container will serve as your virtual environment.

```
docker exec -it blg-env bash
```

- *Example output:*

```
adam@adam:~$ docker exec -it blg-env bash
root@182d123ec039:/blg/wallet-template#
```

5. Install dependencies

Attention: *ONLY Docker Machine*

```
yarn --no-bin-links
yarn global add react-scripts
```

Mac, Linux, Windows NOT Docker Machine

```
yarn
```

- *Example output:*

```
root@2e9e0eda980d:~/blg/wallet-template# yarn
yarn install v1.2.0
[1/4] Resolving packages...
[....]
Done in 42.65s.
root@2e9e0eda980d:~/blg/wallet-template#
```

6. Start the app

Note: The videos will demonstrate a window manager being used, screen, and if preferred you may do so at this time as well, however the following instructions will assume this is not the case and will simply create multiple tabs within your terminal window.

```
CHOKIDAR_USEPOLLING=true yarn start
```

- *Example output:*

```
# CHOKIDAR_USEPOLLING=true yarn start
Starting the development server...

Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://172.17.0.2:3000/
```

(continues on next page)

(continued from previous page)

```
Note that the development build is not optimized.
To create a production build, use yarn build.
```

7. Load the app in chrome, localhost:3000

END Stage 1: Dev Enviroment Set up and Application Bootstrapped!

1.3.2 Stage 2: Testing Your Token

Video Tutorial

1. Create a new tab in your terminal window or a new terminal window for our Ethereum node

---

Note: While within the terminal window select File -> Open Terminal to create a new window.

To create a new tab from within a terminal window:

```
ctrl+shft+t
```

- *Example output: Result is a new empty terminal, in the same directory you were when you initially entered your container. This will push you out of the container.*

```
adam@adam:~/Desktop/blg$
```

2. Attach back into the container in the Etheruem node tab

```
docker exec -it blg-env bash
```

- *Example output:*

```
adam@adam:~/Desktop/blg$ docker exec -it blg-env bash
root@182d123ec039:/blg/wallet-template#
```

3. Start up your Ethereum node, ganache-cli

```
ganache-cli
```

- *Example output:*

```
# ganache-cli
Ganache CLI v6.0.3 (ganache-core: 2.0.2)
[...]
Listening on localhost:8545
```

4. Create a new window or tab for our Truffle commands

Note: While within the terminal window select File -> Open Terminal to create a new window.

To create a new tab from within a terminal window:

```
ctrl+shft+t
```

- *Example output: Result is a new empty terminal, in the same directory you were when you initially entered your container. This will push you out of the container.*

```
adam@adam:~/Desktop/blg$
```

5. Attach back into the container in the Truffle tab

```
docker exec -it blg-env bash
```

- *Example output:*

```
adam@adam:~/Desktop/blg$ docker exec -it blg-env bash
root@182d123ec039:/blg/wallet-template#
```

6. Create the Test Case

Note:

- contracts/Token.sol has been provided or do update it with the Token that was completed previously.
- Also one test file template has been provided in order to test the buy method was implemented correctly.

- Open the repo, `~/Desktop/blg/wallet-template`, in your text editor, atom, sublime or the like and we can get to coding!
- Open the test file within Sublime, `src/test/test_buy.js`
- Import the token's build artifacts, `src/test/test_buy.js` line 2

```
const Token = artifacts.require("./Token.sol")
```

- Define the owner account, note `truffle test` exposes the accounts array for us, line 6

```
const owner = accounts[0]
```

- Create a new instance of the token contract, line 10

```
const token = await Token.new({ from: owner })
```

- Specify the wei value of tokens you wish to purchase, line 13

```
const value = 100
```

- Send the transaction to the token's buy method, line 16

```
const txResponse = await token.buy({ from: owner, value })
```

- Pull the rate from the token, line 19

```
const rate = await token.rate()
```

- Compute the token amount to be minted to the buyer, line 22

```
const tokenAmount = value * rate
```

- Access the event object from the transaction receipt, line 25

```
const event = txResponse.logs[0]
```

- Assert the correct values were emitted, line 28-31

```
assert.equal(event.event, 'TokensMinted', 'TokensMinted event was not emitted.')
assert.equal(event.args.to, owner, 'Incorrect to was emitted.')
assert.equal(event.args.value, value, 'Incorrect value was emitted.')
assert.equal(event.args.totalSupply.toNumber(), tokenAmount, 'Incorrect totalSupply␣
↪was emitted.')
```

Ensure the state of the contract is updated correctly

- Assert the buyer's balance is correct, line 34-35

```
const balance = await token.balanceOf(owner)
assert.equal(balance.toNumber(), tokenAmount, 'Incorrect token balance.')
```

- Assert the total supply is correct, line 38-39

```
const supply = await token.totalSupply()
assert.equal(supply.toNumber(), tokenAmount, 'Incorrect total supply.')
```

7. Execute the Test Case

```
cd src && truffle test
```

- *Example output:*

```
# cd src && truffle test
Using network 'development'.
Contract: Token.buy()
  ✓ should buy new tokens. (133ms)
1 passing (148ms)
#
```

END Stage 2: Testing Your Token

1.3.3 Stage 3: Token Deployment

Video Tutorial

Note:

- A default, and required, initial migration script(src/migrations/1_initial_migration.js), has been included. Do *not* remove this script.

1. Write the Deployment Script

- Create a new file in order to deploy the token, src/migrations/2_deploy_contracts.js
  - Simply right-click on the migrations directory and create the new file.
- Import the token's artifacts, line 1

```
const Token = artifacts.require("./Token.sol");
```

- Define the owner account, note `truffle migrate` exposes the web3 object, line 2

```
const owner = web3.eth.accounts[0]
```

- Utilize truffle's deployer object in order to deploy an instance of the token, line 4-6

```
module.exports = deployer => {
  deployer.deploy(Token, { from: owner })
}
```

2. Deploy your Token

```
truffle migrate
```

- *Example output:*

```
# truffle migrate
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0x26ff3f480502a228f34363e938289c3164edf8bc49c75f5d6d9623a05da92dbf
  Migrations: 0x3e47fad1423cbf6bd97fee18ae2de546b0e9188a
Saving successful migration to network...
  ... 0x19a7a819df452847f34815e2573765be8c26bac43b1c10d3b7528e6d952ac02c
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying Token...
  ... 0x4a69e7840d0f96067964fb515ffea1a04a98fc5759849d3308584af4770c8f7b
  Token: 0xd58c6b5e848d70fd94693a370045968c0bc762a7
Saving successful migration to network...
  ... 0xd1e9bef5f19bb37daa200d7e563f4fa438da60dbc349f408d1982f8626b3c202
Saving artifacts...
#
```

END Stage 3: Token Deployment

1.3.4 Stage 4: Token Interface

Video Tutorial

1. Import the web3 library, `src/app.js` #line 5

```
import Web3 from 'web3'
```

2. Import the token build artifacts into the application, app.js#line 14

```
import tokenArtifacts from './build/contracts/Token.json'
```

3. Create a web3 connection to the local Ethereum node(ganache-cli), app.js#line 26

```
this.web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
```

4. Check if the connection was successful, app.js#line 28-30

```
if (this.web3.isConnected()) {

}
```

5. Detect the current network id that is connected, app.js#line 29-31

```
this.web3.version.getNetwork(async (err, netId) => {

})
```

6. Extract the recently deploy token address from the build artifacts, app.js#line 30-33

```
// Create a reference object to the deployed token contract
if (netId in tokenArtifacts.networks) {
  const tokenAddress = tokenArtifacts.networks[netId].address
}
```

7. Create a client side reference to the contract and save it in state, app.js#line 33-35

```
const token = this.web3.eth.contract(tokenArtifacts.abi).at(tokenAddress)
this.setState({ token })
console.log(token)
```

8. Refresh your chrome browser and open up the developer console

This can be accomplished by right-clicking anywhere in the chrome browser and in the dropdown selecting `inspect` or `inspect element` or by utilizing the shortcut: `ctrl+shift_i`.

View in the developer console the token instance is now present

- *Example output:*

```
Contract {_eth: Eth, transactionHash: null, address:
↪"0xd58c6b5e848d70fd94693a370045968c0bc762a7", abi: Array[20]}
```

END Stage 4: Token Interface

1.3.5 Stage 5: Load Available On-chain Accounts

Video Tutorial

1. Get the available accounts from the web3 connection, this is to wrap the existing token interface code, line 29 & 39

```
this.web3.eth.getAccounts((err, accounts) => {  // Line 29

})                                              // Line 39
```

2. Set the default account to use, line 30

```
const defaultAccount = this.web3.eth.accounts[0]
```

3. Load the available accounts into the user interface

- Import the Material UI MenuItem, line 8

```
import MenuItem from 'material-ui/MenuItem';
```

- Add an availableAccounts arrary into the app's state, line 21

```
availableAccounts: [],
```

- Append all accounts into the UI dropdown menu, line 34-41

```
// Append all available accounts
for (let i = 0; i < accounts.length; i++) {
  this.setState({
    availableAccounts: this.state.availableAccounts.concat(
      <MenuItem value={i} key={accounts[i]} primaryText={accounts[i]} />
    )
  })
}
```

4. Set the default account

- Add a defaultAccount variable to the state, line 22

```
defaultAccount: 0,
```

- Set the defaultAccount in the state when the drowdown value changes, line 86

```
this.setState({ defaultAccount })
```

END Stage 5: Load Available Accounts

1.3.6 Stage 6: Token Interaction - GET

Video Tutorial

1. Load the token metadata from the contract

- Add the token's symbol to the state, line 23

```
tokenSymbol: 0,
```

- Load the token's symbol, line 52-55

```
// Set token symbol below
token.symbol((err, tokenSymbol) => {
  this.setState({ tokenSymbol })
})
```

- Add the token's rate to the state, line 23

```
rate: 1,
```

- Load the token's rate, line 58-61

```
// Set wei / token rate below
token.rate((err, rate) => {
  this.setState({ rate: rate.toNumber() })
})
```

END Stage 6: Token Interaction - GET

1.3.7 Stage 7: Load Account Balances

Video Tutorial

1. Load the default account's ETH and Token balances, completing the `loadAccountBalances` method

- Confirm the token has been loaded, line 73-75

```
if (this.state.token) {

}
```

- Add tokenBalance to the state, line 24

```
tokenBalance: 0,
```

- Set the token balance, line 75-78

```
// Set token balance below
this.state.token.balanceOf(account, (err, balance) => {
  this.setState({ tokenBalance: balance.toNumber() })
})
```

- Add ethBalance to the state, line 23

```
ethBalance: 0,
```

- Set the eth balance, line 81-84

```
// Set ETH balance below
this.web3.eth.getBalance(account, (err, ethBalance) => {
  this.setState({ ethBalance })
})
```

- Call the `loadAccountBalances` method on load, line 67

```
this.loadAccountBalances(defaultAccount)
```

- Also load the balances whenever a new account is selected in the dropdown, line 111

```
this.loadAccountBalances(this.state.availableAccounts[index].key)
```

2. View the default account balances and token information in your browser!

END Stage 7: Load Available Account Balances

1.3.8 Stage 8: Purchasing Tokens

Video Tutorial

1. Add token amount to the state, line 21.

```
amount: 0,
```

2. Complete the method to buy tokens, sending a transaction to the token contract, line 99-104.

```
this.state.token.buy({
  from: this.web3.eth.accounts[this.state.defaultAccount],
  value: amount
}, (err, res) => {
  err ? console.error(err) : console.log(res)
})
```

3. In the GUI buy tokens with several available accounts.

Note: Note transaction hash in the developer console

Example transaction hash: `0x4b396191e87c31a02e80160cb6a2661da6086c073f6e91e9bd1f796e29b0c983`

---

4. Refresh the browser or select a different account and come back, and view the account's balance of shiny new tokens!

END Stage 8: Purchasing Tokens

1.3.9 Stage 9: Events

Video Tutorial

1. Add an event to listen for when tokens are transferred and reload the account's balances, line 94-99

```
// Watch tokens transfer event below
this.state.token.Transfer({ fromBlock: 'latest', toBlock: 'latest' })
.watch((err, res) => {
  console.log(`Tokens Transferred! TxHash: ${res.transactionHash} \n ${JSON.
↪stringify(res.args)}`)
  this.loadAccountBalances(this.web3.eth.accounts[this.state.defaultAccount])
})
```

2. Load the contract events, line 66

```
this.loadEventListeners()
```

3. Buy tokens and view the log confirmation in the developer console and token and ETH balance updated dynamically!

END Stage 9: Events

1.3.10 Stage 10: Transfer Tokens

Try this portion on your own! [Solution noted at the bottom]

The required components included:

1. Add the transferAmount and transferUser to the app's state.
2. Add the React transfer tokens form component.
3. Complete the transfer method to send the transfer transaction.

Finally transfer tokens between accounts and review balances.

END Stage 10: Transfer Tokens

1.3.11 Bonus: Extend Your Wallet

1. Metamask Integration

- Ensure Metamask is installed, unlocked and connected to the local client(localhost:8545).
- Fund your metamask account!

```
$ truffle console
truffle(development> web3.eth.sendTransaction({ from: web3.eth.accounts[0], to:
↪'METAMASK_ADDRESS', value: 1e18 })
```

- Transfer tokens to your metamask account(from within the application).
- Add a conditional to use the Metamask web3 provider if present, wallet-template/src/App.js#L35

```
if (window.web3)
    this.web3 = new Web3(window.web3.currentProvider)
else
```

- Refresh the browser and connect to your Metamask account. View your Metamask account now available within the application.

2. Sync an Ethereum node of your own

Note: Look to setup a node locally or via Azure. Azure is a nice option to begin with as a node locally can be quite heavy and resource intensive.

- Getting Started With Azure
- Sync a Parity node to Kovan
  - Instructions to deploy to Azure here
  - Parity Homepage
- Sync a Geth node to Rinkeby
  - Instructions here
  - Geth Homepage

3. Interact with your token that was deployed to Kovan

4. Interact with another participant's token on Kovan

5. Enable the wallet to support multiple ERC20 tokens

1.3.12 Clean up

1. Detach from the container

```
ctrl+d
```

2. Stop the container

```
docker stop blg-env
```

- *Example output:*

```
adam@adam:~/$ docker stop blg-env
blg-env
adam@adam:~/$
```

1.3.13 BONUS

1. Add withdraw functionality! Enable the token owner to withdraw the ETH put in to purchase tokens.
2. Complete the sections from Blockchain Fundamentals

1.3.14 Solutions

Stage 10: Transfer Tokens

Video Tutorial

1. Add the transferAmount and transferUser to the app's state, line 28 & 29.

```
transferAmount: '',
transferUser: '',
```

2. Add the React transfer tokens form component, line 150-161.

```
<div>
  <h3>Transfer Tokens</h3>
  <TextField floatingLabelText="User to transfer tokens to." style={{width: 400}}␣
↪value={this.state.transferUser}
    onChange={(e, transferUser) => { this.setState({ transferUser }) }}
  />
  <TextField floatingLabelText="Amount." style={{width: 100}} value={this.state.
↪transferAmount}
```

(continues on next page)

(continued from previous page)

```
    onChange={(e, transferAmount) => { this.setState({ transferAmount })}}
  />
  <RaisedButton label="Transfer" labelPosition="before" primary={true}
    onClick={() => this.transfer(this.state.transferUser, this.state.transferAmount)}
  />
</div>
```

3. Complete the transfer method to send the transfer transaction, line 117-124.

```
if (amount > 0) {
  // Execute token transfer below
  this.state.token.transfer(user, amount, {
    from: this.web3.eth.accounts[this.state.defaultAccount]
  }, (err, res) => {
    err ? console.error(err) : console.log(res)
  })
}
```

Complete Wallet Solution

- `git clone https://github.com/Blockchain-Learning-Group/wallet-eod2.git`
- `cd wallet-eod2`
- `git checkout tags/2.0`

1.4 DeXchange Project

View Completed DeXchange Demo

1.4.1 Stage 1: Restart Your Dev Environment and App

Note: Begin instructions in a fresh terminal instance. Not within any existing window manager, ie. screen or tmux.

Video Tutorial

Attention:

Docker Machine ONLY - if Docker shell exited

- Double-click the *Docker QuickStart* icon on your Desktop to restart docker machine.
- Execute everything following from within the Docker shell.

1. Start your container back up

- Confirm container is not already running

```
docker ps
```

- *Example output: Container IS running*

```
CONTAINER ID        IMAGE                          COMMAND             CREATED
↪   STATUS              PORTS                                                NAMES
fabb387d8088        blockchainlg/dapp-dev-env   "node"              15 hours ago
↪   Up 15 hours         0.0.0.0:3000->3000/tcp, 0.0.0.0:8545->8545/tcp   blg-env
```

- *Example output: Container is NOT running*

```
CONTAINER ID        IMAGE                          COMMAND             CREATED
↪   STATUS              PORTS                                                NAMES
```

If the conainter is NOT running continue, else move to step 2

```
docker start blg-env
```

- *Example output:*

```
adam@adam:~$ docker start blg-env
blg-env
adam@adam:~$
```

2. Attach into your container

Container will serve as your virtual environment.

```
docker exec -it blg-env bash
```

- *Example output:*

```
adam@adam:~$ docker exec -it blg-env bash
root@9c52f3787e28:/blg/wallet-template#
```

3. Start the app

```
CHOKIDAR_USEPOLLING=true yarn start
```

- *Example output:*

```
# CHOKIDAR_USEPOLLING=true yarn start
yarn run v1.2.0
$ react-scripts start
Starting the development server...

Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://172.17.0.2:3000/
```

(continues on next page)

(continued from previous page)

```
Note that the development build is not optimized.
To create a production build, use yarn build.
```

4. Create a new tab in your terminal window or a new terminal window for our Ethereum client

Note: While within the terminal window select File -> Open Terminal to create a new window.

To create a new tab from within a terminal window:

```
ctrl+shft+t
```

- *Example output: Result is a new empty terminal, in the same directory you were when you initially entered your container. This will push you out of the container.*

```
adam@adam:~/Desktop/blg$
```

5. Attach back into the container and start Etheruem node

```
docker exec -it blg-env bash
```

- *Example output:*

```
adam@adam:~/Desktop/blg$ docker exec -it blg-env bash
root@182d123ec039:/blg/wallet-template#
```

- start the node(emulator)

```
ganache-cli
```

- *Example output:*

```
root@182d123ec039:/blg/wallet-template# ganache-cli
Ganache CLI v6.0.3 (ganache-core: 2.0.2)
[...]
Listening on localhost:8545
```

6. Create a new window or tab for our Truffle commands

Note: While within the terminal window select File -> Open Terminal to create a new window.

To create a new tab from within a terminal window:

```
ctrl+shft+t
```

- *Example output: Result is a new empty terminal, in the same directory you were when you initially entered your container. This will push you out of the container.*

```
adam@adam:~/Desktop/blg$
```

- Attach back into the container

```
docker exec -it blg-env bash
```

- *Example output:*

```
adam@adam:~/Desktop/blg$ docker exec -it blg-env bash
root@182d123ec039:/blg/wallet-template#
```

7. Deploy your Token

```
cd src && truffle migrate
```

- *Example output:*

```
root@182d123ec039:/blg/wallet-template# cd src && truffle migrate
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  [..]
  Deploying Token...
  Token: 0xd58c6b5e848d70fd94693a370045968c0bc762a7
  [..]
Saving artifacts...
#
```

10. Load the app in chrome, localhost:3000

END Stage 1: Restart Your Dev Environment and App

1.4.2 Stage 2: Create the Exchange Contract

Video Tutorial

1. Create a new file 'line

```
Exchange.sol
```

2. Copy Exchange Template into the new file, wallet-template/src/contracts/Exchange.sol

3. Review the contents of the provided template.

END Stage 2: Create the Exchange Contract

1.4.3 Stage 3: Write the submitOrder Method

Video Tutorial

1. Ensure the exchange has been given a sufficient token allowance, line 31

```
require(Token(_bidToken).allowance(msg.sender, this) >= _bidAmount, "Insufficient␣
↪allowance given.");
```

2. Compute a `unique` id for the order, line 34

```
bytes32 orderId = keccak256(msg.sender, _bidToken, _bidAmount, _askToken, _askAmount);
```

3. Confirm this order does not already exist, line 35

```
require(orderBook_[orderId].askAmount == 0, "Order already exists."); // check for␣
↪existence, default to 0, assume no one is giving tokens away for free
```

4. Add the order to the order book, line 38-44

```
orderBook_[orderId] = Order({
  maker: msg.sender,
  bidToken: _bidToken,
  bidAmount: _bidAmount,
  askToken: _askToken,
  askAmount: _askAmount
});
```

5. Emit the order submitted event, line 47

```
emit OrderSubmitted(orderId, msg.sender, _bidToken,_bidAmount, _askToken, _askAmount);
```

END Stage 3: Write the submitOrder Method

1.4.4 Stage 4: Test the submitOrder Method

Video Tutorial

1. Create a new file wallet-template/src/test/test_submit_executeOrder.js

```
test_submit_executeOrder.js
```

2. Copy the test template into wallet-template/src/test/test_submit_executeOrder.js

Test Setup

3. Define the accounts to be used, maker and taker, line 12-13

```
const maker = accounts[0]
const taker = accounts[1]
```

4. Deploy a new exchange and token in the test case, line 19-20

```
exchange = await Exchange.new()
token = await Token.new()
```

5. Define the order parameters, line 25-29

```
const rate = await token.rate()
const bidToken = token.address
const bidAmount = 100
const askToken = 0
const askAmount = 100
```

6. Setup the transaction by minting tokens to the maker and giving allowance to the exchange, line 34-35

```
await token.buy({ from: maker, value: bidAmount / rate });
await token.approve(exchange.address, bidAmount, { from: maker })
```

7. Send the transaction submitting the order, line 40-44

```
const tx = await exchange.submitOrder(bidToken, bidAmount, askToken, askAmount, {
    from: maker,
    gas : 4e6
  }
)
```

Assertions

8. Confirm the correct event emitted, line 49-50

```
const log = tx.logs[0]
assert.equal(log.event, 'OrderSubmitted', 'Event not emitted')
```

9. Confirm the order stored on-chain is correct, line 55-61

```
orderId = tx.logs[0].args.id
const order = await exchange.orderBook_(orderId)
assert.equal(order[0], maker, 'maker incorrect')
assert.equal(order[1], bidToken, 'bid token incorrect')
assert.equal(order[2], bidAmount, 'bid amount incorrect')
assert.equal(order[3], askToken, 'ask token incorrect')
assert.equal(order[4], askAmount, 'ask amount incorrect')
```

10. Execute the test and confirm it is passing!

```
truffle test test/test_submit_executeOrder.js
```

- *Example output:*

```
# truffle test test/test_submit_executeOrder.js
Contract: Exchange.submitOrder() && executeOrder()
✓ submitOrder(), should succeed by adding a new order to the orderBook on-chain.␣
↪(183ms)
✓ executeOrder(), should succeed by trading the tokens. Maker bids ether.


2 passing (365ms)

  #
```

END Stage 4: Test the submitOrder method

1.4.5 Stage 5: Write the executeOrder method

Try this part on you own! Solutions at the bottom...

1.4.6 Stage 6: Test the executeOrder method

Try this part on you own! Solutions at the bottom...

Note: This will fail at first and there is a bug located in `Token.sol's transferfrom` method for you(you are welcome!) Take a close look at line 76: `require(_amount <= 0, 'Cannot transfer amount <= 0, Token.transferFrom()');`

1.4.7 Stage 7: Deploy the Exchange

Video Tutorial

1. Add the exchange to the deployment script(`src/migrations/2_deploy_contracts`), line

- Import the exchange artifacts, line 2

```
const Exchange = artifacts.require("./Exchange.sol");
```

- Deploy the Exchange, line 6

```
deployer.deploy(Exchange, { from: owner })
```

2. Deploy the exchange(a new token).

```
truffle migrate --reset
```

- *Example output:*

```
# truffle migrate --reset
Using network 'development'.

Running migration: 1_initial_migration.js
  Replacing Migrations...
  ... 0xaf3df4616497a63d75879d900ee9bd580881e3d88b359942aa89beb12ff05416
  ----------------------------------------------
  Migrations: 0x4d52502c81f1b7119a59d7a69ca8b061d557e071
Saving successful migration to network...
  ... 0xa57ed9864bf4a34835ad0f074083030011e9f36aae813b58182f7d8cde8d4571
  ----------------------------------------------
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Replacing Token...
  ... 0xfb84339717eebb27f7593d5419633086c6961a46736d9f730185f9584bbca671
  ----------------------------------------------
  Token: 0x1f8fbc989937346cbc923da292b1b6f9f958eafe
  Deploying Exchange...
  ... 0xd4566da630267b7f41a554b3773ea4c2880d98828275632e4c9e6fd7f8d26b03
  ----------------------------------------------
  Exchange: 0xb9d7ffb8c064384f167199025ef2ad0a130c49c6
Saving successful migration to network...
  ... 0x97f51a0d5d97de1bf4d3f5028783349616fa25e0ddbadadecafe76fb1895189d
  ----------------------------------------------
Saving artifacts...
#
```

END Stage 7: Deploy the Exchange

1.4.8 Stage 8: Add Basic Routing to the DApp

1. Add basic routing to render navigate between the exchange and wallet components

Video Tutorial

- Add the `react-router-dom` package to the project

```
yarn add react-router-dom@4.3.1
```

- *Example output:*

```
root@0121f7449409:/blg# yarn add react-router-dom@4.3.1
yarn add v1.2.0
[1/4] Resolving packages...
[..]
Done in 5.34s.
root@0121f7449409:/blg#
```

- Import the router components into the app, line 2

```
import { BrowserRouter, Route, Link } from 'react-router-dom'
```

- Wrap components with the router, line 172 & line 179

```
<BrowserRouter>
</BrowserRouter>
```

- Add a button to navigate to the exchange route, line 137-139

```
<Link to={'exchange'}>
  <RaisedButton label=">>> Exchange" secondary={true} fullWidth={true}/>
</Link>
```

- Confirm selection of the new button will change the route in the url to `/exchange`

2. Create the exchange component and the routes

Video Tutorial

- Add a template exchange component with a link back to the wallet, line 173-177

```
const exchange = <div>
  <Link to={'/'}>
    <RaisedButton label="Wallet <<<" primary={true} fullWidth={true}/>
  </Link>
</div>
```

- Add a default route, line 186

```
<Route exact={true} path="/" render={() => component}/>
```

- And an exchange route, line 187

```
<Route exact={true} path="/exchange" render={() => exchange}></Route>
```

END Stage 8: Add Basic Routing to the DApp

1.4.9 Stage 9: Create the Reference Exchange Object

Look to follow the exact same process used for the token. Solutions at the bottom...

Note: Some hints...

1. Build Artifacts
2. State attribute
3. Contract address
4. Contract interface
5. Web3 to create reference object
6. Load the object into state

1.4.10 Stage 10: Create the UI Component to Submit an Order

Video Tutorial

1. Add the components to load the active accounts, line 184-191

```
<h3>Active Account</h3>
<DropDownMenu maxHeight={300} width={500} value={this.state.defaultAccount} onChange=
↪{this.handleDropDownChange}>
  {this.state.availableAccounts}
</DropDownMenu>
<h3>Account Balances</h3>
<p className="App-intro">{this.state.ethBalance / 1e18} ETH</p>
<p className="App-intro"> {this.state.tokenBalance} {this.state.tokenSymbol}</p>
<br/>
```

2. Add the form to submit an order, line 192-207

```
<h3>Submit an Order!</h3>
<p>The default exchange supports only the pairing of {this.state.tokenSymbol} / ETH</
↪p>
<TextField floatingLabelText="Bid" style={{width: 75}} value={this.state.tokenSymbol}␣
↪/>
<TextField floatingLabelText="Amount" style={{width: 75}} value={this.state.bidAmount}
  onChange={(e, bidAmount) => this.setState({ bidAmount })}
/>
<TextField floatingLabelText="Ask" style={{width: 75}} value="ETH" />
<TextField floatingLabelText="Amount" style={{width: 75}} value={this.state.askAmount}
  onChange={(e, askAmount) => this.setState({ askAmount })}
/>
<br/>
<RaisedButton label="Submit" labelPosition="after" style={{width: 300}} secondary=
↪{true}
  onClick={() => this.submitOrder()}
/>
<br/>
<br/>
```

END Stage 10: Create the UI Component to Submit an Order

1.4.11 Stage 11: Create the Functionality to Submit an Order

Video Tutorial

1. Add the bid and ask amounts to the state, line 23-24

```
askAmount: 1,
bidAmount: 10,
```

2. Write the method to submit an order, line 142-162

```
// Submit a new order to the order book.
submitOrder() {
  const { askAmount, bidAmount, defaultAccount, exchange, token  } = this.state
  const from = this.web3.eth.accounts[defaultAccount]
  const gas = 1e6

  // First give the exchange the appropriate allowance
  token.approve(exchange.address, bidAmount, { from, gas },
  (err, res) => {
    if (err) {
      console.error(err)
    } else {
      console.log(res)
      // Submit the order to the exchange
      exchange.submitOrder(token.address, bidAmount, '0', askAmount*10**18, { from,␣
↪gas },
      (err, res) => {
          err ? console.error(err) : console.log(res)
      })
    }
  })
}
```

3. Buy tokens to ensure the account has a sufficient token balance.

4. Submit an order and view the transaction hashes(approve and submitOrder) in the browser developer console.

END Stage 11: Create the Functionality to Submit an Order

1.4.12 Stage 12: Listen for Submitted Order Events

Video Tutorial

1. Create an event listener for the order submitted event, line 113-117

```
this.state.exchange.OrderSubmitted({ fromBlock: 'latest', toBlock: 'latest' })
.watch((err, res) => {
  console.log(`Order Submitted! TxHash: ${res.transactionHash} \n ${JSON.
↪stringify(res.args)}`)
  this.loadAccountBalances(this.web3.eth.accounts[this.state.defaultAccount])
})
```

2. Submit an order and view the caught event.

END Stage 12: Listen for Submitted Order Events

1.4.13 Stage 13: Create the Order Book Table

Video Tutorial

1. Import Material UI table components, line 14

```
import { Table, TableBody, TableHeader, TableHeaderColumn, TableRow, TableRowColumn }␣
↪from 'material-ui/Table';
```

2. Add the order book to the state, line 31

```
orderBook: [],
```

3. Add the order book component, line 240-257

```
<h3>Order Book</h3>
<p>Select an order to execute!</p>
<RaisedButton label="Execute Order" labelPosition="after" style={{width: 300}}␣
↪secondary={true}
  onClick={() => this.executeOrder(this.selectedOrder)}
/>
<Table style={{ maxHeight: 500, overflow: "auto" }} fixedHeader={true}␣
↪multiSelectable={false}
  onRowSelection={r => { if (this.state.orderBook[r[0]]) this.selectedOrder = this.
↪state.orderBook[r[0]].key}}>
  <TableHeader>
    <TableRow>
      <TableHeaderColumn>Maker</TableHeaderColumn>
      <TableHeaderColumn>Bid Token</TableHeaderColumn>
      <TableHeaderColumn>Bid Amount</TableHeaderColumn>
      <TableHeaderColumn>Ask Token</TableHeaderColumn>
      <TableHeaderColumn>Ask Amount</TableHeaderColumn>
    </TableRow>
  </TableHeader>
  <TableBody> { this.state.orderBook } </TableBody>
</Table>
```

4. View new order book table in the ui.

END Stage 13: Create the Order Book Table

1.4.14 Stage 14: Add an Order to the Order Book When Submitted

Video Tutorial

1. Create an addOrder method, line 172-194

```
// Add a new order to the order book
addOrder(order) {
  const { orderBook, tokenSymbol } = this.state
  const { id, maker, askAmount, bidAmount } = order;

  // Confirm this order is not already present
  for (let i = 0; i < orderBook.length; i++) {
    if (orderBook[i].key === id) {
      console.error(`Order already exists: ${JSON.stringify(order)}`)
      return
    }
  }

  const row = <TableRow key={id}>
    <TableRowColumn>{maker}</TableRowColumn>
    <TableRowColumn>{tokenSymbol}</TableRowColumn>
    <TableRowColumn>{bidAmount.toNumber()}</TableRowColumn>
    <TableRowColumn>ETH</TableRowColumn>
    <TableRowColumn>{askAmount.toNumber() / 10**18 }</TableRowColumn>
  </TableRow>

  this.setState({ orderBook: [row].concat(orderBook) })
}
```

2. Add the order to the order book when the order submitted event is fired, line 119

```
this.addOrder(res.args)
```

3. Submit an order and view it added to the order book.

END Stage 14: Add an Order Element to the Table When Submitted

---

1.4.15 Stage 15: Select and execute an Order

Exactly as we sent a transaction to submit the order! Solutions at the bottom...

Note: Hint: first you will need to add an attribute to the state to hold the selected order!

1.4.16 Stage 16: Listen for executed order events

Video Tutorial

1. Add the method to remove the order from the order book table, line 218-230

```
// Remove an order from the orderBook.
removeOrder(orderId) {
  const { orderBook } = this.state

  for (let i = 0; i < orderBook.length; i++) {
    if (orderBook[i].key === orderId) {
      let updatedOrderBook = orderBook.slice();
      updatedOrderBook.splice(i, 1);
      this.setState({ orderBook: updatedOrderBook })
      return
    }
  }
}
```

2. Add an event to listen for executed orders, line 123-127

```
this.state.exchange.OrderExecuted({ fromBlock: 'latest', toBlock: 'latest' })
.watch((err, res) => {
  console.log(`Order Executed! TxHash: ${res.transactionHash} \n ${JSON.stringify(res.
↪args)}`)
  this.removeOrder(res.args.id)
})
```

3. Execute an order and see that it has been removed from the table.

END Stage 16: Listen for executed order events

1.4.17 Stage 17: Load the Order Book

Video Tutorial

1. Add a method to load the order book, line 238-253

```
// Load all orders into the order book via exchange events
loadOrderBook() {
  const { exchange } = this.state

  exchange.OrderSubmitted({}, {fromBlock: 0, toBlock: 'latest'})
  .get((err, orders) => {
    for (let i = 0; i < orders.length; i++) {
      // confirm the order still exists then append to table
      exchange.orderBook_(orders[i].args.id, (err, order) => {
        if (order[4].toNumber() !== 0) {
          this.addOrder(orders[i].args)
        }
      })
    }
  })
}
```

2. Load the order book when the page renders, line 81

```
this.loadOrderBook()
```

3. View the loaded orders in the order book table.

Success your exchange running locally is complete! Try it out!

1.4.18 Bonus: Extend Your Exchange

1. Sync a node of your own! Instructions can be found here
2. Add other ERC20 / ETH pairings
3. Enable ERC20 / ERC20 pairings
4. Automated order matching, partial fills, matched by ratio not user selected.
5. Write tests for the exchange and token, failure cases
6. Update gas amounts sent with each transaction. Leverage web3's gas estimation!
7. Sort the orders in the order book table

1.4.19 Clean up

1. Detach from the container

```
ctrl+d
```

2. Stop the container

```
docker stop blg-env
```

- *Example output:*

```
adam@adam:~/$ docker stop blg-env
blg-env
adam@adam:~/$
```

1.4.20 Solutions

State 5: Write the executeOrder method

Video Tutorial

1. Load the order struct into memory(will save gas cost for subsequent reads), line 53

```
Order memory order = orderBook_[_orderId];
```

2. Confirm enough ether was sent with the transaction to fill the order, line 56

```
require(msg.value == order.askAmount);
```

3. Execute the trade.

- Moving ether to the maker, line 59

```
order.maker.transfer(order.askAmount);  // safe and will throw on failure
```

- AND tokens to the taker, line 60

```
require(Token(order.bidToken).transferFrom(order.maker, msg.sender, order.bidAmount),
↪"transferFrom failed.");
```

4. Remove the filled order from the order book, line 63

```
delete orderBook_[_orderId];
```

5. Emit the order executed event, line 66

```
emit OrderExecuted(_orderId, order.maker, msg.sender, order.bidToken, order.bidAmount,
↪ order.askToken, order.askAmount);
```

END Stage 5: Write the executeOrder method

Stage 6: Test the executeOrder method

Video Tutorial

Test Setup

1. Get the initial ether balances for both accounts, line 68-69

```
const makerBalanceBefore = web3.eth.getBalance(maker).toNumber()
const takerBalanceBefore = web3.eth.getBalance(taker).toNumber()
```

2. Submit the transaction to execute the order, line 74-79

```
const tx = await exchange.executeOrder(orderId, {
    from: taker,
    gas : 4e6,
    value: 100 // ask amount from previously submitted order
  }
)
```

Assertions

3. Confirm the execute order event emitted, line 84-85

```
const log = tx.logs[0]
assert.equal(log.event, 'OrderExecuted', 'Event not emitted')
```

4. Confirm the token balances updated correctly, line 90-93

```
const makerTokenBalance = (await token.balanceOf(maker)).toNumber()
const takerTokenBalance = (await token.balanceOf(taker)).toNumber()
assert.equal(makerTokenBalance, 0, 'Maker token balance incorrect.')
assert.equal(takerTokenBalance, 100, 'Taker token balance incorrect.')
```

5. Confirm the ether balances updated correctly, line 98-102

```
const makerBalanceAfter = web3.eth.getBalance(maker).toNumber()
const takerBalanceAfter = web3.eth.getBalance(taker).toNumber()
assert.equal(makerBalanceAfter, makerBalanceBefore + 100, 'Maker eth balance incorrect
↪')
// Note taker also had to pay for the executeOrder tx
assert.isBelow(takerBalanceAfter, takerBalanceBefore - 100, 'Taker eth balance␣
↪incorrect')
```

6. Confirm the order was removed from the order book, line 107-108

```
const order = await exchange.orderBook_(orderId)
assert.equal(order[4], 0)
```

7. Fix the token's transferFrom method `src/contracts/Token.sol` line 76

```
require(_amount > 0, 'Cannot transfer amount <= 0, Token.transferFrom()');
```

8. Execute the test and confirm it is passing!

```
truffle test test/test_submit_executeOrder.js
```

- *Example output:*

```
# truffle test test/test_submit_executeOrder.js
Contract: Token.buy()
✓ should buy new tokens. (116ms)

Contract: Exchange.submitOrder() && executeOrder()
  ✓ submitOrder(), should succeed by adding a new order to the orderBook on-chain.␣
↪(298ms)
```

(continues on next page)

(continued from previous page)

```
  ✓ executeOrder(), should succeed by trading the tokens. Maker bids ether. (493ms)


3 passing (951ms)

#

.. success::
  Success, The exchange contract is complete!
```

END Stage 6: Test the executeOrder method

Stage 9: Create the Reference Exchange Object

Video Tutorial

1. Import the exchange build artifacts, line 17

```
import exchangeArtifacts from './build/contracts/Exchange.json'
```

2. Add the exchange to the state, line 27

```
exchange: null, // exchange contract
```

3. Create the reference object to the deployed exchange, line 61-64

```
const exchangeAddress = exchangeArtifacts.networks[netId].address
const exchange = this.web3.eth.contract(exchangeArtifacts.abi).at(exchangeAddress)
this.setState({ exchange })
console.log(exchange)
```

4. View the exchange object in the browser developer console.

END Stage 9: Create the Reference Exchange Object

Stage 15: Select and execute an Order

Video Tutorial

1. Add a selectedOrder attribute to the state, line 33

```
selectedOrder: null
```

2. Add a method to execute the selected order, line 199-216

```
// Execute a selected order
executeOrder(orderId) {
  if (orderId) {
    const { exchange } = this.state
    const from = this.web3.eth.accounts[this.state.defaultAccount]
    const gas = 1e6
```

(continues on next page)

(continued from previous page)

```
    // Get the ask amount of the order from the contract, ether to send along with␣
↪the tx
    exchange.orderBook_(orderId, (err, order) => {
      exchange.executeOrder(orderId, { from, gas, value: order[4] },
      (err, res) => {
        err ? console.error(err) : console.log(res)
      })
    })
  } else {
    console.error(`Undefined orderId: ${orderId}`)
  }
}
```

END Stage 15: Select and execute an order

1.5 Solidity Exercises

1.5.1 1. Voting Exercise

- View Final Solution Demo

Video Tutorial[1.1 - 1.3][no audio]

1.1 Copy the exercise over to remix.

1.2 Define the duration of the vote, Line 7

```
uint256 public constant VOTE_DURATION = 2 minutes;
```

1.3 Complete the castVote method, beginning on Line 36

- 1.3a When a vote has been cast increment that candidates total, Line 41

```
candidateTotals_[_candidate] += 1;
```

- 1.3b Create an event for when a vote is cast, Line 18

```
event VoteCast(address voter, string votedFor);
```

- 1.3c Emit an event that a new vote has been cast, Line 46

```
emit VoteCast(msg.sender, candidateIds_[_candidate]);
```

- 1.3d `Run` the contract, deploying within remix and test the castVote method.

Video Tutorial[1.4 - 1.6][no audio]

1.4 Complete the tallyVote method, starting at Line 59

- 1.4a Add a for loop to find the winner of the vote, Lines 61 - 65

```
for (uint8 i; i < candidates_.length; i++) {
  if (candidateTotals_[i] > candidateTotals_[currentWinner]) {
    currentWinner = i;
  }
}
```

- 1.4b Set the winner, Line 70

```
winner_ = candidateIds_[currentWinner];
```

- 1.4c Emit an event that the vote has completed, Line 75

```
emit VoteComplete(winner_);
```

1.5 Add other candidates to the vote, Line 32

```
candidates_.push("YOUR NAME");
candidateIds_[1] = "YOUR NAME";
```

1.6 `Run` the contract, deploying within remix and test the castVote method and tallyVote methods

- Confirm candidates
- Cast several votes and after each confirm the total for the candidate has increased
- Tally the vote before the duration has elapsed
- Tally the vote after the duration has and view the winner
- Attempt to cast votes after the duration has elapsed

1.5.2 2. Token Exercise

- View Final Solution Demo

Video Tutorial [2.1-2.6][no audio]

2.1 Copy the exercise over to remix.

2.2 Compile and deploy the contract. Confirm variables and methods are available.

2.3 Update the contract metadata to be your own! Line 8 & 9.

```
string public constant symbol = 'YOUR NAME';
string public constant name = 'YOUR NAME Token';
```

2.4 Specify the rate for the purchase of your token, line 14

```
uint public constant rate_ = 2;  // rate of token / wei for purchase
```

2.5 Complete the buy method.

- May purchase only with > 0 ETH, line 46

```
require(msg.value > 0, 'Cannot buy with a value of <= 0, Token.buy()');
```

- Compute the amount of tokens to mint, line 49

```
uint256 tokenAmount = msg.value * rate_;
```

- Update the total supply and the user's balance, line 52 & 53

```
totalSupply_ += tokenAmount;    // NOTE overflow
balances_[msg.sender] += tokenAmount; // NOTE overflow
```

- Finally emit events to notify the outside world, line 56 & 57

```
emit TokensMinted(msg.sender, msg.value, totalSupply_);
emit Transfer(address(0), msg.sender, msg.value);
```

2.6 Compile, deploy and confirm you can purchase your token. Confirm balance updated in `balances` mapping.

Video Tutorial [2.7-2.10][no audio]

2.7 Complete the transfer method.

- Ensure from address has a sufficient balance, line 70

```
require(balances_[msg.sender] >= _value, 'Sender balance is insufficient,␣
↪Token.transfer()');
```

- Update the from and to balances, line 73 & 74

```
balances_[msg.sender] -= _value;  // NOTE underflow
balances_[_to] += _value;  // NOTE overflow
```

- Finally emit an event of the transfer, line 77

```
emit Transfer(msg.sender, _to, _value);
```

2.8 Compile and deploy and confirm buy and transfer working.

2.9 Note error output if insufficient balance and other errors correct.

2.10 Usage

1. Purchase of tokens

2. Transfers

Important: But how can you get your hard earned ETH out of the contract that has been accumulating as tokens have been sold?!

2.11 Add a withdraw method, and claim the ETH sent to the contract! Line 102

- Solution below...

Important: Save this contract to disk if you wish to use it again! However a completed token will be made available should you wish.

1.5.3 Solutions

1. Voting Exercise Solution
2. Token Exercise Solution

2.11 Token Withdraw method

- Confirm only the owner may withdraw, line 104

```
require(msg.sender == owner_, "only the owner may withdraw");
```

- Transfer the balance of the contract(this) to the wallet, line 107

```
_wallet.transfer(address(this).balance);
```

1.6 Project Submission

1.6.1 1.0 Github Account

If you have not already please create a github account. Please do so at github.com.

1.6.2 2.0 Submit Your Project

Video Tutorial

Note: Replace USERNAME with your username below, the example below will use blockchainLG.

1. Navigate to the BLG/Projects repo.
2. In the top right corner click on the fork icon. This should create your own copy of the repo.
3. Clone this new copy onto your machine.

```
git clone https://github.com/USERNAME/projects.git
```

- *Example output:*

```
adam@adam:~/Desktop/blg$ git clone https://github.com/blockchainLG/projects.git
Cloning into 'projects'...
remote: Counting objects: 58, done.
remote: Compressing objects: 100% (48/48), done.
remote: Total 58 (delta 4), reused 52 (delta 2), pack-reused 0
Unpacking objects: 100% (58/58), done.
Checking connectivity... done.
adam@adam:~/Desktop/blg$
```

4. Create a directory to submit your project and copy the contents of your project into the directory.

Note: This may be done with any file browser as well, below are linux commands to do so.

```
mkdir projects/submissions/USERNAME
cp -a wallet-template/* projects/submissions/USERNAME/
```

5. Push your changes to your fork

```
cd projects
git add .
git commit -m "USERNAME submitting course project"
git push
```

6. Navigate back to the BLG/Projects repo.
7. Select new pull request
8. Select compare across forks
9. Select your fork
10. Enter a pull request message and create the PR!

Your project has been submitted and will be reviewed shortly!