Decentralized Applications (Dapps)
Course 3 of the specialization

Contents

Chapter 1

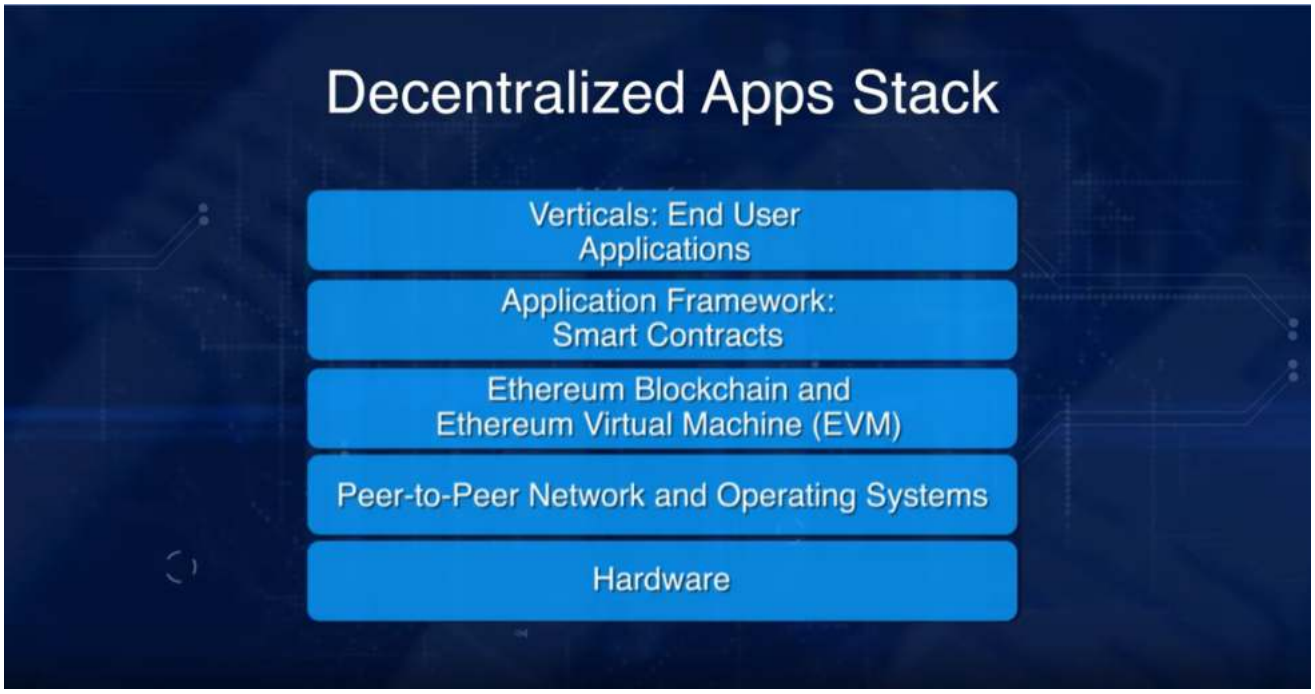
Decentralized Applications

1.1 Blockchain Server

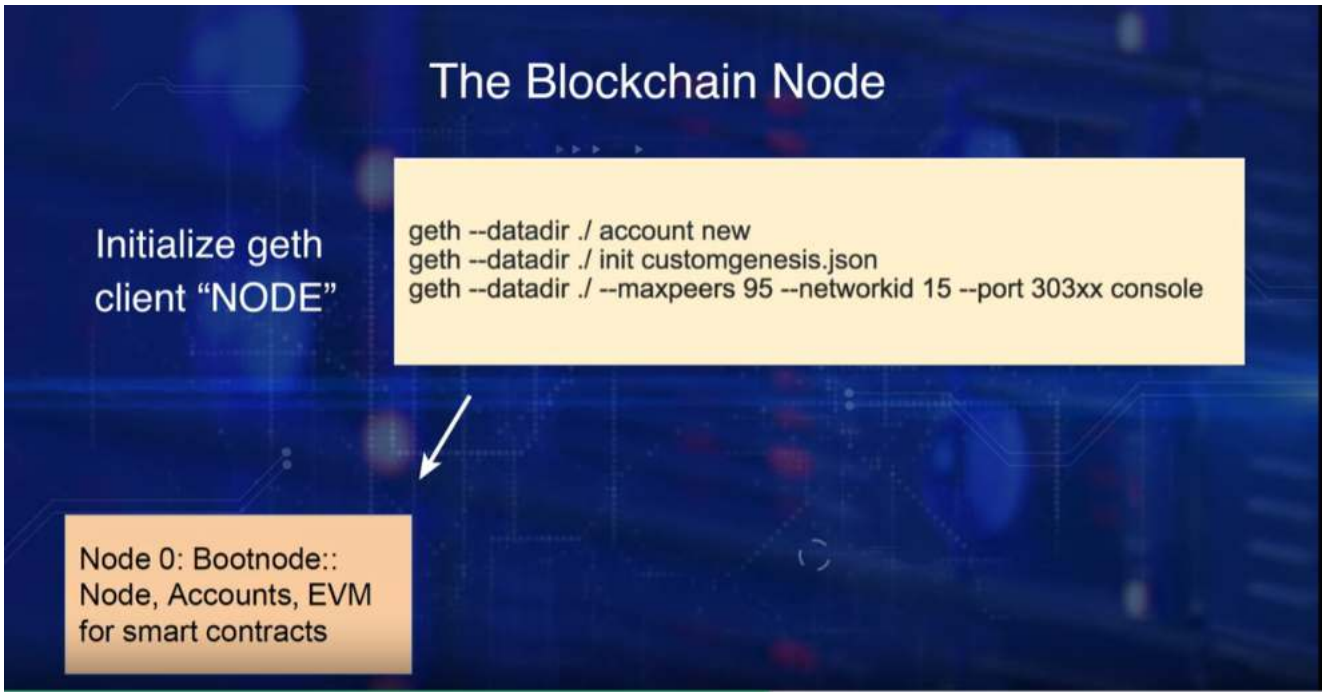
1.1.1

Decentralized applications or DApps, open up the features and services of the blockchain to the whole world for review, interaction, and enjoyment.DApp gives access to people, applications and systems, not necessarily known to each other to transact peer to peer. DApp is an end to end application on a blockchain.We open this course with essential concepts, such as the blockchain server. DApp architecture, and the details of the supporting application programming interfaces or the APIs. We'll go on to experience a hands-on, practical incremental end to end development of a DApp using the Truffle IDE. We will then explore a few smart contract specific best practices to improve the basic design of DApp. And finally, we'll review standards that are essential for transformation of an emerging technology into a mature evolving technology.On with this road map of the DApp development process, you will be prepared to chart your path through the emerging decentralized software culture. A DApp or a decentralized application depends on the functionality of a blockchain for its infrastructure and operations.In its simplest form, a DApp has a client interface as a front-end, and a back-end that includes a blockchain and a smart contract.Consider, for example, a simple wallet application client, and the Bitcoin blockchain decentralized infrastructure.This is similar to the architecture of a web browser and a web server, but with one significant difference. The blockchain enables a decentralized infrastructure.Remember, the blockchain is not in one centralized place, but in the computing nodes of the peer participants.The client or the front-end can be a web app, HTML and Javascript framework. A command line interface, CLI, a desktop application, a mobile application, or even an IoT, Internet of Things. Understand web front-end is outside the blockchain protocol, and it can only link into the blockchain smart contract, using artifacts generated by the smart contract compile process. Recall, we discuss these artifacts generated by Remix IDE in course two.Additionally, a DApp can be created with a non-blockchain back-end.Interplanetary file system, IPFS, is an example of such an architecture. On completion of this course, you will be able to discuss the architecture of a Dapp. Design and develop end to end decentralized application using Truffle IDE, smart contracts, a web client, and a meta mask client.Explore best practices in designing a smart contract in the context of the Dapps.Discuss Dapp models and explore emerging standards that are essential for predictable behavior of Dapps. [MUSIC] On completion of this module, you will be able to explain the architecture of the Ethereum blockchain server, and explain the high level architecture for Dapp. This is a high level of the Dapp's stack.It has a peer-to-peer network on which that blockchain operates. The node of a blockchain hosts the EVM and the smart contract runs on the EVM. The user interfaces of the Dapp runs on top of this layer, they use the smart contract for their logic.Next, we'll dig deeper into the architecture of the blockchain based Dapp. Using the actual commands that are used to create the architectural components of a Dapp. It is important that you have mastered the basic concepts from course one and course two. And have some basic knowledge of command line interface to get the most out of this course.We've been looking at bits and pieces of blockchain infrastructure in the last two courses. Now it's time to put them together, and have a grand view of the whole structure.We're going to coin the term blockchain server for representing the infrastructure and the functionality the blockchain provides.This nomenclature is drawn from the similarity in a web server web apps. Mobile server, mobile client apps, database server, database clients, etc. So the term, blockchain server for all the functionality it provides, Blockchain server Dapps. Here you see the set of commands that install the Ethereum services and the application programming interfaces API it exposes. To

enable node creation, application development, and transactions on the blockchain. At this point make note of the commands, you don't need to memorize the commands. Just try to understand the concept of the blockchain server, on the base on which the Dapps will be built. The term blockchain server also helps to differentiate the blockchain, the ledger from the protocol services. Now that you've established the foundation, let's create the node on the blockchain server. Here we create a single node, Node 0, at the genesis node for our blockchain server. The network ID for a blockchain server is 15 as you can observe in the third command line, it is still a private local chain. Are you curious to know the network IDs for the common public networks? That network ID for the Ethereum main network including metropolis, is 1. Ropsten, the public cross-client Ethereum testnet is 3, Rinkeby's network ID is 4. There is even a Musicoin, the music blockchain, with network ID 7762959. When you deploy a public blockchain, you should be aware of the network IDs of other public nets. Also, when you're in a private environment, you should be aware of the other network IDs in use if there are any. Think about the highway structure in the US, there can be only one number in state highway that can carry across state lines. For example, the interstate Route 90, there can not be another Route 90 across the nation. In the case shown, we are using geth command, provided by Ethereum server recently installed.

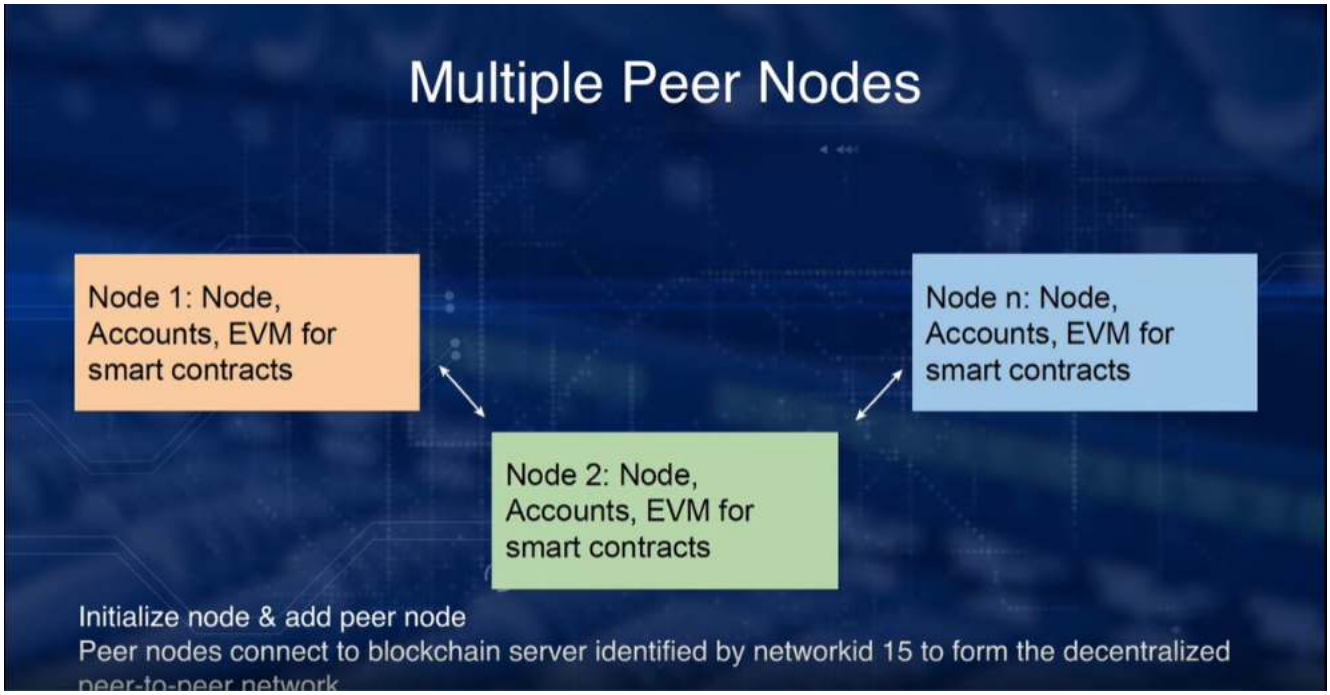


We use geth command to install a node by creating a new account number an externally owned account, EOA. Recall that we have discussed EOAs in earlier courses. We then initialize the node using a custom genesis block that has a specification, the first block of the chain. Finally, we specify the network ID and the port to buy into for the node that interact with the blockchain. At the bottom, you will see the Ethereum node, enode address for this genesis node or the bootnode created. Then enode address is used by other nodes to connect to this bootnode, and establish a peer-to-peer network on which the blockchain operates. Recall that a peer-to-peer network is one of the fundamental concepts in blockchain technology.



We have not listed all the commands needed for the networking details. That is beyond the scope of this course, the focus of which is on Dapp development. We have completed the creation of one node, let's now create at least one more peer node. Here we are creating node one, and adding it as a peer node to the bootnode we created earlier. After creation of this node, observe that we are using admin APIs, addpeer command, the fourth command, and the enode address of the bootnode we created earlier. The IP address and the binding port of the bootnode, 303xx, which represents the port number of the original node 0 that we create. This is how peer nodes are created and connected to join the blockchain network. Here you see multiple peer nodes 1 to n. When you want to join the blockchain network ID 15, install the base Etherium server. Then execute the commands as shown earlier for a single node creation, and add it as a peer.

Thus, we have a peer to peer network enabling the blockchain operation. Recall course one's project that used the same techniques, but the commands were hidden from you. There are alternate methods to create the blockchain network, and we have shown just one approach.



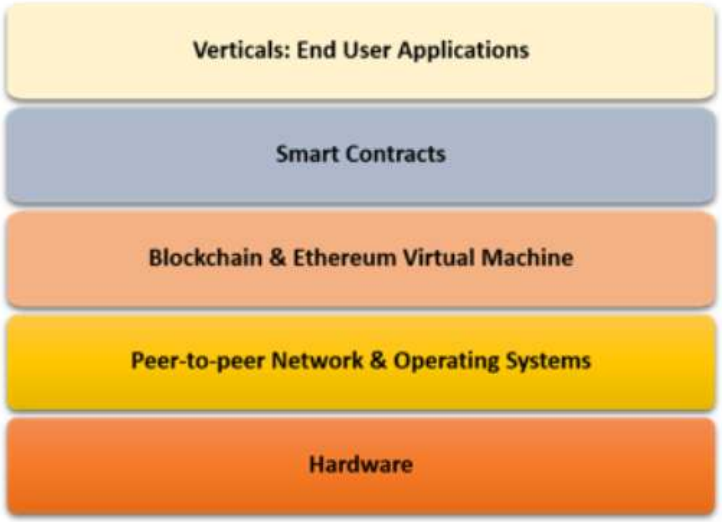
Summarizing, we have establish the notion of blockchain server as the foundation for a Dapp. We have demonstrated how to install the blockchain server and establish a peer-to-peer network of nodes. You explored an Ethereum blockchain server and a geth node, so that you can be an informed Dapp developer. It's a common practice to develop and test a Dapp on a local test network before deploying it on a public network. Now, let's move on to developing and testing a Dapp.

1.1.2 Resources

What is the Difference Between a Blockchain and a Database?
DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Decentralizing Everything with Ethereum's Vitalik Buterin — Disrupt SF 2017
What is and how to use the Ethereum Name Service (ENS)
Ethereum-ens

1.1.3 Quiz

1. What does the following figure represent?



- ◯ Service layer
- ◯ Blockchain stack
- ◯ Server
- ◯ Dapp stack

[scale = 0.9]pics/dapps-1-2.png

3. What does the enode address represent?

- ◯ Address of the ethereum node
- ◯ Addresses of the ethereum accounts
- ◯ Hash of your accounts
- ◯ Address of the coinbase

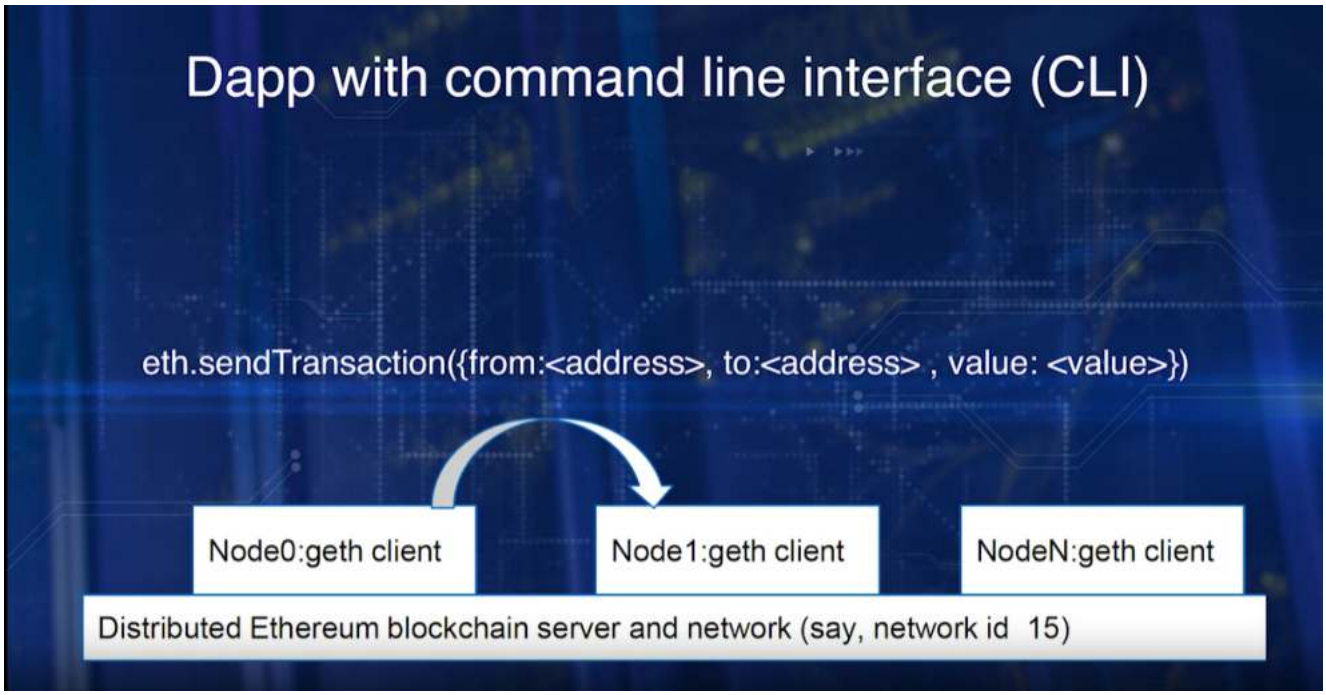
4. What is the use of enode address?

- ◯ Used by the system to communicate with its coinbase
- ◯ Used by nodes to connect and form a p2p network
- ◯ Used by account to receive money from others
- ◯ Used by account to send money to others

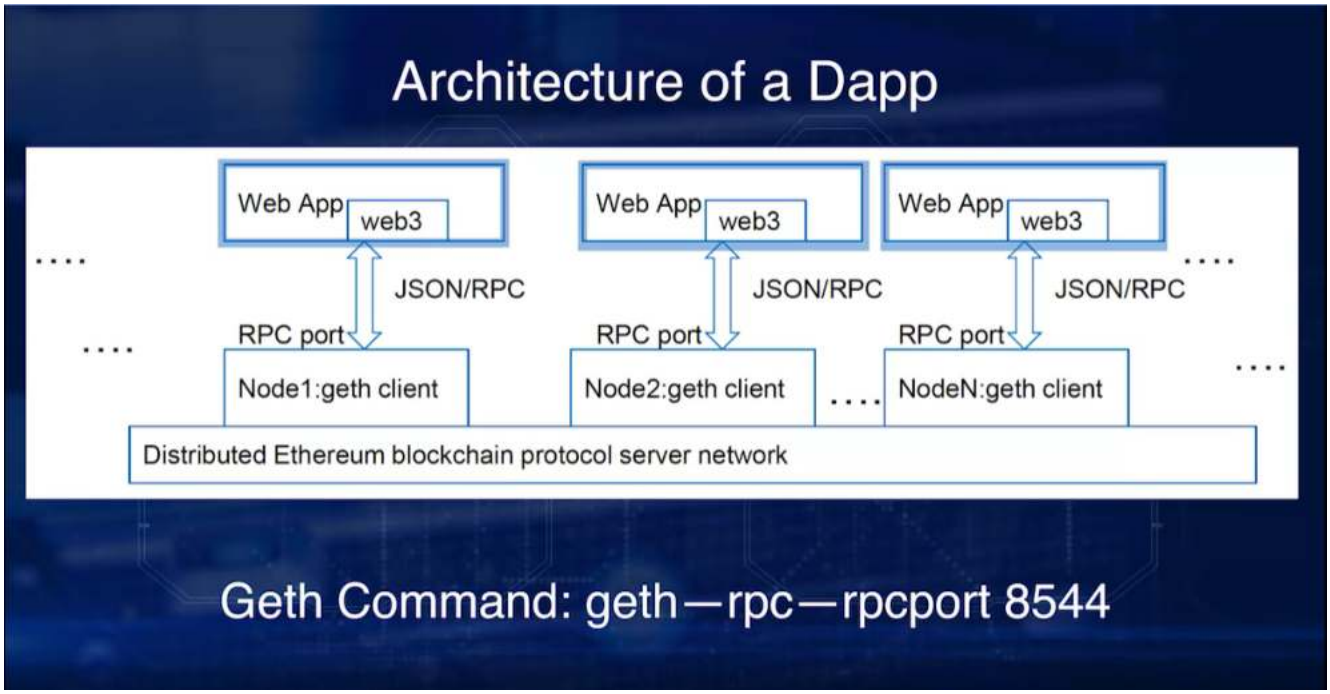
1.2 Dapp Defined

1.2.1

On completion of this lesson, you will be able to explain the architecture of the Dapp, explain at high level, the access from web client to the blockchain server. What is a Dapp? A Dapp, or decentralized application, solves a problem that requires blockchain services and blockchain infrastructure for realizing its purpose. Typically, a Dapp has a web front-end, and a blockchain back end, and the code connecting the two.



In such an architecture, the front-end of a Dapp channels any external stimulus from the users to the blockchain infrastructure and returns any response back to them. It initiates transactions to invoke functions on the smart contract. That in turn, records the transactions and state transition and receipts on the blockchain. The front-end of a Dapp can be as simple as a command line interface. It can also be a sophisticated web app or an easy to use mobile app. The front-end development may involve development of a web client with the HTML, JavaScript, and CSS's, and other web assets, or a web app framework, such as Express. Here, the blockchain server is the E-node on the underlying infrastructure. And the front-end is a web client with embedded web3.js script, communicating using JSON over RPC pipeline. Recall from course two that web3 deploy script is an important artifact generated by the remix compile process.



We'll use a simple web app as a front-end for the Dapps we develop in this course. If you are programmatically assembling the Dapp without the help of any development involvement, you copy the web3 deploy script into your app, say an HTML JavaScript file as a script. Deploy the smart contract using the functions in the script. And invoke the smart contract function using the functions in the script, and the ABI, and the smart contract address. At this time, if you're a programmer, you can review the web3 deploy script generated in the Remix IDE. Don't worry, we plan to use an IDE to simplify the development of a Dapp later on in the module two of this course. Thus far, we covered the foundational concepts of a Dapp. Now for the first time, we are looking at the architecture of a Dapp. Here is a simple Dapp architecture with the nodes that we created earlier with geth command and Ethereum blockchain server and the network. Once the nodes have been initialized with the same network, you can send transactions using the command line interface. Also, you should be able to deploy a smart contract from the command line. Be aware that these commands may have lengthy payloads and parameters that may span multiple lines. Moreover, command line interface may not be familiar to a non-programmer or a user of a Dapp. So we will use a simple but intuitive web interface as the front-end to the blockchain Dapp we are planning to develop. Here is an updated architecture for a Dapp. There are multiple full nodes, with only three of them, Node1, Node2, NodeN, shown. Geth command is used to expose RPC port 8544 using the commands geth–rpc–rpcport 8544. From the web client, the functions including smart contract deploy and invoke functions are facilitated through web3.js module. To summarize, we explored the installation of an Ethereum blockchain server, a single node, and a network of peer to peer nodes. We also examined two alternatives for the Dapp architecture. One with command line client and the other with a web client. In the next lesson, we'll explore some support APIs for developing a Dapp.

1.2.2 Resources

Decentralized Applications-dApps
The Future will be Decentralized

1.2.3 Quiz

1. What is the CLI command used to send ethers after the nodes have been initialized?

- ◯ **eth.sendTransaction()**
- ◯ **eth.sendIBANTransaction()**
- ◯ **eth.submitTransaction()**
- ◯ **eth.sendRawTransaction()**

2. In this lesson, what port do we expose the RPC for the Ethereum network on?

- ○ 8544
- ○ 7545
- ○ 8545
- ○ 9545

3. To access smart contracts from the Dapp web interfaces we need the Web3 script. True or False?

- ○ True
- ○ False

1.3 Ethereum APIs

1.3.1

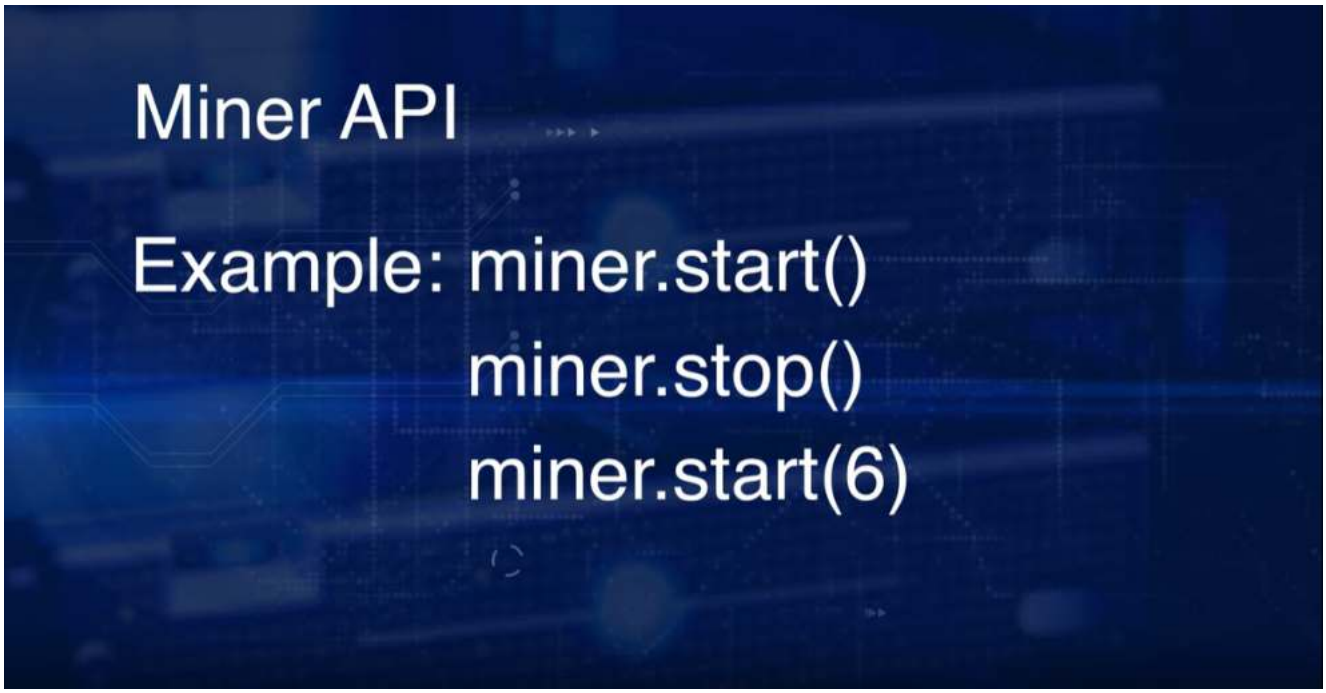
What is an API? API, or Application Programming Interface, is a convenient and standard way to expose a set of functions related to a specific dataset and services. APIs also lend to reusability of code. An API publishes a set of functions or methods that can be used programmatically to invoke operations, access data, and store data. Access to an API can be controlled by specific access method, for example, public keys, if so warranted by the application. On completion of this lesson you will be able to list the Ethereum APIs that facilitate Dapp development, explain the functions and usage details of the various APIs. Two well-known examples of APIs outside Ethereum blockchain are the Twitter API to access tweets that can be filtered by query terms, Google Map APIs that allow for applications to embed the map features, such as geolocation, in their own applications, leveraging and reusing the power of Google Map API. Why are APIs important? Blockchain server and node provide the blockchain functionality and data structures. How do applications call the functions of a blockchain, invoke them? How do they use them for accomplishing tasks of an application? We need a well-defined standard approach to get things done for a Dapp. The answer is in the APIs. In our case, the specific APIs expose the services of the blockchain server using standard functions. When you develop a Dapp, interactions with the geth node on the blockchain server is accomplished by invoking these APIs. For example, in the command miner.start(), miner is API, and start() is the function of the miner API. Why in the context of a Dapp I'll be learning about APIs?

Admin API

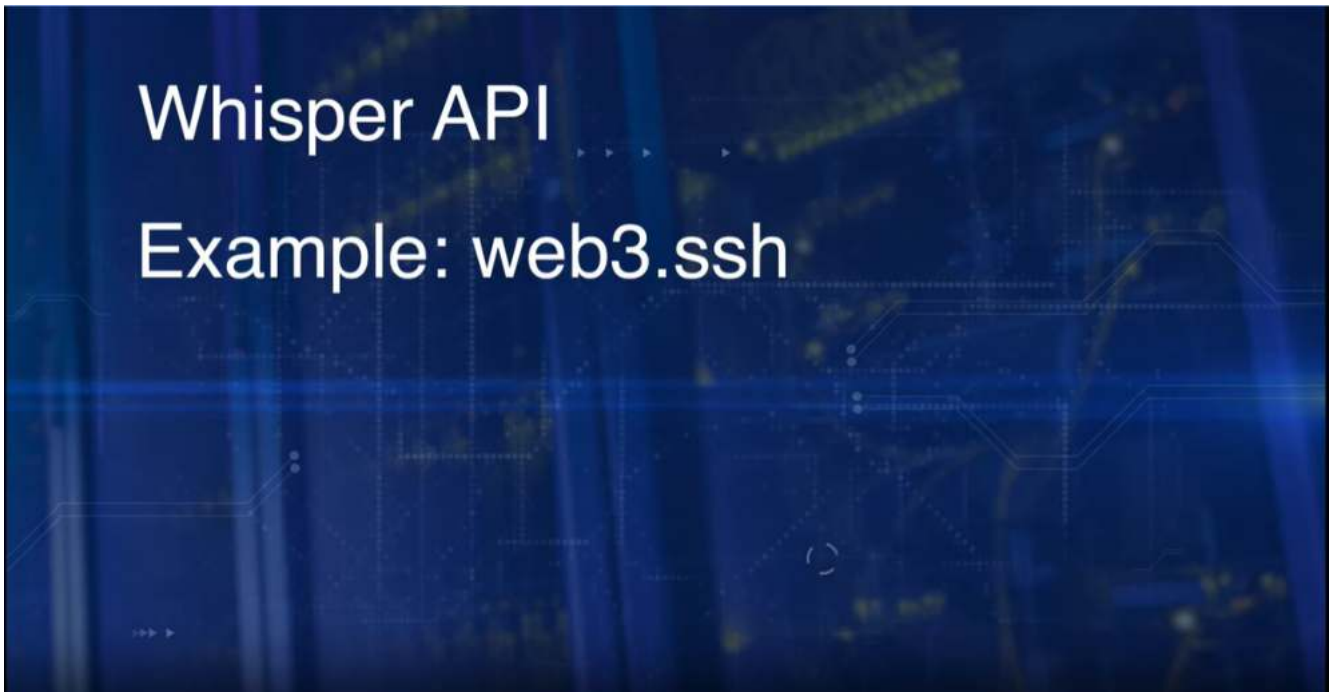
Example: admin.addPeer()
admin.nodeInfo()

Debug API

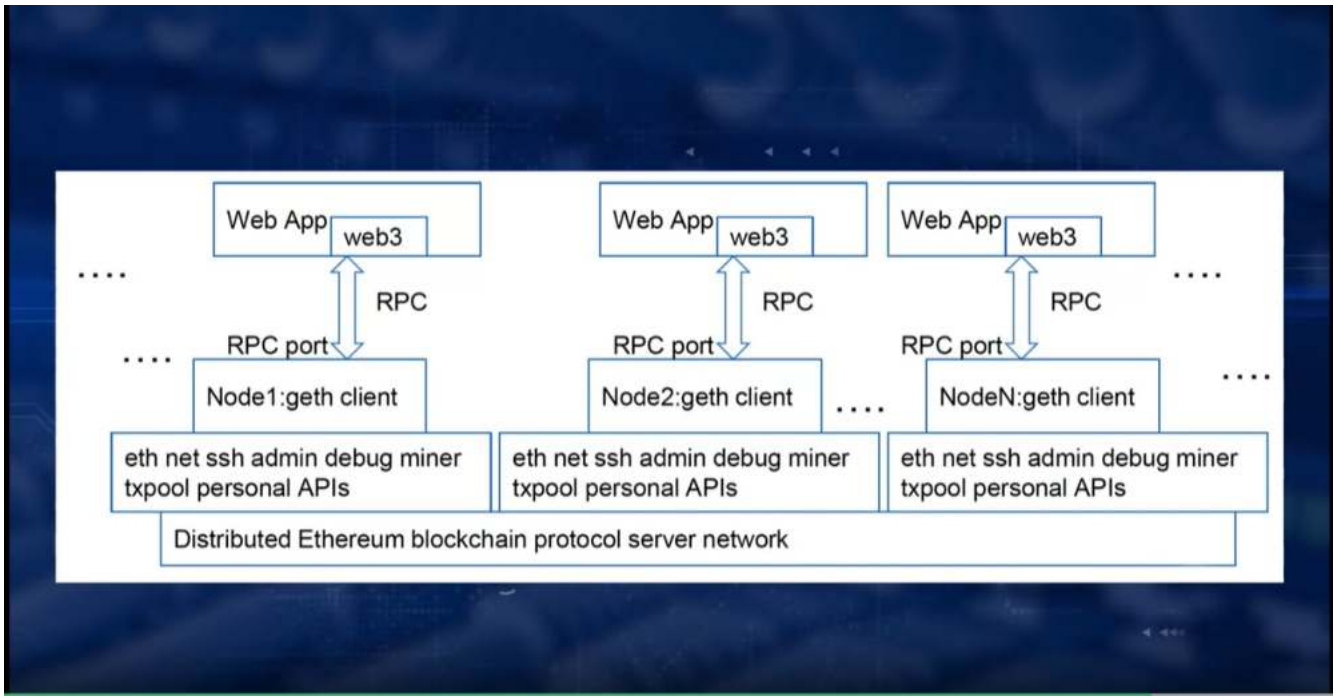
Example: debug.dumpBlock(16)


Miner API
Example: miner.start()
miner.stop()
miner.start(6)

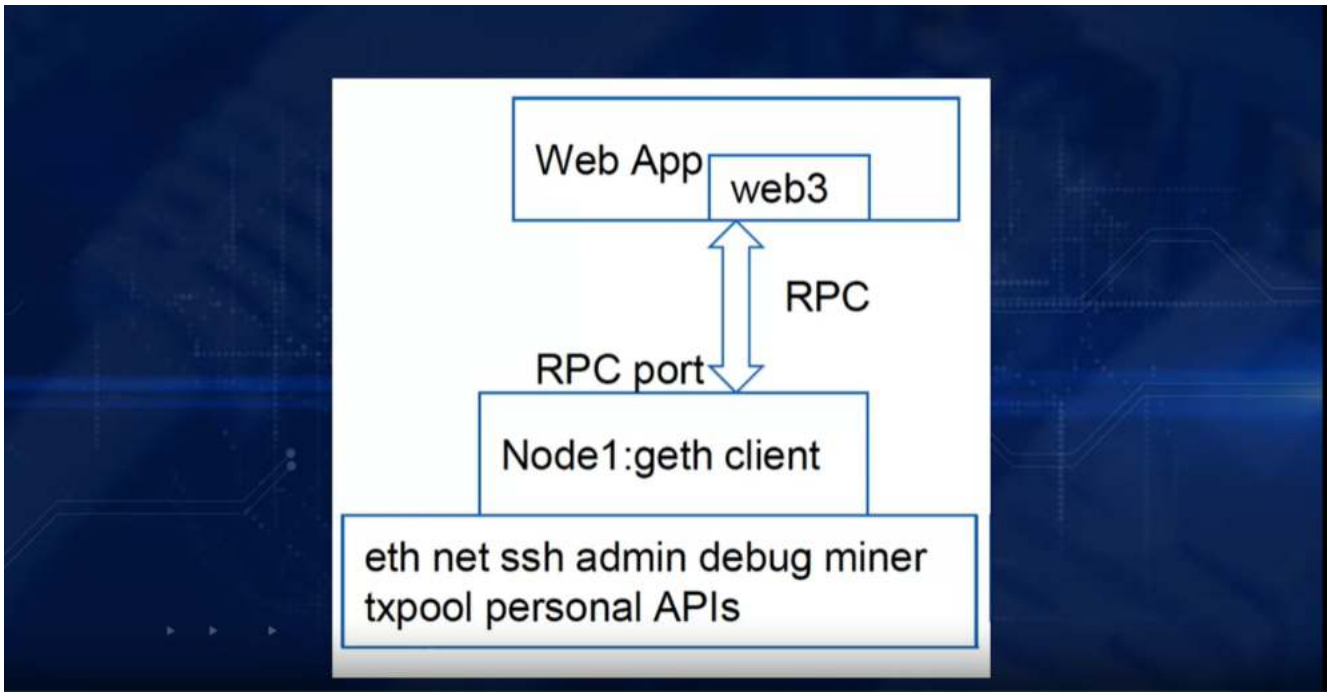

Personal API
Example: personal.newAccount()
Txpool API
Example: txpool.inspect()



Blockchain technology has ushered in a whole new culture in software and systems development and management. In the beginning, software systems were proprietary products. Eventually, some moved on to become open source. With blockchain technology, developers are not only contributers to software, many of them help run the network by standing up nodes, supporting mining, etc. More importantly, developers also propose an oath on improvements and changes to the blockchain protocol. Thus, we want you to be more than developers. You can be contributors, you can be committers to code, you can help the governance, and help shape the protocol. This is the technology culture underlying the blockchain revolution. To enable you to be informed developers in this context, you need a basic knowledge of the APIs and the inner workings. Ethereum APIs, so what are the APIs Ethereum provides? There are two major categories of APIs. The first one is the management APIs, that includes admin, debug, miner, personal, and txpool. They support methods for management of the geth node. The second one is the web3 API, web3, eth, and net. They support methods for development of Dapps.



Next, we'll examine each of the APIs in detail. Admin, the Admin API allows you to use functions to work with your Geth instance, including network peer and the RPC endpoint management. Examples, admin.addPeer(), admin.nodeInfo(). In this case, admin is the API, and addPeer and nodeInfo are functions of the admin API. You can observe that admin supports functions for management of the node. Debug API, example, Debug.dumpBlock(16). This will display the block header details of block 16. You can observe that the debug API provides you the ability to peek into the blockchain, study it, and debug any issues by looking at the block. Miner API, the miner API allows you to control the nodes mining operation and set various mining specific settings. Very easy to understand. Example, miner.start(), miner.stop() are sample functions that will start and stop the miner. You can also say miner.start(6), where 6 parallel threads are assigned to the mining operation. Personal API deals with the creation and management of accounts within a node. It also manages private keys in the key store, that's why it is called personal API. Example, personal.newAccount() will create a new account within a node. Txpool API, the Txpool, or transaction pool API, gives you access to several non-standard RPC methods to inspect the contents of the transaction pool, containing all the currently pending transaction, as well as those queued for future processing. Example, txpool.inspect() lists you all the pending transactions for you to peruse and collect for building a block of transactions. So far, we looked at the management APIs of Ethereum, now on to the web3 APIs. Web3.js library, when included into Dapp, lets you use web3 object provided by the web3.js library and all its objects. It also lets you communicate with a local node through the RPC port. It also provides access to the eth object and its function via web3.eth, and net object via web3.net, and their respective function. You can also access other management APIs to the web3 object. There is also Whisper API, web3.ssh. It is used for secure gossiping and enables the Whisper protocol.



Web3 is a JavaScript library that's specifically designed for use with web client or Ethereum Dapps. It's a portal through which all the underlying operations of the Ethereum node on the blockchain server can be invoked. For example, a Smart Contract deployment and Smart Contract function invocation. You will use web3 in the development in the front-end of the Dapp and for interacting with the blockchain. Now that we know the APIs, here is a simple architecture of a Dapp. It comprises multiple full nodes with only three of them, Node 1, 2, and N. You can also see all of the APIs discussed. Geth command is used to expose the RPC port, as well as the APIs. So far we discussed the architecture and API details. When a web request is initiated by a user, and if it's a regular web app request, it is directed to the HTTP endpoint, say port 8080, and to the web server to be executed. For a Dapp, the geth client has to expose an RPC endpoint using RPC port command. A web3 object is instantiated in the web page script. Recall that web3.js is a JavaScript library. Requests or calls are invoked on the web3 object. Requests are transmitted as a JSON or RPC pipeline between the web client and the geth client. Requested call and function is executed using appropriate API and Smart Contract code. And the result, return to the back client. As a recap, in this module, we introduce a way at looking at the blockchain technology, blockchain server, and client. We created a two node peer-to-peer blockchain network with ID 15. We learned the foundational knowledge required to be a developer of a Dapp and a blockchain citizen. We explored the Dapp architecture and the supporting APIs. Armed with all this knowledge, we are ready to develop our first Dapp.

1.3.2 Practitioner's Perspective: Public Network Architecture

The problem is those private networks are invisible. You don't know that they exists. There is no telling how many hyperledger fabric networks are out there today. We don't think that there is millions of them or thousands of them, but it's an honorable number because it's not any kind of subscripts like the public network, where you can actually do any kind of analysis on that. So what's needed is a registar. And we think that the public network is where that's going to live, just like you have DNS today and you can go and register a name. There is something on Ethereum called the ENX, right? The naming service for Ethereum. And that allows you to do some really cool things. In fact, students of this course should really look at the naming service on Ethreum, because it's a structure that allows you to some really interesting things around, not only identity, but around organization and things like access control and user management. The naming services is vastly underutilized compared to where it's going. It's very important piece of technology, the Ethereum main net. And with that and some other things, you can start to imagine having a registrar of both private networks and public ones. So from ConsenSys's point of view, I think and from Ethereum's point of view, there's a vision that some of us share that I think is right where the public main net becomes sort of the referee of millions of what we call en lateral peer-to-peer connections. You and me, us this room, bigger group, huge group, public group, right? And you

want all those to be able to coordinate so that our smart contract and they book other smart contracts, they can book other smart contracts, ultimately with a public network, but without invoking race conditions and all the other problems that the classic distributed systems have. So here you have this sort of a kind of time keeper, you have a timestamp on the block and you have this public massive public network that's pretty hard to corrupt, vanishingly small probability of corrupting, right? Not zero probability, but limiting the zero. And that public network, especially Ethereum, because it's got the attributes of the turn and complete system and it's more than a key value per storage system. It's a little more complex. It allows for more attributes. We think it's a pretty god referee for all these millions of this factorial of peer-to-peer connections, peerwise connections between individual nodes, right? And once you have that, then you can say, I want to register that. I'm going to register my node, my node and your node, our connection. I can register it, or I can say this is a network, or this is another bigger network. This is the supply chain for the automotive industry network, but here is the rubber providence supply chain or providence network for making sure that we don't have conflict materials in our rubber supply. All right, those are two different networks that created at different times without even knowing about each other. But if they're all coordinated by the public main net, then they can start to as long as we don't get the data models too wrong. We can start to bring those together and say, all right, we started here. But now we're going to start doing transactions with this larger thing. All right, I'm good here. I'm a tire manufacturer. I've got this rubber providence supply chain, but I'm part of the automotive supply chain. I need to be part of both networks. And I don't want to have two separate nodes that do one and the other separately. I want one set of nodes that does everything. If I'm setting up that consortium, I want to be able to register that as a thing that I can join on the main net. What tools are out there to build interesting solutions or to tackle these problems, right? The interesting thing is that a lot of these tools are just merging. There's a lot of opportunity to build new tools, so if you're a tool builder, now's your time. Get out there, tell us about it. My job is the seeker of awesomeness, so if you've got an awesome tool, come to me, something about it. Maybe we can turn you into a new venture. But there are tools and then there are tools. So there are tools like plasma and protocol level tools, and then there are tools on top of that, just like computer science has always been a layer, a set of abstraction layers all the way down and up, right? So here we are at this abstraction layer. We've got distributed systems. We have ways of managing those distributed systems. We have consensus layers and all these other things. We have the data layer and on top of that, we have this layer of enabling tools that help you build interesting solutions. Two of my favorites are uPort and OpenLaw.

1.3.3 Resources

World's First DAPI: Decentralized Application Programming Interface
Web3 JavaScript app API for 0.2x.x

1.3.4 Quiz

1. What does API stand for?

- ◯ **Application Performance Indicator**
- ◯ **Application Programming Interface**
- ◯ **Automatic Process Initiator**
- ◯ **Application Protocol Interface**

2. Which of these are Ethereum APIs?

- ☐ enode
- ☐ admin
- ☐ miner
- ☐ eth

3. What library/API is used for smart contract deployment and invocation from Dapp ?

- ○ contract
- ○ web3
- ○ eth
- ○ admin

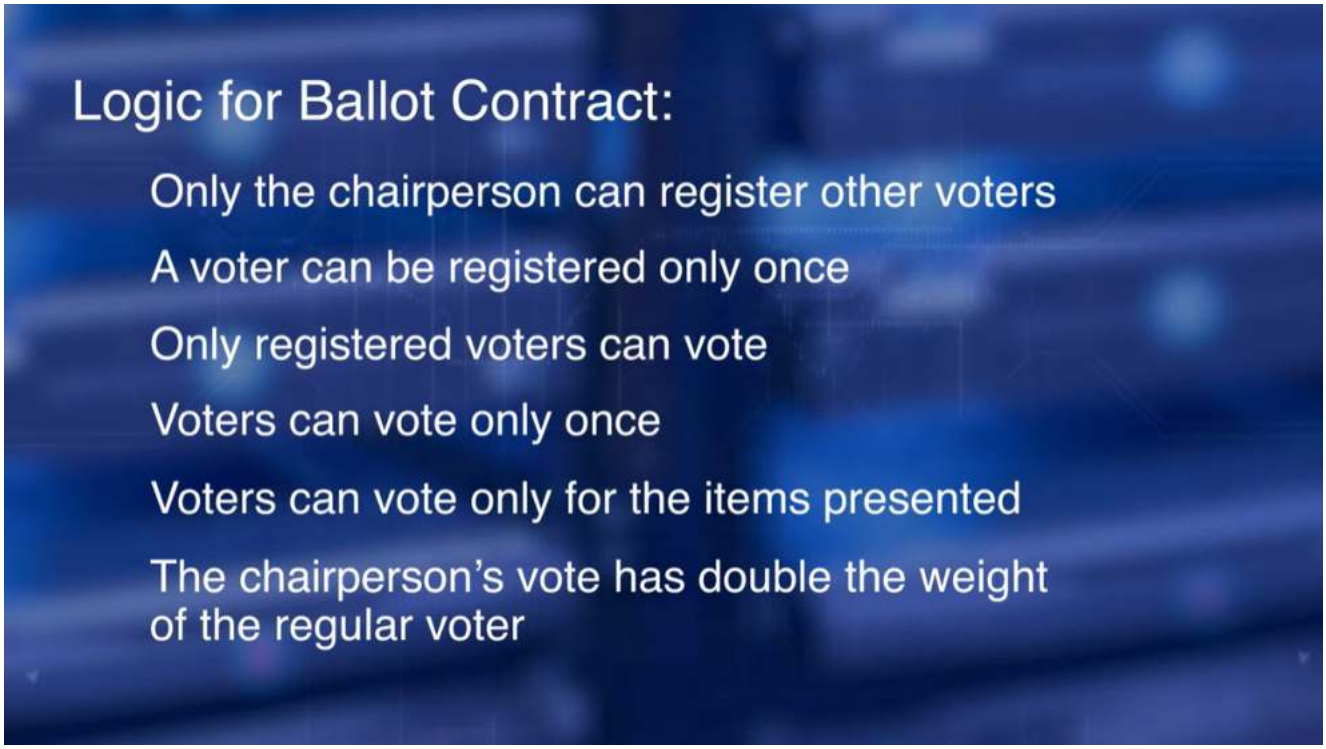
Chapter 2

Truffle Development

2.1 Truffle IDE

2.1.1 Part 1

In the last module we talked about foundational concepts of Dapp. Now, we are ready to develop an end to end decentralized application. Truffle is a one stop IDE that provides everything from an initial application template to a local blockchain for testing the completed Dapp. Similar to how we use remix IDE for testing a smart contract, we'll use the Truffle IDE environment for assembling various components of a Dapp. Upon completion of this module you will be able to work with Truffle IDE for Dapp development, explore the Dapp development process by creating a smart contract and an end to end Dapp. Illustrate test-driven development that is especially critical for smart contracts. Additionally, you will be able to interact on the front-end of the blockchain server through a local blockchain and Injected Web3 environment Metamask.

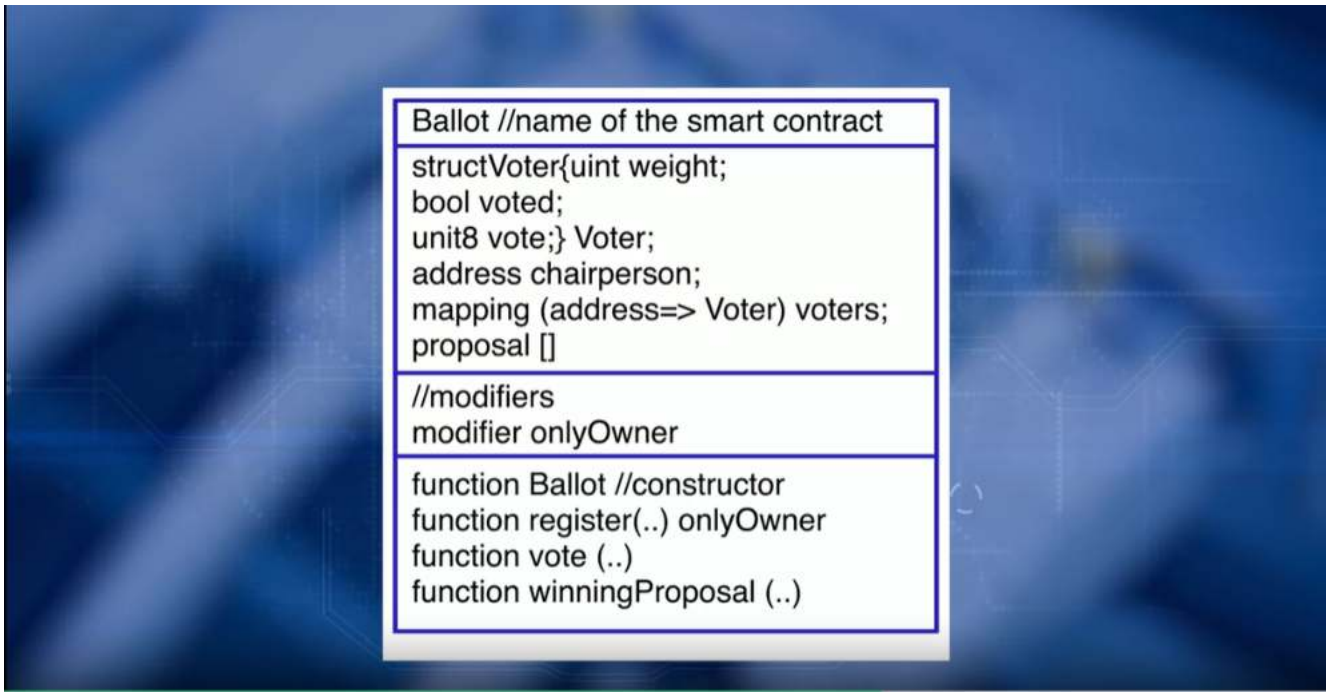


For ease of understanding the Dapp development process we will reuse the problems already discussed in the smart contracts course of the specialization course two. These smart contracts are also available in the solidity documentation. On completion of this lesson you will be able to present a Dapp problem definition, outline an

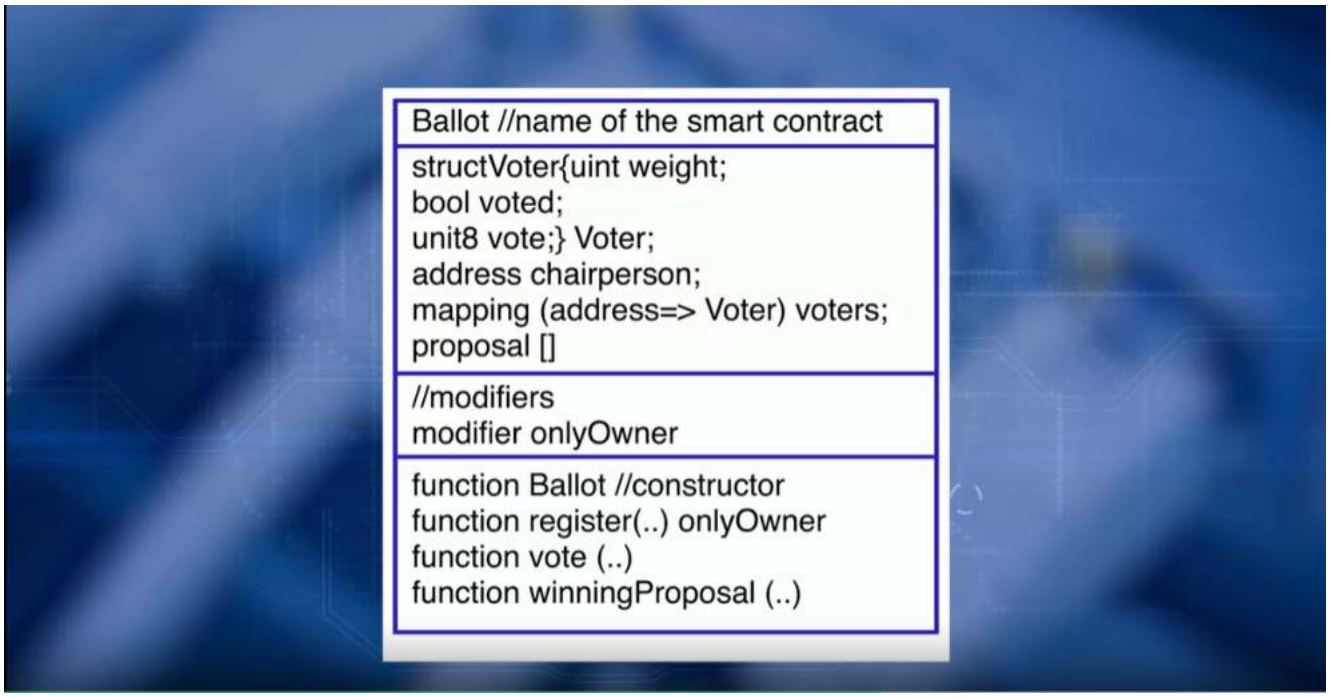
approach to solve the problem, design a high level solution using smart contracts, and identify components of the Truffle IDE. We have chosen to discuss the problem specified in the smart contract, voting dot sol or ballot dot sol of the solidity documentation. It represents the universal problem of choosing a winner or a leader using the democratic process of voting. It represents a wide variety of scenarios from reviews of a product and it's purchase based on the good reviews it received. Also,

for passing a law on a budget in a governing body. Let's start with the review of the ballot Dapp. Our goal is to design and develop a Dapp for voting that selects among number of items such as a set of proposals or products. In our case, we'll opt for the popularity of four different fruits. As a reminder from our previous lessons, the chairperson organizes the voting. Voters are identified by their account addresses.

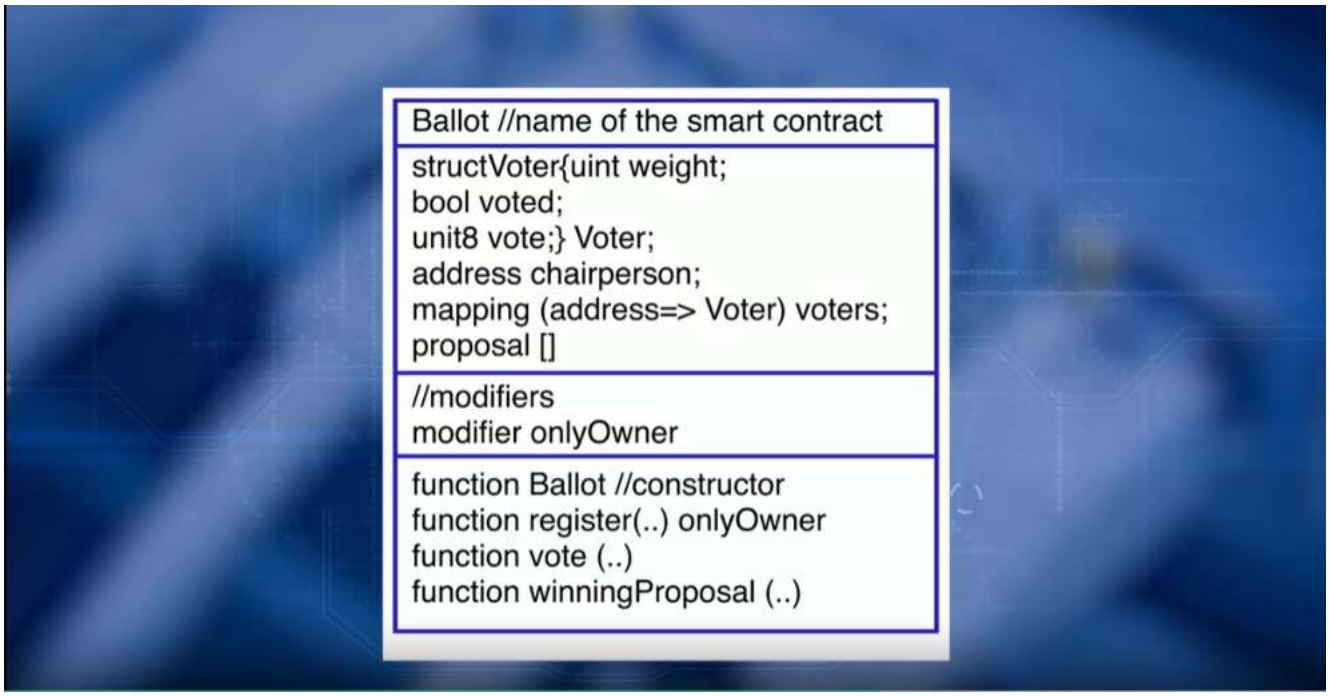
Here is the logic or the rules to implement. Only the chairperson can register other voters. A voter can be registered only once. Only registered voters can vote. Voters can vote only once. There is no "write in" that is voters can vote only for the items presented. And the chairperson's vote has double the weightage of the regular voter. To maintain the focus solely on the process of Dapp development we'll not handle the time elements or stages of the balloting. At this time we'll talk about the four steps that make up the design process. We'll represent the main logic of the problem using the smart contract and solidity. We'll begin with data and operations. To this we'll add modifiers and validation checks to make sure rules specified in the problem definition are addressed. We'll design a test smart contract that will ensure all the stated requirements are met. This like unit testing. After the successful completion of unit testing we'll add the front-end component and test the completed Dapp. This is like integration testing.



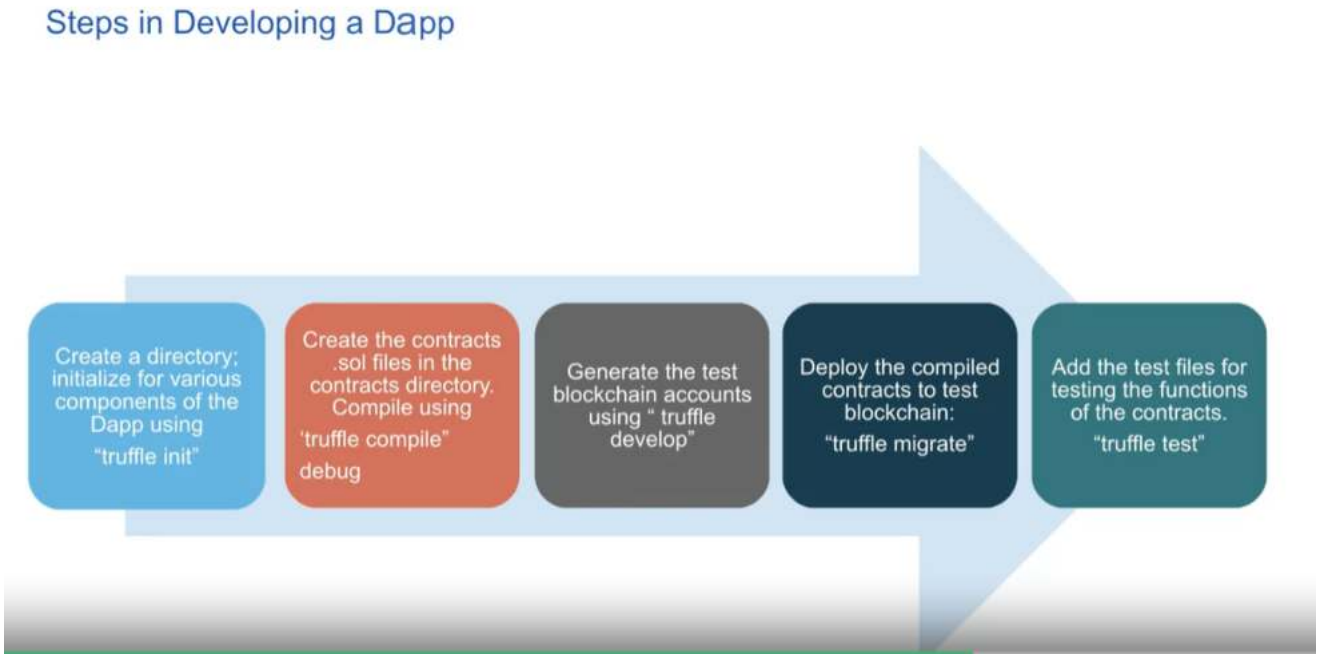
Applying the general design process just discuss to the ballot App requires five steps. Step one, we'll design the ballot dot sol. Step two, we'll then illustrate the modifiers with just one modifier, only owner referring to the qualified person as the chairperson. Recall that in the design of a smart contract you can use modifiers to



represent global rules. Step three, we'll add a tester code for the required problem specification and run the test to make sure they all pass. Step four, we'll add the user interface component. And step five, we'll test the complete application by interacting with the interface. Remember, design before you code. Here is a family of ballot smart contract design. Please note the addition of a compartment for modifiers.



This compartment is indeed very important. Smart contract design now has four components. The smart contract name, data, modifiers, and functions. Recall that modifiers represent the common rules and policies and constraints. Now that we have discussed the problem, how do we solve or import a solution to the problem into the Truffle IDE? Truffle is an integrated development environment or IDE that provides a range of features for Dapp development including commands for initializing a template or base directory structure for a Dapp, truffle in it. Smart contract compilation, truffle compile. Local test blockchain for testing with a console, truffle develop.



Migration scripts for deploying the smart contract, truffle migrate. A test environment for testing the deployed contract, truffle test. These are just a few operations possibly with truffle. These are sufficient develop

and tested Dapp. After you complete the course you can explore other advanced features truffle offers. Next, we need to install a few software packages including Node.js which will serve as your web server for the Dapp front-end. Truffle 4.0.4 and Metamask 3.14.1 plugin for Chrome. You may have to add the Metamask plugin from Metamask IO to your Chrome browser. It will link to the blockchain created by truffle to manage the accounts acting as a bridge between the application front-end and the blockchain node that hosts the accounts. To save time and effort we've already installed Node.js and truffle IDE in the virtual machine image we provided in the blockchain basic scores.

You used it to complete your project in course one. The virtual machine you downloaded has the required software pre-installed. This pre-installation and VM ensures that we are all in the same versions. This is especially important since software requires updates frequently. This quotient's synchronized to the particular software quotients. Before you start the development, please download a copy, all the files and sub-files needed from the course resource section. Also, it's good to refresh common Linux commands such as cd, ls, mv, make directory, and so on. We have provided a Linux command sheet to help you with this. You may also refer to ballot one demo in the last course.

2.1.2 Reading

Truffle v4.0.4 which is installed on the VM is outdated and will provide this error if you proceed with the instructions without performing the updates:

Steps to update Truffle: 1. Run sudo npm install -g truffle@4.1.15 (You will need to provide the system password which is ubuntu).

2. Run sudo cp ./.npm-global/bin/truffle /usr/bin/truffle

3. Truffle should be updated to v4.1.15. You can check the version by typing truffle version

For some reason, doing Step 1 alone installs Truffle to a different directory than the one for normal commands, so we copy it in Step 2 to the Environment Variables folder (not necessary to know for this course).

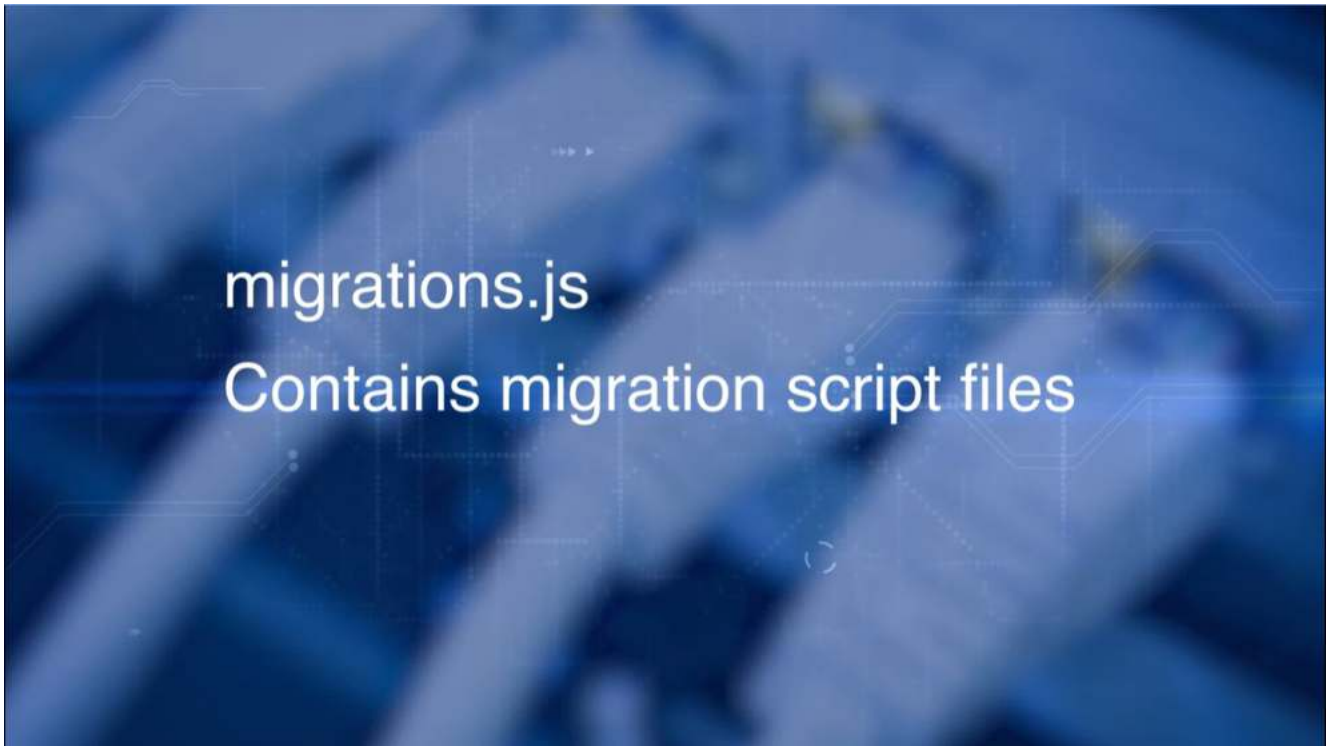
2.1.3 Part 2

Next we are going to create a folder Ballot1 since we are going to work with several versions of the ballot. Truffle provides a template directory of folder with a required structure. Know that you'll have to run the virtual machine and start a terminal before you issue the commands in a command line. First, create a folder, Ballot1, for your project. Navigate to your workspace folder.

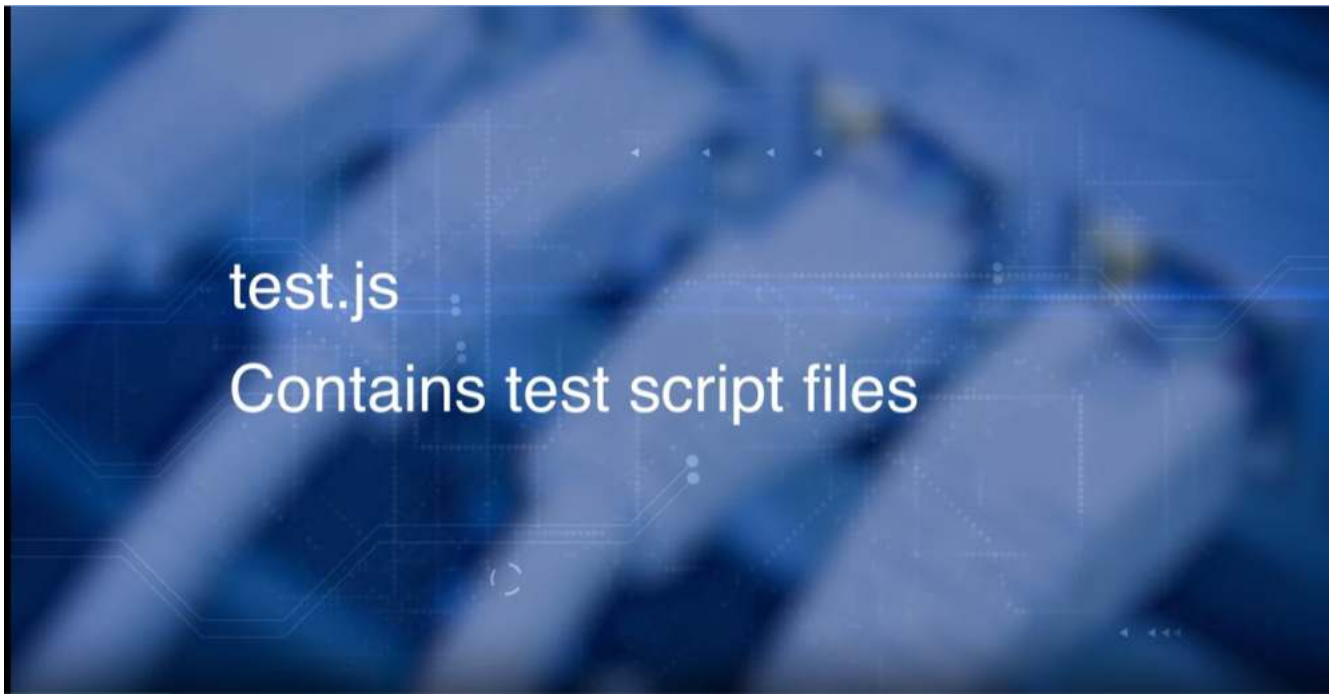
In this folder, you will create a directory or folder called Ballot1 by mkdir Ballot1. Then navigate into this ballot directory by cd ballot1. Once you're inside, you're ready to initialize a project. You can do that by using truffle init command. Once truffle init executes, you will see three directories of folders, contracts, migrations, and test. And contracts contains solidity source files for our smart contracts. There is an important contract here called migration.sol, that is the smart contract for facilitating deployment.

Migrations folder, Truffle uses a migration system to handle contract deployments. A migration is additional special small contract that keeps track of changes.20. Test directory contains both JavaScript and solidity tests for our smart contract. Next you see truffle.js, the truffle configuration file, that specify the blockchain network

ID, IP, and RPC port. We'll now discuss adding contents to the directories that we created and configuring truffle.js environment for the blockchain properties. Okay, let's add ballot.sol to the contracts directory. Review the familiar content of the contract and its functions. Now, change directory back to the root directory of the ballot, and compile using truffle compile.



No errors should be present. Hello everyone, let's get started with the demos for course three. We've given you a virtual machine imagine, make sure that you have installed that. And also, you have installed Truffle if you are not using that IDE you've installed. Truffle is the IDE we are going to use in the development of the decentralized app that we are going to create. We are going to create several apps. Also, make sure that you download the code files, the folders and the code snippets, and other file that we have provided in the sources section of course three.



I've created a workspace here for coursera and course3. It doesn't have anything at this point, as you can see, and I'm going to keep adding to this folder as we develop several demos. To start with, let's start initializing the template that is needed for the first demo. This demo is about the ballot that we've been using in the second course, course two, and also right here in course three. And I'm going to do that by issuing the command traffic init. That'll create a template for Dapp.

It's ready. We can see that by looking at the folder and I have contracts, migration, test, and a couple of script files that I have for configuration. And they may not have anything at this point, so let me just go through

and look at the contracts. Contracts folder has one smart contract, as you can see, it's in solidity, that helps in the migration of the smart contracts that we write. And so that's why it's called migration smart contract. Let's go back, and in the migrations, there is a script file that helps in the deployment of the migration.sol that we talked about just now. And let me see, let me go back to full screen here. And we have other files test where there's nothing at this point, but we can add our test script in here. So this is the basic template into which that you are we are going to bring in our Ballot.sol. How do you do it? I'm in the right directory, this is the base directory. And I'm going to copy from our CourseraDocs, that's where I copied all the files from the resources. I'm interested in a smart contract for Ballot, and I'm going to copy that into my contracts folder It's copied right, so let me just go to my contracts folder, make sure it's there. It is there, okay. So what do I do, I go back to my base and now I'm ready to compile. How do I compile? I just use truffle. [INAUDIBLE] do that, it goes through. It compiles the two smart contracts that are in the contracts folder and it creates the build artifacts. And as you can see, it has created a new folder build, and that's where the build artifacts are that the compiler generated. Now let's see whether if we write a smart contract from the scratch and if we make some syntax errors if Truffle compile is able to find out. For that I'm going to use an editor. You can use any editor, Atom, Sublime, but I'm going to use the editor that comes with this, Ubuntu. And I'm going to do that and I'm going to go into, Contracts, and here, and I'm going to g, gedit Ballot.sol,. And I'm putting it in the background so that I can be doing something else when the editor is open. And keep it open as I'm doing the other items in the directory. So I'm going to go here to the register function, this is a familiar smart contract for you. I'm going to remove that semicolon just to introduce an error in this smart contract. You see that it is in the third line, one, two, three, of the register. And you will see that this is the next line and that's going to be creating the syntax error. And I'm going to save it, and let's go back to Ubuntu, and I'm going to go back and just check where I am. I'm going to go back to my base and I'm going to initiate my truffle compile again. I'm going to initiate my truffle compile, And it's going to give me an error. And it says that, expected token semicolon, that's what we removed. And it says before this line there was semicolon expected that was not there. So let's go back to my gedit and correct the error that we introduced here. Put the semicolon back, save it. Come back here, and I can use an up arrow to go back to the previous command and I truffle compile. And now I shouldn't get any errors. So we compiled all right. This is how you compile a smart contract in a Truffle IDE. You can also try editing the ballot also. Remove a semicolon somewhere and repeat the compile process. You will observe that truffle flag syntax errors. You can go back and debug it, save it, and recompile it using truffle compile. A better practice would be to remove any errors using a familiar Remix environment, introduced in the smart contracts course. Recall that Remix has just in time compiler. That compiler can catch syntax errors as you type in the smart contract code. Now we are ready to configure the truffle environment. To do this, replace the empty truffle.js file in the ballot home directly to the one shown here. That sets the configuration of the local blockchain you will deploy next. 9545 is the RPC port for the test chain provided by Truffle. Now we are ready to deploy the local blockchain with Truffle. In a new terminal, navigate to the home of the ballot and deploy the development blockchain. The command is truffle develop. This command will deploy a local test blockchain with ten account addresses that you would see. It'll also display seed words while linking these accounts into a wallet or an interface such as MetaMask that we'll use later. Save the words somewhere, you may also copy the saved words into a file. Just one more additional step. We need to add a file to the migrations directory to deploy our smart contract. Create a new file named $2_deploy_contracts.js$in the migrations directory. Add the following content to the $2_deploy_contracts.js$ file. Here is the content. You can simply copy $2_deploy_contracts.js$ from the Resources for the course into the migrations folder or directory. Are you ready to deploy the smart contract?

2.1.4 Part 3

Once again from the base directory of the ballot, issue the command Truffle, migrate, dash dash, recent. On execution you will see Saving successful migrations to network with a valid contract name. Also, you will see the message Saving artifacts. These are indeed artifacts that were generated by the compile process we examined in Course two. Dash dash, reset is to ensure that the new version of the smart contract is deployed, resetting the older version that you may have on the block chain. ¿¿ You have successfully compiled the ballot that's sold our smart card, right? Using Truffle ID. Now we are ready to deploy it or migrate it onto a test chain. The test chain has to be deployed first before we can deploy the smart contract on it. So let's do that. In order to do that, just observe in our folder, we have a truffle.js that is the configuration for our test chain. There's nothing. So in order to get something into this, we're going to copy from our Coursera Docs. truffle.js we have readily provided you a configuration card. And let's look at it now. I copied it, and let's look at it now. It's got some details about what the host IP is and what the port number through which its going to connect to the web application and the

network ID itself. So now let's, I'll clear that, and let's deploy this test chain. In order to do that I'm going to go into another term, a new term, and then I'm going to go into the same folder that, where I have my truffle.js. And you can see that's my base directory, and here I'm going to do truffle development. And that will deploy a test chain for me with 10 accounts, addresses as you can see account 0 to account 9 and also the private keys for those private keys are 256 bits. And the addresses are, we know from our course one, that are 160 bits. It also gives you seed words. Make sure that you copy and keep it, at some point when we are doing the testing, we may need this for connecting to this test chain. So our chain is ready, now we are ready to migrate or deploy our smart compile smart contract into this. In order to deploy, we need one more file, and I'm going to copy that file to, Set our docs again to deploy. This is one, this is a file that goes into Migrations folder, and it will help us deploy our contracts into it. So let's see what is in this file. And you can see that it's got, it says that you, the files to be deployed, the smart contract to be deployed is valid. And now let's go back. And we are now ready to deploy. So how do you deploy the smart contract? By using truffle? Of course migrate. And when I do that I won't use minus minus reset every time. Remember, this is development phase so we can overwrite the previous smart contract if you're not satisfied with the previous version of smart contract that you deployed. This is only for development. Once you go to production, once you deploy the smart contract you cannot change it. Here I'm resetting in any previous ballot that I might have deployed. So truffle migrate minus minus reset, dash dash reset. That will deploy the compiled smart contract. You can see that this uses the network development that we created just now. And it deploys the initial migration that is for the base migration, and then it deploys the valid contract that we compile. And it also saving the artifact that we'll use later on as we connect it to the. ¿¿ Congratulations, you have deployed your ballot smart contract on the test blockchain that Truffle provided. The truffle develop command that you executed earlier on a different terminal opened up a console for you. What can you do with the Truffle console? The console provides a command line interface to the accounts created on the test chain and also to the management APIs we discussed earlier in Module 1. Here are a few commands to transfer ethers between accounts, illustrating the power of command line. We are also testing the Truffle test chain by sending a transaction from one account to another and checking the balance of the sender. In summary, in this lesson we introduce the ballot problem to illustrate the Dapp development process on Truffle. We introduced the Truffle IDE commands. Remember, Truffle is already pre-installed on the VM from course one. We also discussed the directory structure for DApp developed on Truffle IDE. We concluded with of the development and deployment of a smart contract. In the next lesson, we learn to test this marker correctly.

2.1.5 Resources

TRUFFLE TUTORIAL INDEX

2.1.6 Quiz

1. What are the steps in Dapp development process ?

- ◯ Prototype smart contract on remix, develop and test in truffle, deploy
- ◯ Analyze problem, prototype smart contract on remix, develop and test in truffle, deploy
- ◯ Analyze problem, prototype smart contract on remix, develop and test in truffle
- ◯ Analyze problem, develop and test in truffle, deploy

2. What does truffle init do?

- ◯ Creates a base directory and initializes templates with base Dapp code
- ◯ Connects to node running in your local
- ◯ Starts the truffle network for you
- ◯ Downloads a sample Dapp

3. What is the RPC port for test chain deployed by “truffle develop”?

- ◯ 8544
- ◯ 7545
- ◯ 8545
- ◯ 9545

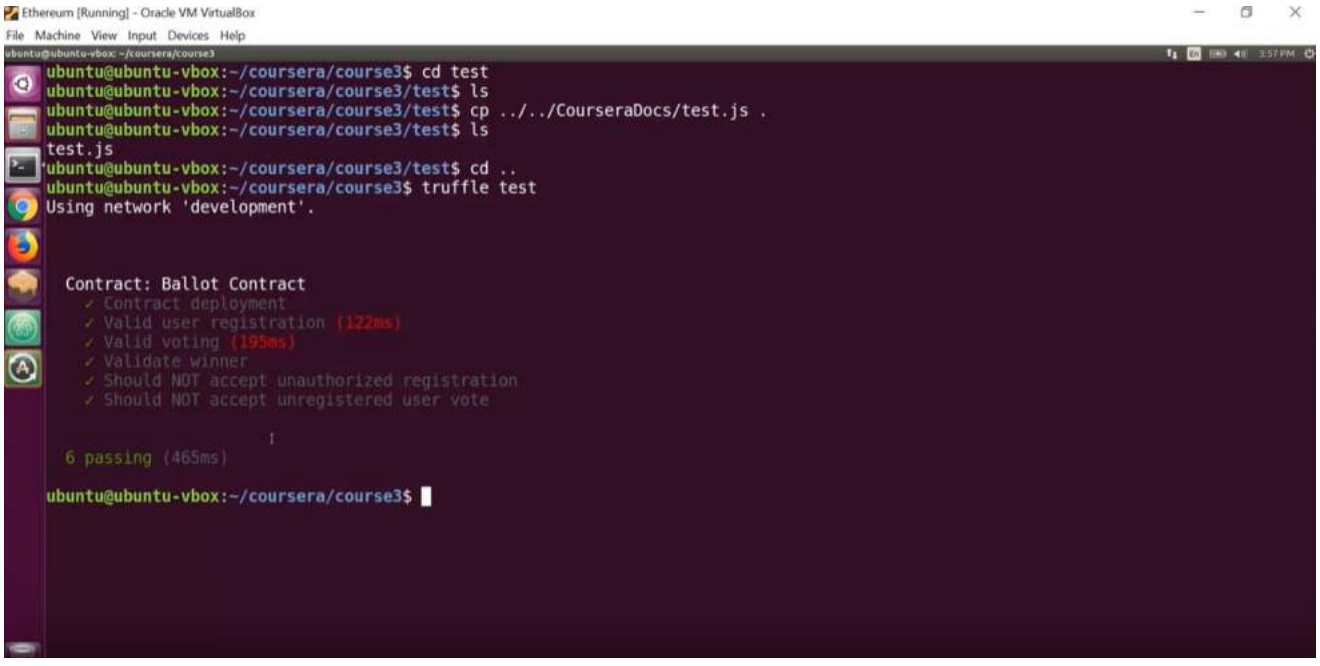
4. What does the option --reset in truffle migrate do?

- ◯ Clears the intermediate transactions
- ◯ Run all migrations from beginning
- ◯ Start execution from the last but one migration
- ◯ Runs the latest migration

2.2 Test-Driven Development

2.2.1 Part 1

Some of us who grew up with emergence of integrated chips and system on a chip, we know the importance of testing. Testing is an essential phase of hardware development. Once a chip is mass-produced, it's impossible to go and fix bugs, the design is hard-coded. To illustrate this, look up the Intel floating-point bug of 1994 that resulted in a massive recall of its Pentium chip. Also review the current Spectre and Meltdown bugs in the hardware chips that may result in security issues. Can you believe these bugs may require hardware redesign and replacement for almost all computing devices in the world using these chips? You can see why testing is a critical step to hardware design before a product is released. On the other hand, bug fixes on software systems are provided through weekly updates, as a routine. How about our smart contracts, the core logic of our Dapps? Smart contracts are like a hardware chip. Once deployed, they're final and cannot be updated, unless special provisions or escape hatches are built in. Recently, we heard of million-dollar heist in DAO hack and Parity wallet issues. These are indeed due to bugs in smart contract code. So testing is absolutely imperative for smart contracts. Upon completion of this lesson, you will be able to explain the importance of a test-driven development and test scripts in detail. Testing with ballot smart contract can be done by positive tests, making sure for a given valid input, it performs as expected. We'll test the complete ballot cycle of deploy, register, vote, winning proposal. And negative test, making sure it handles invalid inputs and situations appropriately. We'll code only two of the many negative tests possible. Usually, testers are written in the same language as the main application to be tested. You will write the tester itself as a smart contract and in Solidity language. There is a fine illustration of this in Truffle Pet Shop example. However, ballot contract uses address type data repeatedly for the chairperson and the voters. This causes a problem when another smart contract is used as a tester. So we'll use alternative language supported by Truffle, namely JavaScript, for writing our test. Truffle allows both languages, JavaScript and Solidity, we'll use JavaScript. Look at test.js, it has four positive tests and two negative tests. If we can open test.js using an editor, you can follow along.

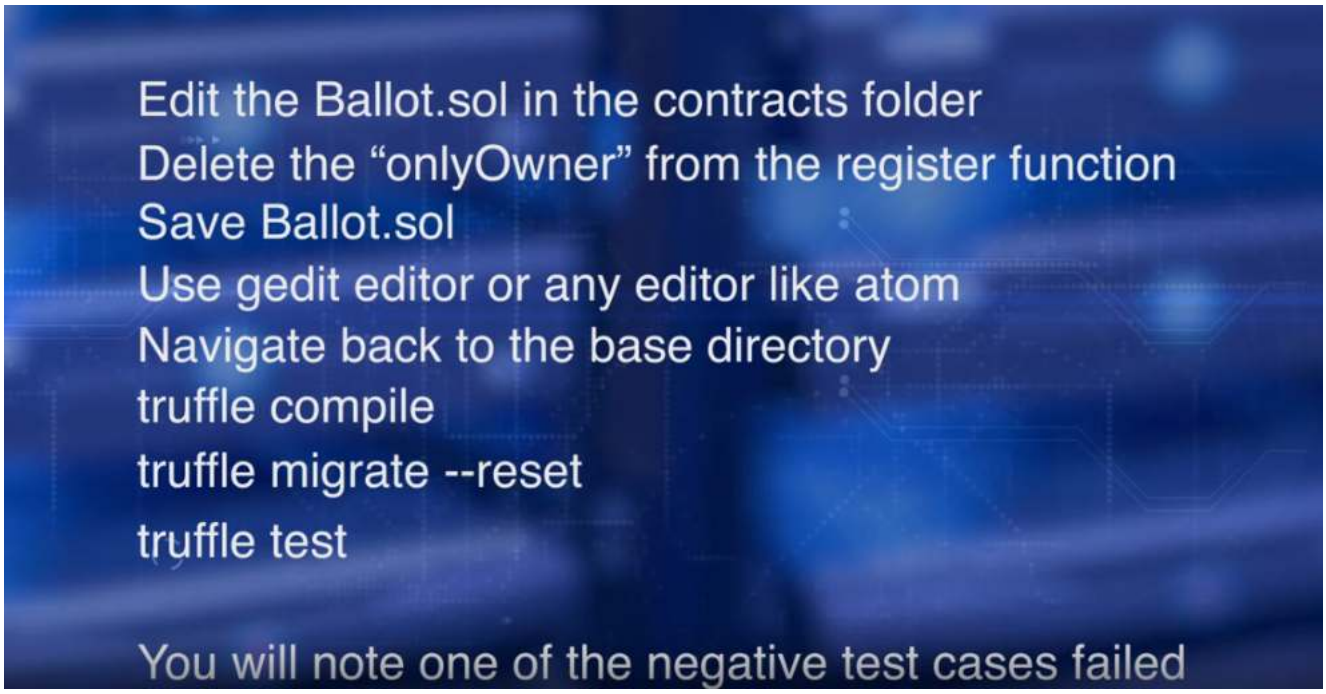


So far, we've used Truffle ID for initializing a template for Dapp development, populated this template with ballot.sol smart contract into the contracts folder. Compiled this smart contract and deployed it or migrated it to the test chain that Truffle ID provides. Now let's see how we can test it. In order to do that, we need to provide some test scripts in the test folder, we don't have anything there, as you can see. And luckily for us, we are provided the test script in the course resources. Let's copy this into the test folder. As you can see, the test script that we are provided is in JavaScript. You can also write test cases in Solidity, you have good examples of that in the literature. Let's see what test.js contains. I have it already open in the Atom editor. And the

way we have designed it is that we have set up positive test cases and then couple of negative test cases. Later on, we can add any number of negative test cases. You can see that first line says that we need the Ballot.sol artifact. Then instant test instance of this ballot is created, and we make sure that it is deployed in the first test case. In the second test case, we want to make sure that good user registrations are going through. And we are registering three accounts and making sure that only the owner can register. So you can see that it's valid registrations are going through. And the third function is valid voting is going through. The fourth function is choosing the right winner. So the first four cases are completely testing the pipeline or the process of deploying, registering, voting, and winning functions. And the next two test cases are negative test cases where not the owner is trying to register an account, and an unregistered voter is trying to vote for that processes. So there are two test cases, there could be many more. And as you will see in the later demo, you will see many more test cases. This is going to be the template for your future applications testing that you may do. Okay, you need some basic JavaScript knowledge to do it. Let's go back here, and I have this available here, and I'm going to go back to the base directory. And of course, the command is truffle test. It compiled it, so we're just going to double test it. You can see that it through the test cases. And there are four test cases that are valid, we gave good inputs, and it did check it. And the next two are bad inputs, but it did indeed capture that and say that your Solidity smart contract did indeed capture these as bad inputs and did not execute the operation. So we passed the test, six tests passed. So this is the end of testing. We're going to add more testing in a future demo on ballot. On to the scenarios where the test failed.

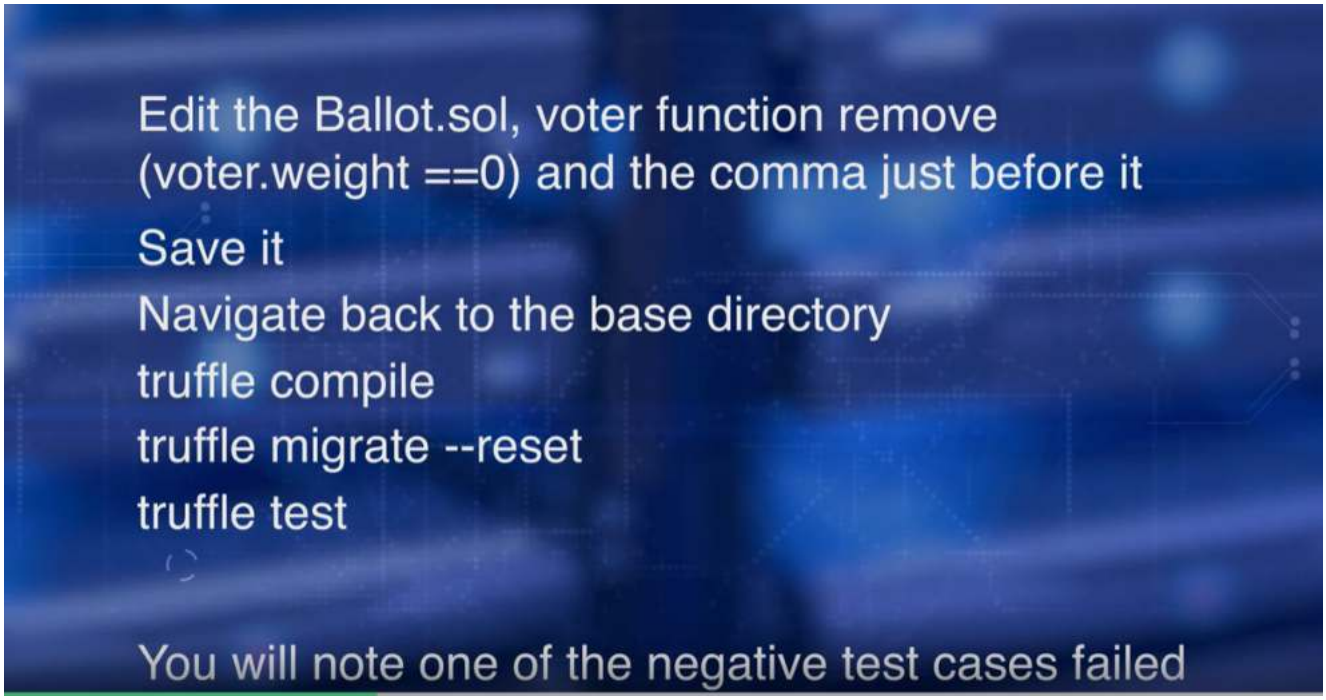
2.2.2 Part 2

Now edit the Ballot.sol in the contracts folder. Delete the onlyOwner from the register function. Save the Ballot.sol. You can use gedit editor or any other editor, like atom, that is familiar to you. Navigate back to the base directory and issue the commands, truffle compile, truffle migrate –reset. In this instance, reset is very important, because we are overriding the previous version of the smart contract and reset takes care of that. Then on to test, truffle test.

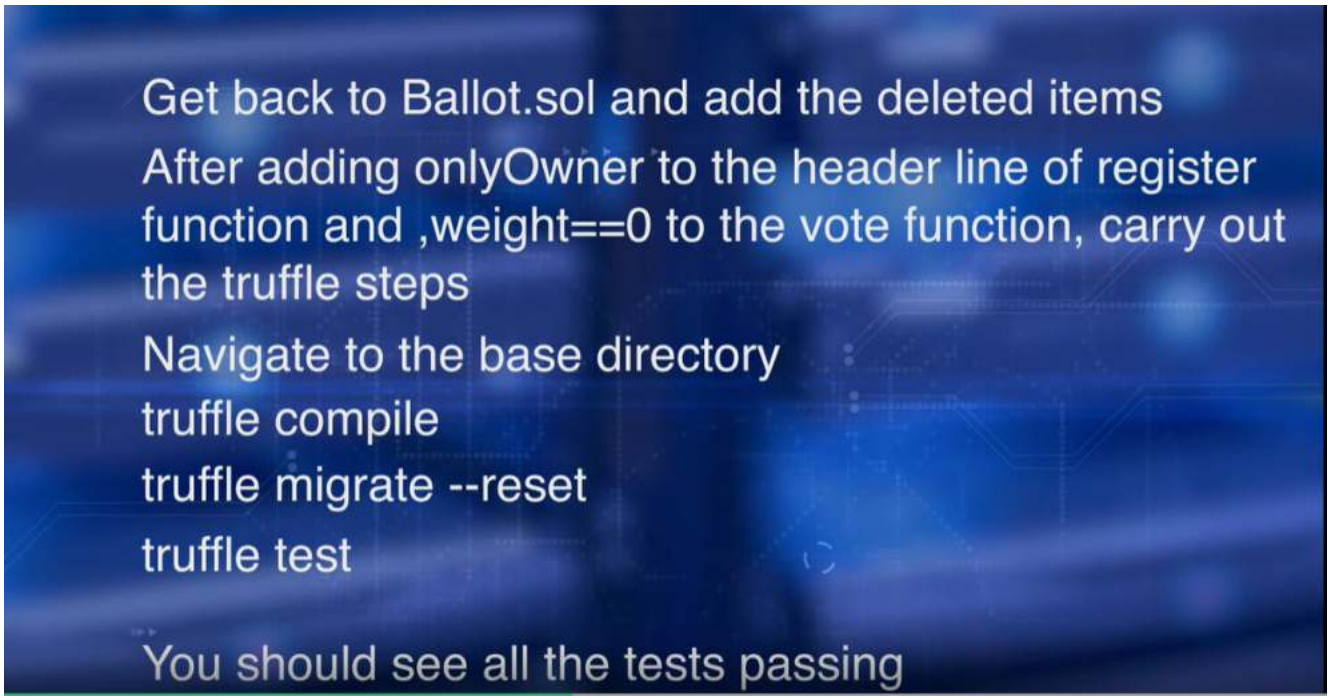


You will note one of the negative test cases failed for the second negative test. Edit the Ballot.sol, voter function. Remove (voter.weight == 0) and the comma just before it so that it does not check for unregistered voter. This is an error. Save it. Navigate back to the base directory and repeat the process again, truffle compile, truffle migrate, –reset, truffle test. You will see one more of the negative test cases failed. Now get back to Ballot.sol, edit it, add the deleted items. After adding the onlyOwner to the header line of the register function, and ,weight==0 to the vote function, save it, and carry out the truffle steps to make sure everything tests okay.

Navigate back to the base directory, truffle compile, truffle migrate, –reset, truffle test. You should see all the tests passing.



Let's now check for the negative tests. There are two negative tests. One, registration by the owner. Only owner can register other orders. And the second one is for accounts registered only can vote. We'll test only one of them now, the second one we'll test later when we have the complete tier. For the first one, I'm going to go into the contracts folder and edit the gedit, my Ballot.sol and go to the register function. In the register function, there is a modifier specified here, only owner. Only owner can do registrations and I'm just going to remove it. This is an accident delete, not specifying a condition that is required for a problem and this will compile, all right? There should be no problem. Okay, so if I do compile. I do compile. If I do truffle compile, it does compile, all right?



All right, so no compilators. But there is an error in the specification of the problem. It did not validate the condition that only owner can register other voters. So let's see how we can do it. I'm just going to do truffle test, because we've written the test case for it. I'm going to test it and voila, you can see the positive test passed because the process was correct. But when it came to the negative test, since we did not specify only owner, and that condition was checked by our negative test case, and it gave us an error. And so, it captured that we did not enforce that condition that registration can be done only by the owner. And that's how you enforce testing of the smart contracts. Can you see how easy the development and testing process is with truffle? Truffle compile, truffle migrate, truffle test, and some edits in between. By now, you should be pretty familiar with the steps to develop a DApp. You can always use the test.js that we have provided as a model for writing your test, even if you're not an expert in JavaScript.

2.2.3 Resources

Ethereum Pet Shop Dapp
Testing of Smart Contracts in the Blockchain world
Smart Contracts and Truffle 101. Part 4 - Testing Functions and Errors

2.2.4 Quiz

1. What is a positive test case in test-driven development? 1 point

- You provide valid values for input and test passes if the system behavior is correct and valid for the given input
- You provide valid values for input and test passes if the system behavior is incorrect and invalid for the given input
- You provide invalid values for input and test passes if the system behavior is incorrect and invalid
- You provide invalid values for input and test passes if the system behavior is correct and valid

2. What is the command for running test scripts ?

- truffle test
- truffle test.js
- test truffle
- test test.js

3. What is a negative test case in test-driven development? 1 point

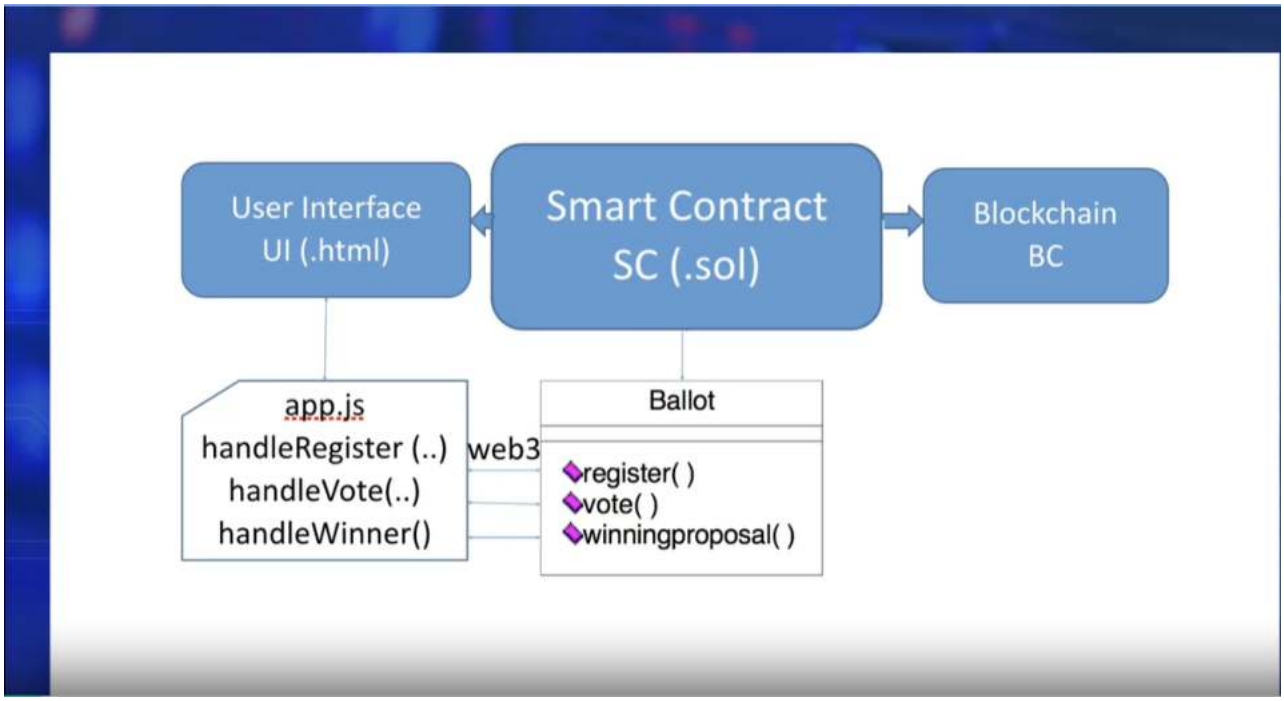
- ○ You provide invalid values for input and test passes if the system catches and flags the invalid inputs
- ○ You provide invalid values for input and test fails
- ○ You provide valid values for input and test passes if the system behavior is correct and valid for the given input
- ○ You provide valid values for input and test passes if the system behavior is incorrect and invalid for the given input

2.3 Web Interface and Testing

2.3.1 Part 1

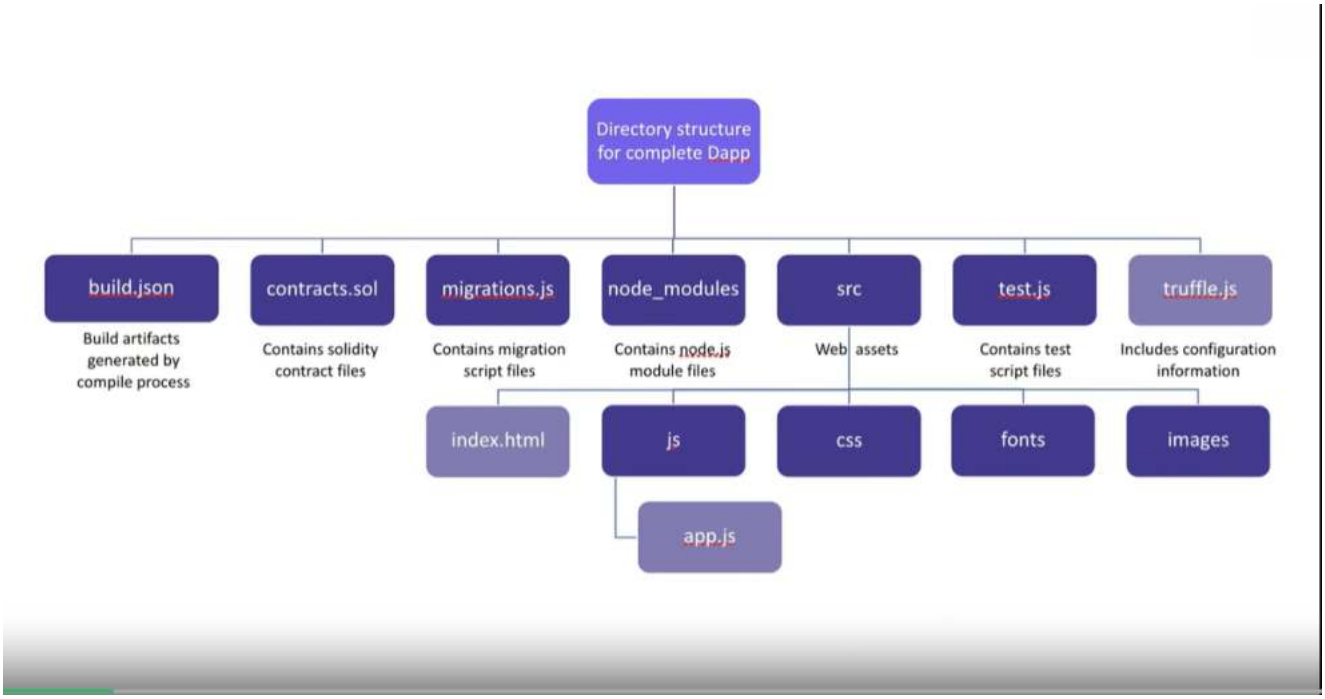
In this lesson, we are going to add the web interface to the DApp. In order to follow along, we have created a pre-filled ballot DApp. You can copy this into your workspace and work along. There is a zip file in the resources called Ballot2.zip. Upon completion of this lesson, you will be able to list the changes to be made to the base code of the Ballot.sol to link to the user interface, and to integrate the user interface front-end to the ballot smart contract, and the blockchain backend.

In order to accomplish this, copy Ballot2.zip into your workspace. Unzip the Ballot2.zip. Note the directory structure that appears. This may be the base structure needed for any DApp project. Note there are few more files and directories than previously outline.



We know that contracts has solidity contracts. Migration folder has migration scripts. Test has the test scripts. Build has the JSON artifacts generated by the compile process. Source folder has the web assets, such as JavaScript, CSS, and index HTML. Node module folder has the node.js modules. And JSON files and JavaScript are the configuration files. Now, navigate to the source directory that provides the web assets. There is an file called app.js in the source Java's directory. This instantiates the objects needed for the web client to communicate with the blockchain node. We need to expose the blockchain accounts created through web3 provider object of

the app. Simply put, web3 init code does that. It shows how web3 object is created and linked to the test chain we've applied.



Secondly, we need to tell the web3 where to find the smart contract artifacts. Init contract method of the app.js does this. Then, you'll have to add methods to handle the calls to the smart contract, and return values from the smart contract to the web interface. So far we have created the backend of the DApp. We've initialized using truffle in it, compile, deployed, and tested it. Now let's see how we can add the front-end. And so, going through the init process, which is beyond the block ten protocol, we've given you the complete DApp in Ballot2.zip in your resources. Copy that into your current folder and let's explore. And I'm going to do that Ballot2.zip from your resources into this folder, and we have it here. Now I'm going to unzip it. And it's going to create a whole lot of files, and I'm just going to clear. And let's see, we got a nice little directory Ballet2 going there.

And we can see that, it's got a lot more folders than we had before, but there are quite a few familiar ones, contracts, migration, build, and source, and test. And source is what holds all the web assets for your application, the front-end, and also the JSON files that you have. The node modules hold the server end of it. The node JS's our little lightweight server that's going to solve the HTTP request. And so, let's go into the source. We're going to look at only one of the files that's going to act as a glue. And one particular one is JavaScript file and that is app.js, and we already have it open in Adam. And I just want to point to a couple of functions there, as you will later on use this template to develop your front-end. I'm creating an app object that initializes the web3 object that is going to be the receiver of all the blockchain reports from your front-end application. And it's going to pass it on to your underlying RPC endpoint that you have opened up in your blockchain.

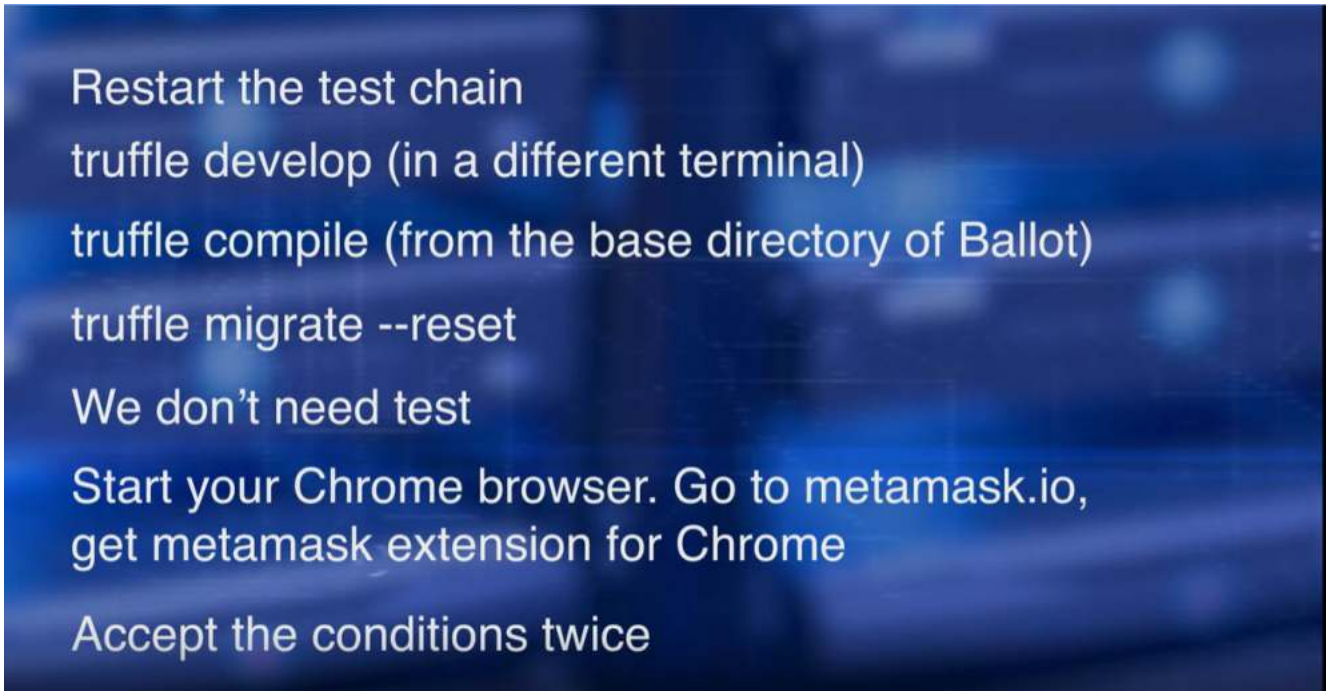
And then, there is also a init contract, that is init a web3 object. And that init contract, which is going to point out to all the artifacts that is needed for your smart contract, in this case Ballot.json. Beisdes that, we have other support functions. The one that I want to point out are, handleRegister, handles the register function of your smart contact. handleVote, that's a glue for your vote function of your smart contract. And handleWinner, that is a Function that handles the winningProposals of your smart contract. And so, you can use this as a template for developing your front-end. And this is an important file that acts as a the pipeline between your underlying smart contract and the front-end.

Now let's go back, and I'm going to go to the base folder, if you come too far inside. And this is my base folder, because I see contracts and build in it. I'm just going to be compiling this. I can compile. I can deploy. I can do all that in one command truffle test. But anyway, but I'll go through the process of compiling. And next test, truffle. Let's test it, okay? And we've added a few more test cases. And you can see all the test cases pass, ten test cases. You can go and explore test.js to understand more about the test cases. And then, I'm going to do truffle migrate. This is very, very important you say a reason, because we did test a couple of ballots before this. And so, we want to overwrite this with a newer version of it. And so, I'm going to be deploying a newer version. I don't know whether we made any changes, but still. Anyway, we've deployed a newer version of it. And finally, I'll have to run lite server. And I'm going to do that. This is for the webpage that we're going to mpm run development server, run dev, and that will open up a web end point, which is going to display all our web assets. We have three handle functions corresponding to the three functions of the ballot smart contract, function handleRegister(), function handleVote() and function handleWinner(). There are other supporter utility methods of functions that are beyond the blockchain protocol. These are the user interface designer's responsibility. However, we provide these functions for completion. Please review these. Let's navigate back to the valid base directory. In order to create a web interface, minimally you'll need an index.html file. We have also added other web assets, such as the images for the items we are voting on, funds, and CSS style file. At this time, you have at least two choices. If you have the front-end development skill, you can create your own front-end, or just use the web assets given in the source directory.

2.3.2 Part 2

Now you're ready to go through the development process once again. Restart the test chain. Truffle develop in a different terminal. Truffle compile from the base directory of the Ballot. Truffle migrate –reset. We don't need the test command, we are done with it. Now, let's go to the browser. Start the Chrome browser. Go to

the metamask.io., get the metamask extension for Chrome. Accept the conditions twice. You will see a brown MetaMask face installed as a plugin into the Chrome.



MetaMask is a bridge between our blockchain server and the web. We've added it as a Chrome extension. This is the preferred approach for our explorations. MetaMask is like a digital wallet that provides features to connect to the underlying blockchain node, and manage accounts, guess points for transactions, and the balances. MetaMask also provides a simple interface to sign the transaction and transfer guess points needed for the execution of the transactions. MetaMask installs quickly. It is easy to use. Don't hesitate to remove it and add it again in case it goes out of sync with your blockchain server. Now, let's see how we can link MetaMask to the blockchain server. In the truffle IDE default port for the test chain deployed by truffle develop command is 9545. Account addresses are exposed through this port. Let's now link MetaMask to manage it for our DApp. I have the frontend node ready and the MetaMask has all the accounts, and I have here created about four or five accounts, and I'm just going to start off by registering some of the accounts. So, let's see. Register that. Okay, that should go successfully, okay, and then it will show that it's successful. And I'm going to go to the next account. These were the support functions you formed in the app.js that brought about these to the web. Okay, and that was registered successfully, and then I'm going to go into the third account, A2 and register. Remember that we're going from the owner, and three accounts have been registered. And let's run through one more, but this time I'm going to go to account number 2. Account number 2, and try to register.

Let’s see what happens. And you can see there’s something happening here. There’s no, okay, I just say Submit, and okay. I’m just going to increase that, and then Submit, and you will see that registration failed, because it was from account number 2, but only owner can do it, okay. So anyway we’ve registered three of them, and let’s see whether we can vote on all of them. So I’m just going to go back to one. Start voting from Account 1. Account number 1 has got twice them, so let’s see whether we can vote for banana. The owner is voting for the banana. That’s good, and that went successfully. And the rest of them, the three of them outsiders other than the owner are going to vote for some other, maybe apple this time. So we’ll vote for apple, and it should go through fine and successfully, and let’s see we have one more. Account 3, and that’s also apple, and that went through all right. And you can see the guess points and other things are indeed truly reflected and taking care of the MetaMask management, and I’m going to go click on 4. And again, vote for apple. That should go though fine. You can see the guess limit. Everything displayed, and finally I’m just going to go to account number 5, which has not being registered, and see what happens here. It has not been registered, we do have test case, and I’m just going to see whether it can vote. And you can see that there’s no guess limit here, that means that account is unknown, when I submit it’s going to say failed, because we do have a test case holding it making sure that it will not work. Not this case but the solidity contract is written in such a way unregistered cannot vote. So we have voted enough, and let’s see whether we can declare the winner, it should be apple. Yes, the apple is the winner, and that is the complete DApp demo. In summary, we learned to set up a web frontend, and tested a complete DApp, and you observed that Truffle provides a user-friendly environment for developing and testing a DApp. By now, truffle commands should be quite familiar to you. Develop, edit, compile, migrate, test, all with truffle in front of them, and with any parameters you may need. This module illustrated all the steps needed in developing and testing a complete DApp. That’s great. Please make sure you’re able to complete the development and testing before moving on.

2.3.3 Part 3

I have the front-end now ready and the MetaMask has got all the accounts and I have here created about four or five accounts. I’m just going to start off by registering some of the accounts. So, let’s register that. Okay. That should go successfully. Okay? Then it’ll show that it’s successful. I’m going to go to the next account. These were the support functions you found in the App.js that brought about these to the Web, and that was registered successfully. Then, I’m going to go into the third account a two, one and then ”Register”. Remember that we’re going from the owner and three accounts have been registered. Let’s run through one more. But this time, I’m going to go to account number two and try to register. Let’s see what happens. You can see there’s something happening here, there’s no. Okay, I just hit ”Submit”. Okay. I’m just going to increase that, and then submit,

and you will see that registration failed because it was from account number two, where only owner can do it. Anyway we have registered three of them, and let's see whether we can vote on all of them. So, I'm just going to go back to one, start voting from account one. Account number one has got twice though. So, let's see whether we can vote for banana, the owner is voting for the banana. That's good, and that went successfully and the rest of them, the three of them outsiders and other than the voter are going to vote for some other maybe apple this time. So, they will vote for an apple and it should go through fine and successfully. Let's see, we have one more account three, and that's also apple and that went through all right. You can see the gas points and other things are indeed truly reflected and taking care of the MetaMask management. I'm going to go to account four and again vote for an apple. That should go through fine. We can see the gas limit and other thing displayed. Finally, I'm just going to go to account number five, which has not been registered and see what happens here. It has not been registered, we do have a test case and I'm just going to see whether it can vote. You can see that there is no gas limit here, that means that account is unknown, when I submit, it's going to say it failed because we do have a test case holding it, and making sure that it will not work. Note in this case, at the solidity contract is written in such a way, unregistered cannot go. So, we aborted enough and let's see whether we can declare the winner, it should be apple. Yes, the apple is a winner. That is a complete DApp demo. In summary, we learned to set up a Web front-end and tested a complete DApp. You observed that Truffle provides a user-friendly environment for developing and testing a DApp. By now, Truffle commands should be quite familiar to you. Develop, edit, compile, migrate, test, all with Truffle in front of them and with any parameters you may need. This module illustrated all the steps needed in developing and testing a complete DApp. That's great. Please make sure you're able to complete the development and testing before moving on.

2.3.4 Resources

Smart Contracts and Truffle 101. Part 3 - Simple functions and tests
Ethereum dApp tutorial - Front end. Part 1

2.3.5 Quiz

1. What role does app.js play in the Dapp design?

- ◯ Deploys smart contracts
- ◯ Instantiates object needed for the web client to communicate with the blockchain node
- ◯ Initiates the blockchain network
- ◯ Handles business logic of Dapp

2. How are accounts in the test chain created by truffle develop connected to the Dapp?

- [ ] Metamask
- [ ] Ganache
- [ ] Web3 script
- [ ] Truffle CLI

3. What is the command to start web server for the Dapp in our lesson?

- ◯ npm dev run
- ◯ npm start server
- ◯ npm start
- ◯ npm run dev

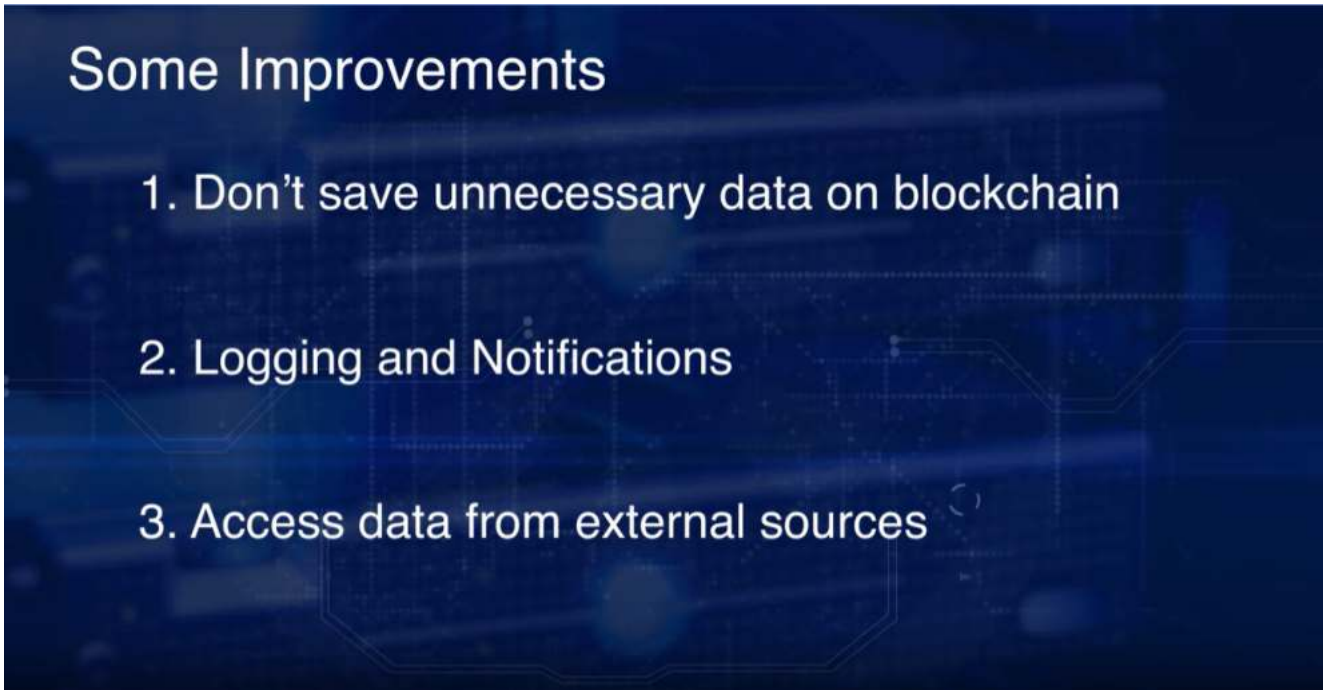
Chapter 3

Design Improvements

3.1 Solidity Features

3.1.1 Part 1

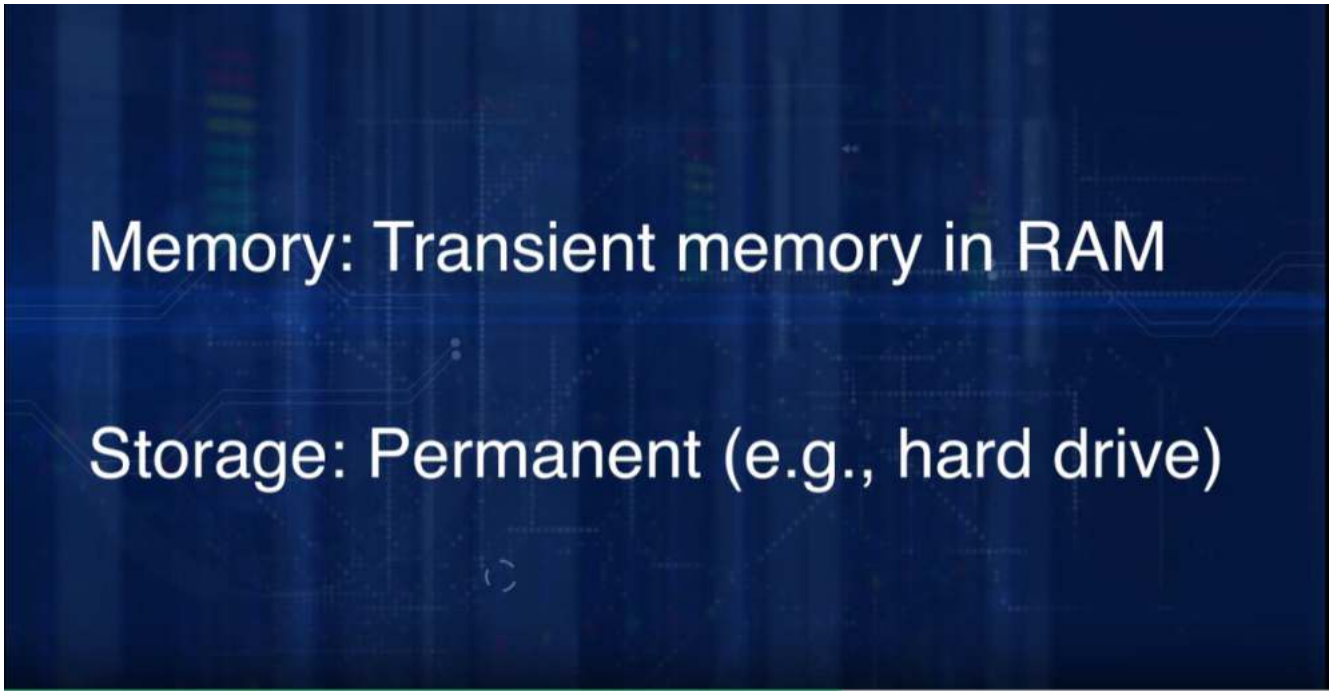
At this point, you should know the basic concepts of a smart contract; how to design and code one in solidity language and how to test it using remix environment. In the last module, we discussed the complete D-App built on a smart contract based solution. This module is about making a D-App better, more efficient, and practical with respect to the blockchain. There are improvements that help in the design of practical decentralized applications around smart contracts. We will discuss a few of the many improvements possible. Here are a few. Number one, we need to make sure we don't save unnecessary data on the blockchain. Recall blockchain is not a data repository.



Number two, is there a way to log the events happening on the chain? Either forgetting a notification of completion of an operation or if something needs attention. Number three, smart contracts cannot access the outside links, C, to get today's US prime interest rate. Is there a provision to get outside information that will still keep the blockchain consistent? US daily interest rates are the same or lower for a given date as long as you normalize the day to epoch time. Recall that epoch time is the universal time in seconds from January 1st, 1970. Then how do we solve this problem of accessing an external resource from a smart contract? We're going

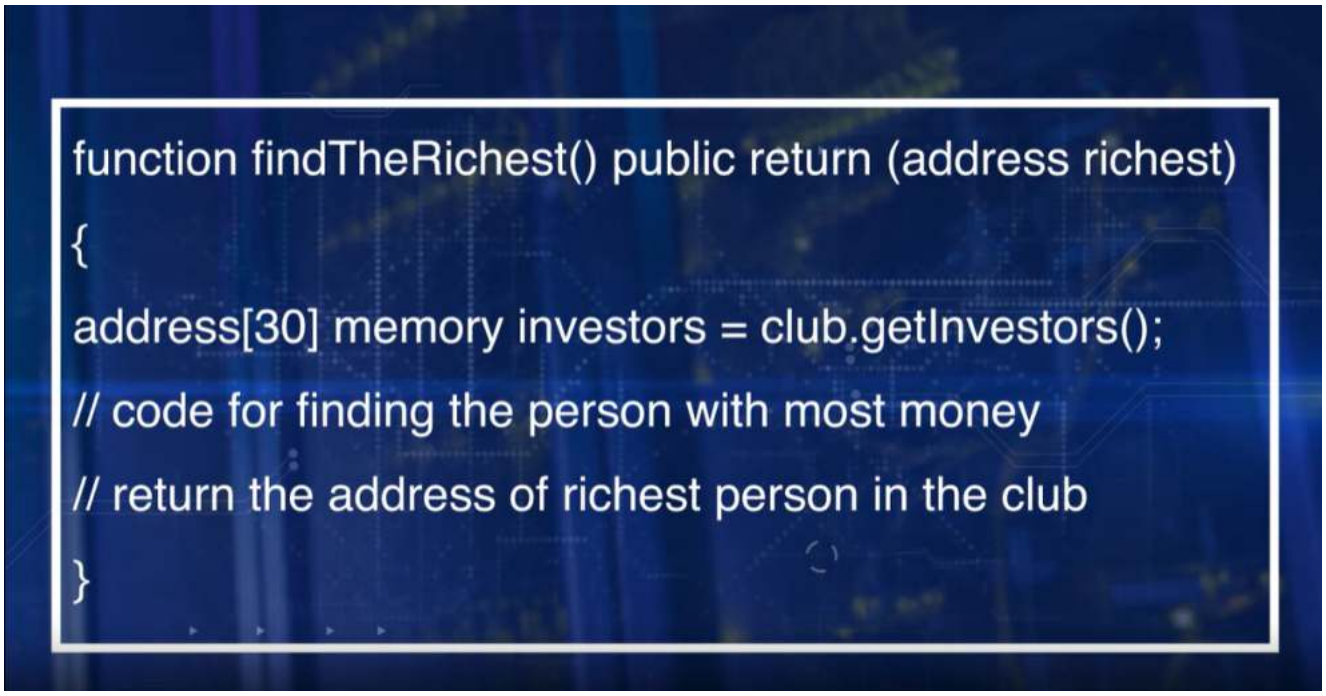
to address these in this module. On completion of this module, you will be able to list additional features of solidity that make a smart contract efficient, practical, and useful.

Reuse code and apply hierarchical structuring of a smart contract for specialization and generalization. Illustrate the use of events and pushing events to subscribing applications. Explain the use of smart contract named using oracles.



Software development, as you know, is incremental and iterative with every iteration improving over the previous. Did you know that security was afterthought and was retrofit into HTTP protocol? From HTTP to

HTTPS. See the beginnings of this improvement in RFC 2660 and RFC 2818. Ethereum improvement proposal framework, EIP, that manages ongoing improvements to ethereum protocol is a good example of iterative process improvement. Recall that EIP is similar to the request for comments RFC of the Internet Engineering Task Force, IETF.

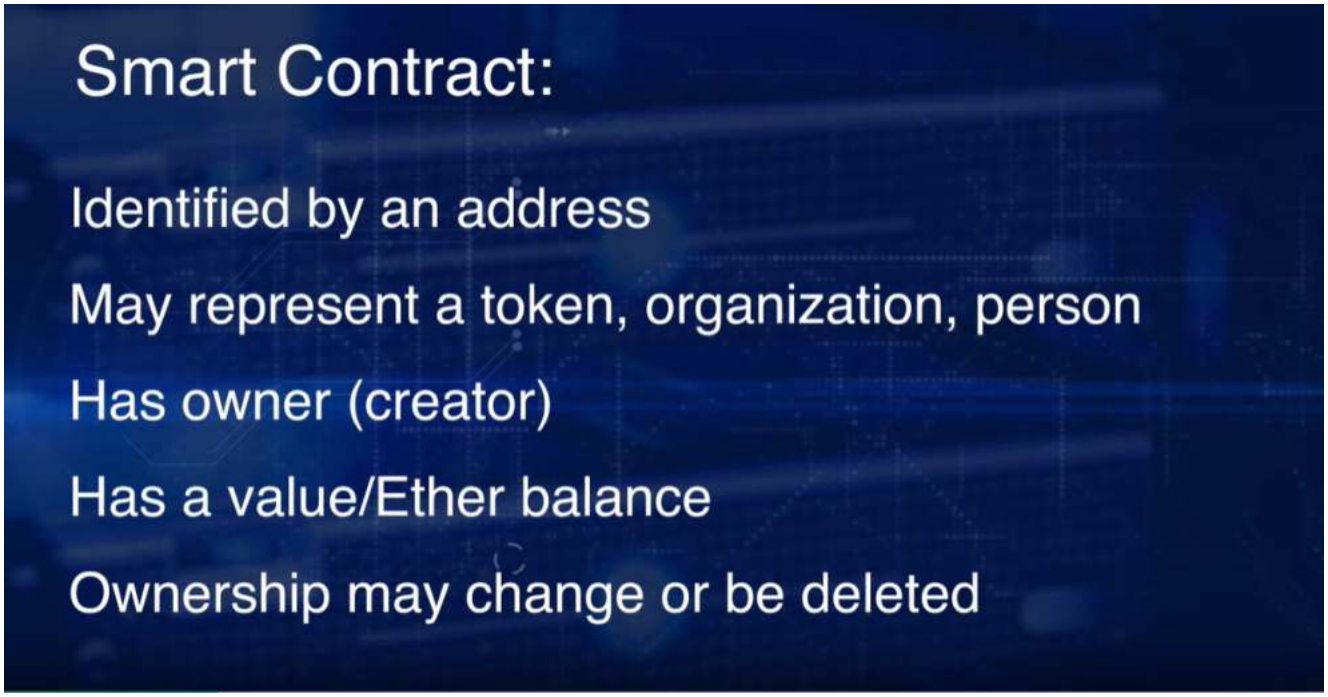


Minimize the footprint on the immutable ledger in the blockchain. Manage ownership and lifespan of a smart contract. Organize related smart contract using interfaces in inheritance for reuse and proper type classification. For minimizing the footprint on the blockchain, let's consider storage versus memory attributes of data. By default, all the state changes did variables define in the smart contract, are saved as state on the blockchain. Recall from, course one, that the state hash of the ethereum block header, is computed using Merkle tree or Patricia Merkle tree of the state variables. We would like only a sliver of data that is needed as state for provenance, governance, and immutability to be saved on the blockchain. So, our goal is to minimize the state or the footprint of the Smart Contract and lower the gas costs. One of the ways you can accomplish this, is by correctly using memory and storage keywords. Memory and storage are indeed keywords in the solidity language and they mean the same as in your regular computing system. Memory is transient memory in RAM and storage refers to persistent store in the permanent storage device like your hard drive. Memory is temporary and is a race between function calls. Memory is a byte array. It's cheaper to use an Ethereum. Since everything cause gas points, we have to be aware of this. Every contract is assigned a storage space and this storage is permanent. The values in it persists between function calls. Storage is a key value store of 32 bytes each for the key and the value. All storage data is considered a state and used in the computation of state hash route of the header. Using a memory location, cause few guess points, 1-3, whereas storage costs in the order of thousands of points. Twenty thousand points to set up, 5,000 points to change a value and so on. You may may wonder why we are so concerned about this memory versus storage difference. Understand, that a smart contract is global and a copy of it is present in every full node. The state footprint hash is present in every block. We don't want unnecessary details on the immutable ledger. There are many ways we could manage the memory versus storage trade-off. Here we'll consider only the composite data struct and arrays. In a solidity smart contract, struct and array are by default assigned storage instead of memory, even when they are local to functions. While a struct or array is used as a parameter or a local variable in a function, declare them as memory variables. If the memory attribute were not there, temporary variable investors would have been a storage variable wasting state variable space. Since we declared investors to be memory type, its space of 30 by 20 bytes is on the temporary memory is some permanent storage of the smart contract. And we all know that a smart contract once deployed is immutable and permanent. We saved 600 bytes of storage on EVM. This space efficiency improvement, is not just for one node but for every full node on the blockchain. In the development of a smart contract in solidity language,

consider using memory variables for any transient data. Use storage variable only for something that needs to be persisted on the blockchain. Now, let's move into the second improvement we'll discuss; how to kill or delete a smart contract.

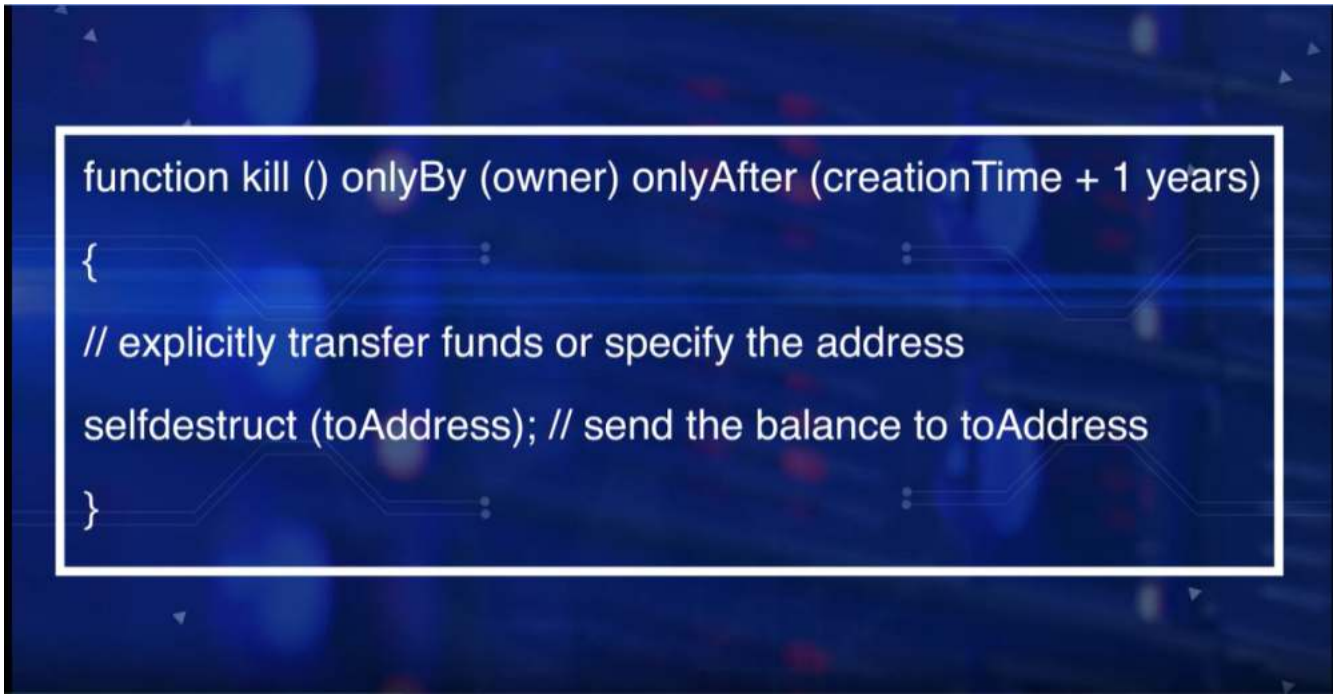
3.1.2 Part 2

Have you wondered, what if a deployed smart contract is outdated? Consider this scenario, in the US, 2016 income tax laws are invalid. So, are the smart contracts we might have deployed for this. Thus the smart contracts are now useless What happens to the money tax that any of the smart contracts would have collected? Can we change the ownership of these smart contracts to a trustee account, as we deploy smart contracts for representing the new tax laws? These are questions to be carefully addressed not just for the Internal Revenue Service, but for any business or organization that may deploy a blockchain and a DApp smart contract solution.



Recall that a smart contract is identified by an address. This is an important concept. It may represent a token, an organization, a person, human or otherwise and the related logic rules and operations. A smart contract has a owner.

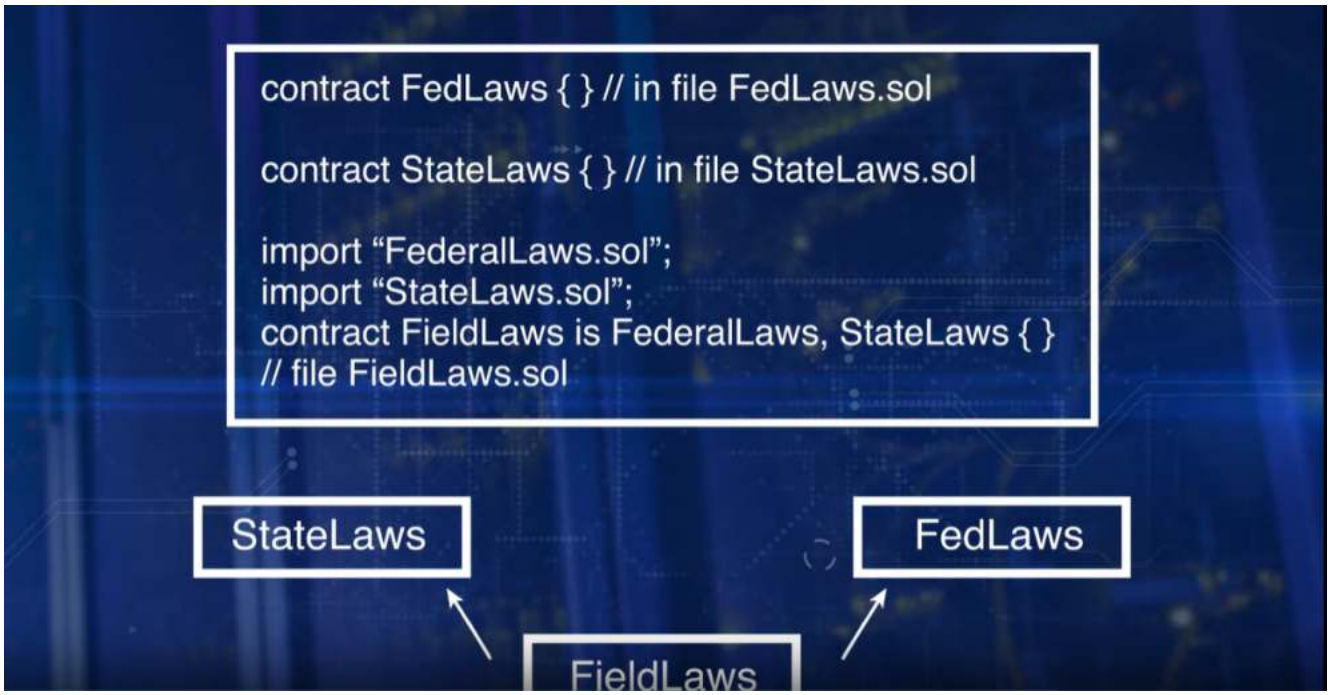
By default, it is the creator of the smart contract. It has a value or ether balance. Its ownership may change or it may have to be deleted. Given all these complex attribute, how do you manage a smart contract? To delete or kill a smart contract, Solidity provides a function called self-destruct. Understand this call is irreversible as per the default blockchain protocol. Once killed, the smart contract cannot be revived or accessed. It is like holding an expired spaceship into the abyss of space. Another the good programming practice is to make sure only the owner or the owners of the smart contract or an authorized person has permission to self-destruct a contract. Assume, a time duration is also specified for the contract.



Modifiers are used to enforce the destructors address and the time duration for this smart contract. Here

is the code. What happens when a kill function is executed. The smart contract code is removed from the state of the blockchain, if the conditions specified by the modifier are satisfied. In our case, the conditions are that, the function is called by the designated address and the time specified has elapsed. The remaining ethers stored in the address of this smart contract is transferred to the address specified in the parameter of the self-destruct call.

The smart contract itself is inaccessible by its current address. The next concept is equally important for smart contract design. Recall that smart contract may represent an asset or a business. During the lifetime of an asset, ownership may change. It may change due to the sale of an asset, donation of an asset or by bequeathment. These are just a few. There are many more scenarios for ownership transfer.



This function can implement a transfer of ownership, but make sure you qualify the transfer with a modifier or modifiers so appropriate rules are enforced from managing ownership and lifetime. We are moving into managing relationships among smart contracts. So, far we've been creating single smart contract. It is possible a complex application could be composed of many smart contracts, one instantiating the other or inheriting others. It is also possible that the symbols or namespace for an organization is defined in a common smart contract that needs to be applied.

Import command can be used to include files in your smart contract. At the time of compilation, associations are made between the important files and the current smart contract and the byte code is generated based

on this association. This leads us to the programming practice three. Composability, specialization, and generalization are important tenets from object orientation, that can help in designing smart contract solutions for complex problems. This also promotes re-usability of code and standards across organization and participants, as you will see in a later module. We'll explore inheritance next and its importance in smart contract-based applications. For example, federal laws referred to here as Fedlaws, can be a legal framework that can very well represent an interface smart contract, letting the state and the field offices implement them, as they see fit. In this case, StateLaws and FieldLaws are also smart contracts and they are associated using the import statements. This is inheritance at work. As a subtopic of relationship among smart contract, it is possible to create libraries and import existing libraries into your Solidity smart contract code. Libraries are special smart contract with no ether balance, no payable functions and no state to be stored in the block chain. A fine example is a SafeMath.sol by Zeppelin, that can be included in your smart contract for any math computation. Import SafeMath.sol. Many libraries are yet to be created. You can be sure, many domain specific libraries will emerge. Here is an opportunity. You too can be a creator of a useful library for your application domain. Summarizing, we explored three important concepts in this lesson, memory versus storage attribute for data in a smart contract, managing lifetime and ownership of smart contracts and inheritance. Lifetime and ownership is something unique to smart contract, and some DApp developers may tend to downplay the importance of this. You should not. You will explore many applications of inheritance concept in the next lessons and also in the next module.

3.1.3 Resources

Scaling Blockchain Computation and Storage by Daniel Larimer

3.1.4 Quiz

1. What type of variable is assigned to a local array inside a function by default?

- ○ memory
- ○ persistent
- ○ referenced
- ○ storage

2. How is a smart contract identified?

- ◯ Its address
- ◯ ABI
- ◯ Deployer's address
- ◯ Its name

3. You can not change the ownership of a smart contract

- ◯ False
- ◯ True

4. Using which keyword you associate an interface with a smart contract

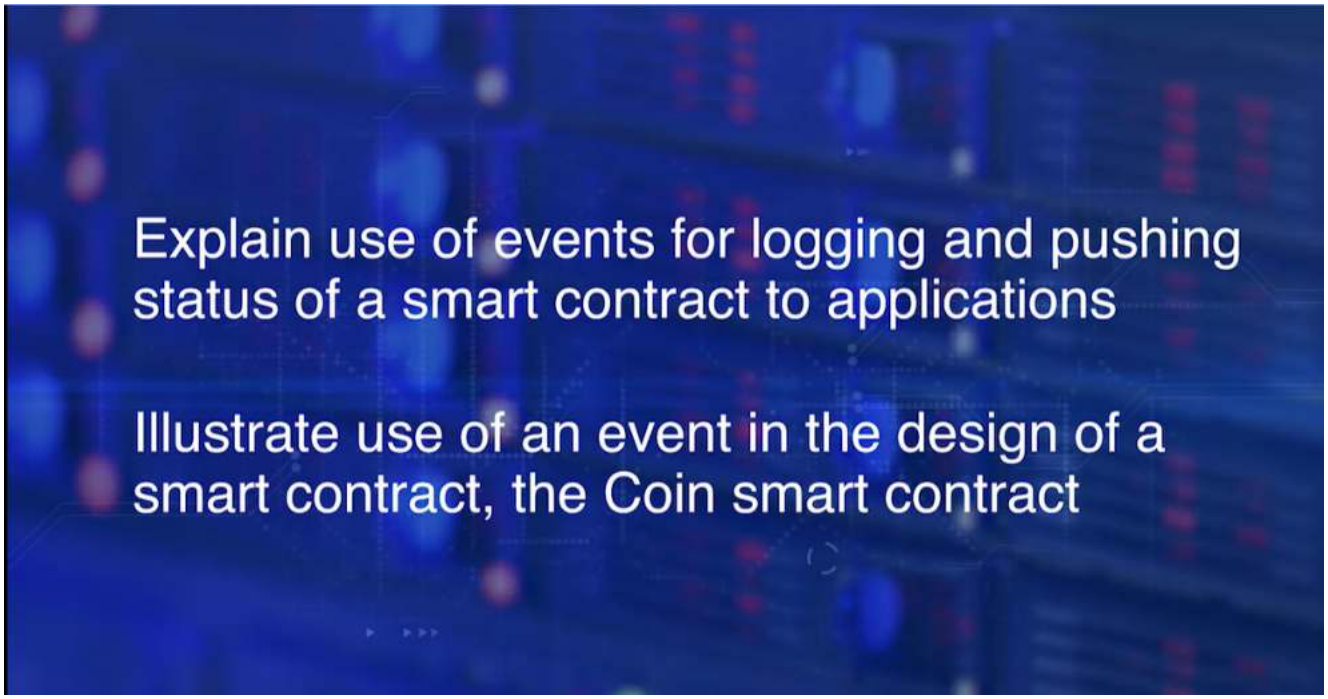
- ◯ contract
- ◯ library
- ◯ import
- ◯ pragma

3.2 Event Handling

3.2.1 Part 1

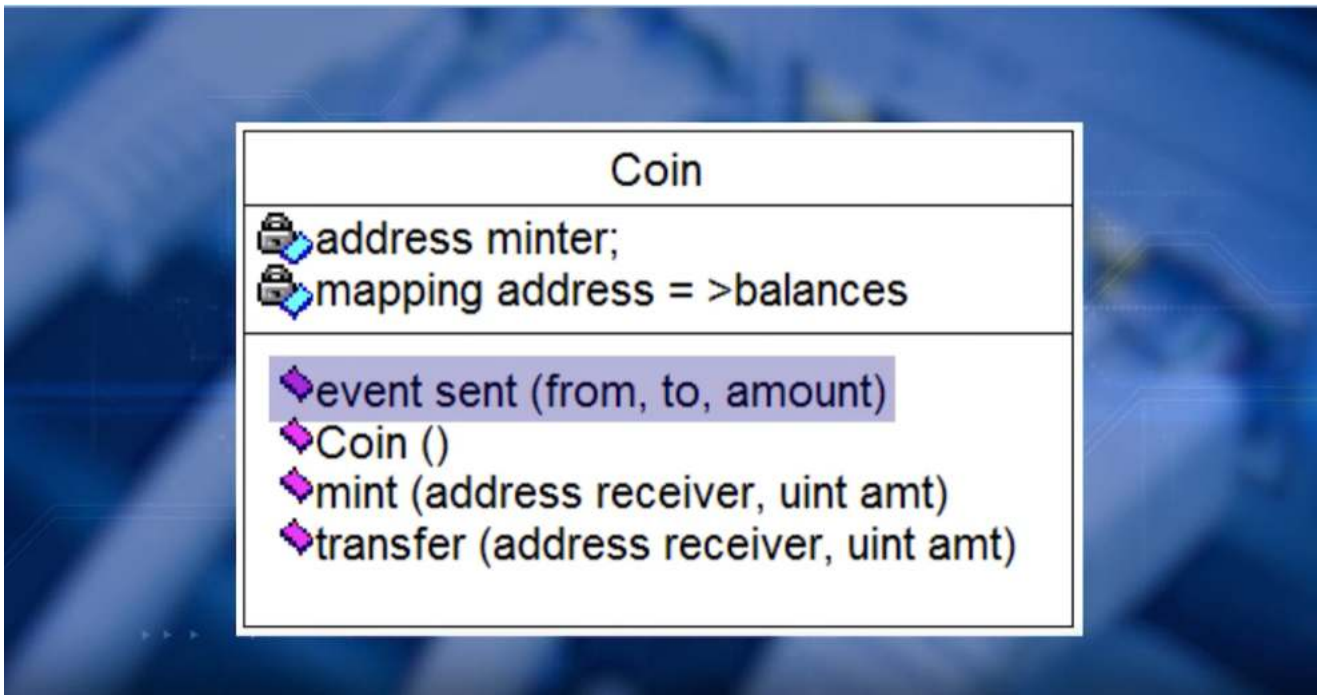
Why do we need events? What purpose do they serve? Recall that we introduced events in course two. Events are useful for pushing notification to indicate completion of a smart contract operation, say the execution of a transfer. Events can be used to notify any event occurrence within a smart contract and the blockchain.

Events are useful for logging activities of a smart contract. In this role, event logging is like your printf or console log. Logs can be reviewed to audit the functioning of a smart contract. Event enables asynchronous operations between an event sender and the receiver. Event framework is an efficient alternative for polling to check if a request has been completed. On completion of this lesson, you will be able to explain the use of events for logging and pushing the status of a smart contract to application, illustrate the use of an event in the design of a smart contract, the Coin smart contract. How do you program set up an invoke an event?



You define an event by its name and parameters. An event can be parameterized like a function. For example, in the coin.sol, you will see event sent with parameters from, to and amount. You invoke an event by

its name and actual parameter values. This will push or log an event occurrence for its listeners. For example, in the coin.



sol you will see sent with parameters sender, receiver and amount. How do you consume an event? You setup the listener for an event or events, and consume or catch an event notification, and act upon it. We're going to illustrate events using Coin de-application. Coin smart contract has two data items. First one is the address of the minter or the owner. The second one is the mapping between the account address and the Coin balance. Please understand this is not the ether that is used for paying gas points. Event sent is defined with three parameters. Event is pushed or logged after the transfer is completed within the transfer function. We'll demonstrate the Coin DApp on truffle ID and metamask in a bit. Coin has a constructor. It has a mint function that only the minter can execute to mint new coins for a given address. You will have to be in account one in metamask or in the minters account to mint. The last function is the transfer that can transfer coins from one account to another. You will have to be in the sender's account in metamask to be able to transfer. Now, there's one more item left. How do you handle the event? There are many ways to set up the event handlers. We have chosen the approach suggested in the truffle documentation. This involves handling it in JavaScript app, App.js. This approach enables you to conveniently handle the events and inject the JavaScript assets in the web to display the event notification. If you haven't already done so, now you can download Coin.zip from the resources section into your workspace. Unzip Coin.zip, cd into the Coin directory. Truffle develop do this in a different terminal. Truffle compile in the current terminal. Truffle migrate –reset. Set up your metamask to connect your test chain. Recall we explained it earlier. Then run the test server npm run dev. Now, go to the Chrome browser where you set up the metamask. In the browser link, enter localhost:3000. It should open up the Coin interface to the DApp, see the events in action. By now, the truffle DApp development process should be pretty routine to you. Prepare the various directories, source code and the assets, truffle develop in a terminal and set up the metamask, create accounts, truffle compile, take care of any syntax errors, truffle migrate –reset, that deploys the smart contract and npm run dev, that deploys the lightweight web server. You're all set to interact with the user interface. In the metamask, make sure you have the right account. For coding the DApp, you can apply all that you learned about Solidity in course two. You will need some basic JavaScript and HTML UI experience to complete the front end.

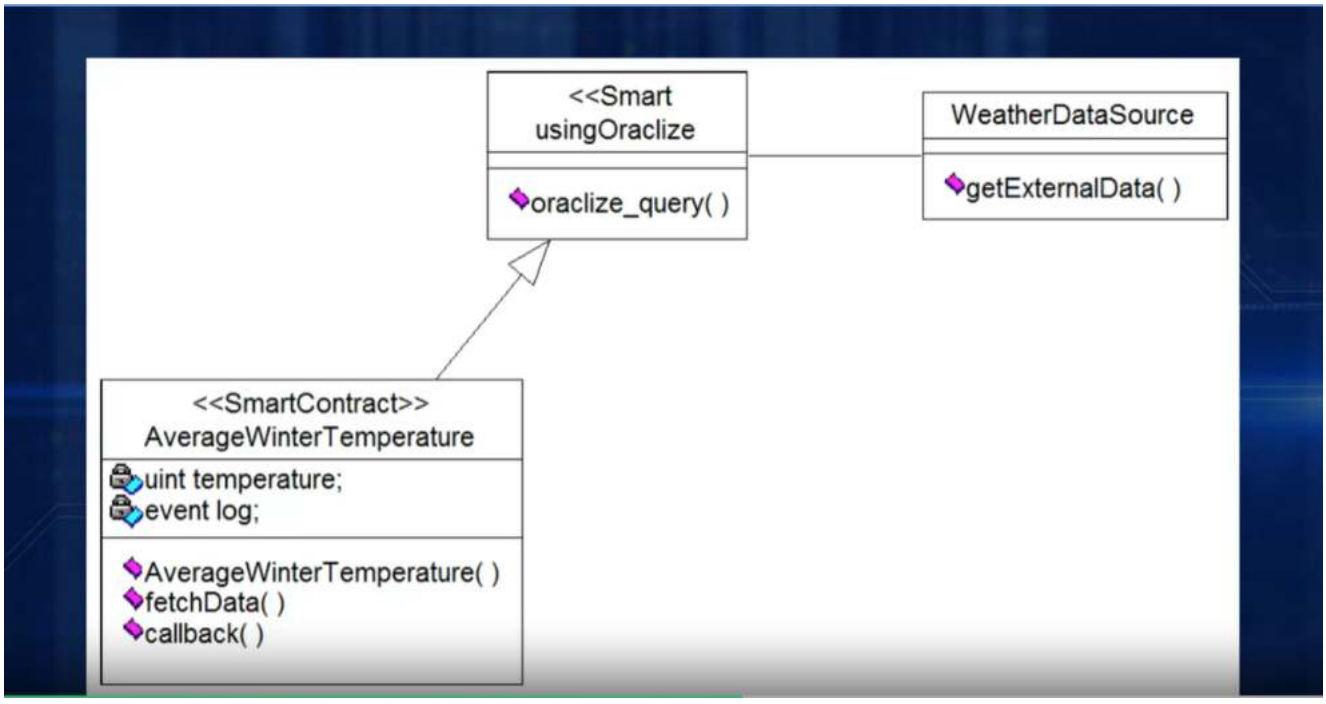
3.2.2 Part 2

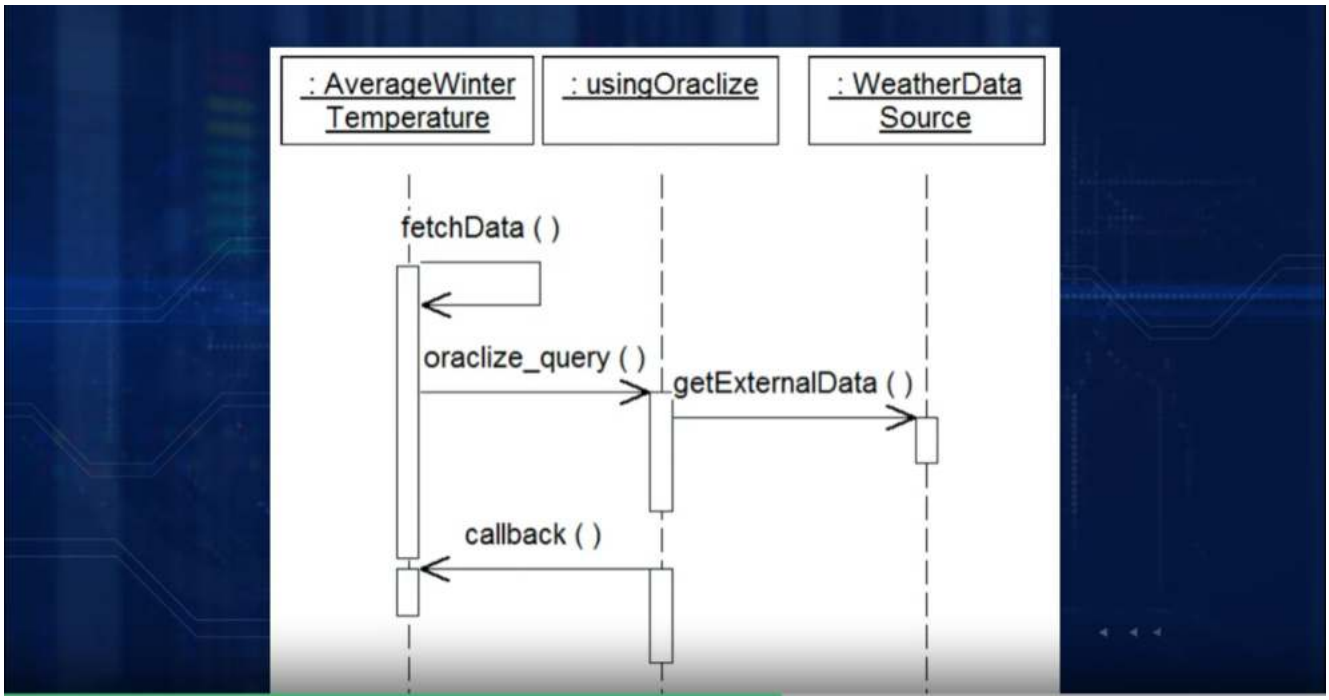
You can illustrate the event logging using the Coin smart contract. I already copied the coin.zip from our resources. Let me unzip it and and that'll create all the artifacts needed for the coins, so I'm just going to go

into the Coin folder where I have it. So you can look at it in the contracts and the Coin.sol has three functions including the constructor, the Coin constructor, Mint, where the coins are minted by only the minter, and the Transfer that can happen between any account when the transfer is completed. A sent event is logged with three attributes, sender, receiver, and amount, and you can see the event getting defined here. And let's run it and see. And I'm going to truffle compile, and you can directly go to test if you would like. And I'm also going to test it, we put some test cases here and you can examine that later on by looking at the test.js. Truffle test, and then you can see the test, and then I'm going to go into truffle migrate, Reset. That's it. Test takes care of it but I'm just going to go through and have it done anyway.

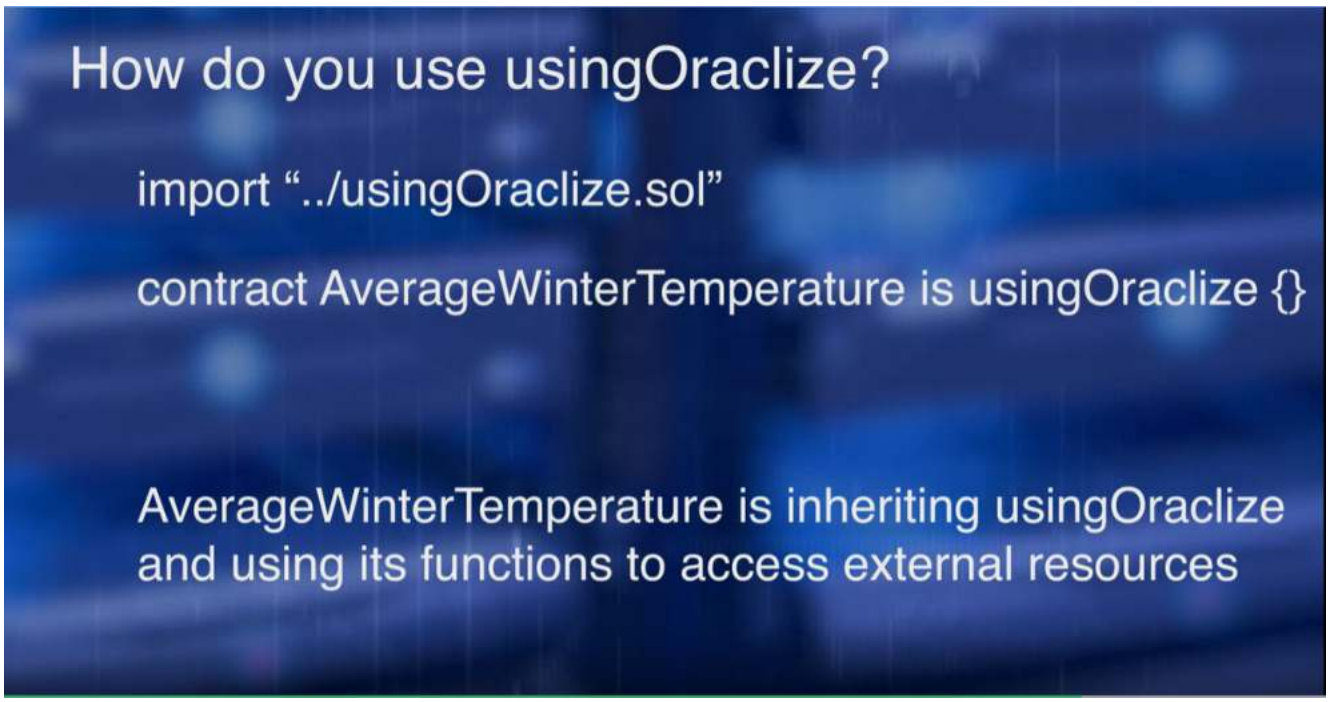
So and, at this point, I'm going to run the npm lite-server And that opens up the posts, and this is the interface you would see. There's an interface for the minter, which is check balance, mint coin, and send coin. There's another interface for all the regular accounts which doesn't have the mint coins. Okay, so let's go back to our, Metamask interface and, Go to local host and, And you can see, this is the interface the regular account will look like. And I've already created the content Metamask. Since it's Account 4, the interface is only check balance and send coins, so let's see whether we can do Account 1.

In Account 1, and I'm going to refresh it to see the interface that you have. When I check balance, I have zero balance. So I have to mint some coins, so I'm going to mint coins and I'm going to mint for myself or the creator of the account, I'm going to use 2,000 coins not Ether. And 2,000 coins will be created, this is mean successfully created as we can see above. And I'm going to check the balance, of course, let's refresh. And I'm going to check the balance . Now, you can send coins to any other account, I'm going to send some coins, it would be 600 coins to that account, and then we sent it. As you can see, this is the event log, 600 coins were sent from this account, that is the Account 1 to the account, another account that we have.





And let's check the balance of that account. So I'm going to go back to the second account in here. And Account 2, and refresh. And I have here the interface with just two functions, and I'm going to check balance. And you can see, the $600 that you sent came in. And I'm going to exercise the send, and I'm going to send back to the owner, maybe 300 of that, and I'm just going to send. And it's going to ask for, and you can see, 300 coins were sent from the owner, not the owner, the current account holder to the owner of the smart contract. And let's go back and check whether the smart contract owner received, that the minter received the $300, so I'm going to go back in to Account 1, and refresh to get my interface, and I'm going to check the balance, it should 1,700.



Summarizing, we explored the use of event feature in solidity for pushing events and logging events. We also had another chance to try our hand a Truffle IDE Dapp development processes with Coin smart contract and became more familiar with the interaction of UI and Metamask with an underline blockchain server.

3.2.3 Resources

Solidity Events Tutorial - Using Web3.js to Listen for Smart Contract Events
Calls, transactions, events, filters and topics
Technical Introduction to Events and Logs in Ethereum

3.2.4 Quiz

1. How is an event defined?

- ◯ They are defined as a state inside the contract
- ◯ They are defined same as functions
- ◯ They are defined with name and parameters

2. In the Coin Dapp we discussed, what is event sent used for?

- ◯ For debugging purpose
- ◯ To invoke eth.sendTransaction() from front end
- ◯ Pushing a notification that coins have been transferred between sender and receiver
- ◯ Sending coins from sender to receiver

3. What are the parameters for the sent event in coin smart contract?

- ○ String from, String to, uint Amount
- ○ uint from, uint to, uint Amount
- ○ Address from, Address to, uint Amount
- ○ Address to, uint Amount

4. What is an event used for?

- ○ For debugging purpose
- ○ Pushing smart contract state changes to any monitoring applications
- ○ To invoke eth.sendTransaction() from front end
- ○ Sending coins from sender to receiver

3.3 Oraclize

3.3.1

Recall that smart contract operates in a sandbox. It cannot call outside function or link to an external resource. So how do you get access to an external resource? By using a special smart contract called using oraclize. On completion of this lesson you will be able to explain a method for accessing external resources from a smart contract and illustrate the working of the oraclize smart contract. Why can't a smart contract access outside sources? Depending on the source code, it may affect the global consistency of the blockchain. The results of an operation on the blockchain has to be deterministic. These conditions limit the applicability of the smart contracts in many real world applications that may involve obtaining facts, data, and assets from external world sources. Moreover, the data has to be obtained at the time of execution and may not be available at the time of deployment of the contract. Let's look at some examples. How about the temperatures on Mount Kilimanjaro on a given day? It is a universal fact, but it has to be obtained in a given day from an authentic external weather source. As a second example, the stock market data, high and low price of a stock on a particular date, on NASDAQ market. The oraclize concept addresses this problem of fetching external data from a smart contract. What is oraclize? When you look up the meaning of oracle, Merriam-Webster's definition of oracle is, an authoritative or wise expression or answer. This definition very closely defines the rule of oraclized service in the smart contract dApp development. Oraclize is described as a data carrier between the web resources, APIs and the URLs, and the smart contract. Of course, oraclize is outside the blockchain protocol. However, it is a useful component of dApps that facilitate the availability of real world facts needed for the functioning of certain smart contracts. How do we use oraclize service? UsingOraclize is a smart contract that provides minimally a query function to access external sources. It not only fetches the data, but it also provides proofs and authentication about the sources if needed by calling

the smart contract. The data requested is returned through a callback function since accessing the data and verification may take some time. We do not want the blockchain smart contract waiting on results. The following is a simple class diagram relating the smart contract, usingOraclize and an external data source. Now let's look at the sequence diagram to follow the timeline of execution of the various functions. With the smart contract, average winter temperature is deployed. It calls fetchData function. That in turn in-walks $oraclize_query$ with the URL for the data source. Since it may take time to fetch the data, a call back function is provided so that it can invoked when the required data has been fetched. UsingOraclize accesses the external data source, authenticates and sends the data with the proof to the originating smart contract. In this case, AverageWinter Temperature. How do you use usingOraclize? First you import usingOraclize smart contract using the import statement. Then the contract AverageWinter Temperature inherits from usingOraclize. Contract AverageWinter Temperature is usingOraclize. Here, AverageWinter Temperature is inheriting usingOraclize and using its function to access external resources. As you observed, using usingOraclize smart contract is simple. Let the smart contract that you write inherit usingOraclize. It provides functions such as oraclize query, and lets you specify the data source address. You specify the external data source address in the query. Additionally, you may also have methods to verify the proof and authenticity of the data fetched. In summary, we learned about some best practices that can improve the basic design of dApp. First one is of the data structure level, memory versus storage. That helps to minimize the footprint of the Dapp on the blockchain. The second one is about event logging that is useful for asynchronous operation of Dapp clients, which improves efficiency. The third one is about accessing data and facts from external sources that is essential for many practical Dapps. These are just few of the many best practices exclusive to smart contracts and Dapps that'll improve your basic Dapp design.

3.3.2 Resources

Getting data from the internet with Oraclize
Oraclize Documentation

3.3.3 Quiz

1. What is the purpose of a callback function, when using Oraclize?

- ○ Synchronous processing of data request
- ○ Batch processing of data request
- ○ To send data request to Oraclize
- ○ Asynchronous processing of data request

2. What is Oraclize as described in the lesson?

- ◯ To check if a smart contract is mined and confirmed
- ◯ Means for smart contracts to push data to external sources securely
- ◯ Means for smart contracts to access data from external sources securely
- ◯ Communication between two different blockchain network

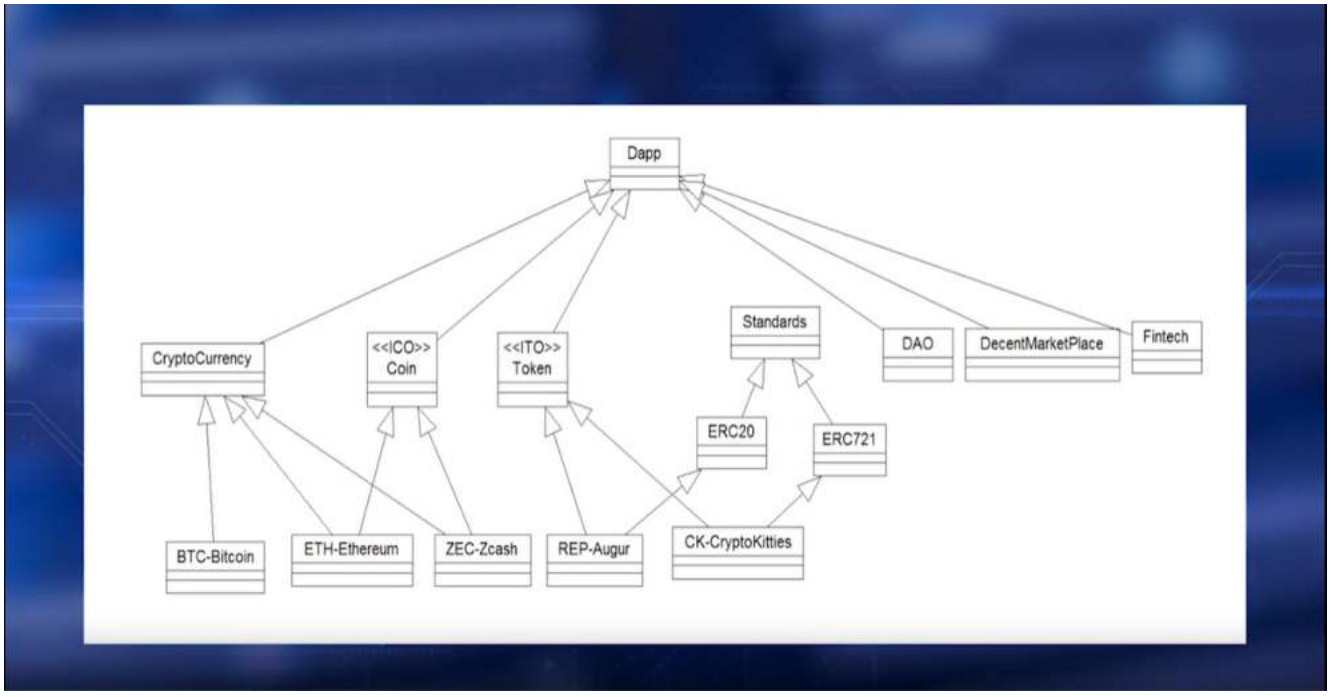
Chapter 4

Application Models and Standards

4.1 Dapp Models

4.1.1 Part 1

We've learned a lot about developing end to end Dapps in this course. Have you wondered, what problems Dapps can solve? What are some use cases? We can find many current examples for Ethereum-based Dapps in this link. Our aim in this open ended module of the Dapp course is to explore some of the models of Dapp and its ecosystem. We hope these discussions, will encourage you to think about newer and possibly disruptive application and application models. There is still work to be done for developing Dapps and also contributing to the protocol improvements and imagining newer Dapp models. On completion of the module, you will be able to list the different application models of Dapp and explain Ethereum improvement proposal (EIP) and a Ethereum requests for Comments frameworks. You will also be able to explain the importance of standards in a decentralised system and explore two standard tokens: ERC 20 and ERC721.



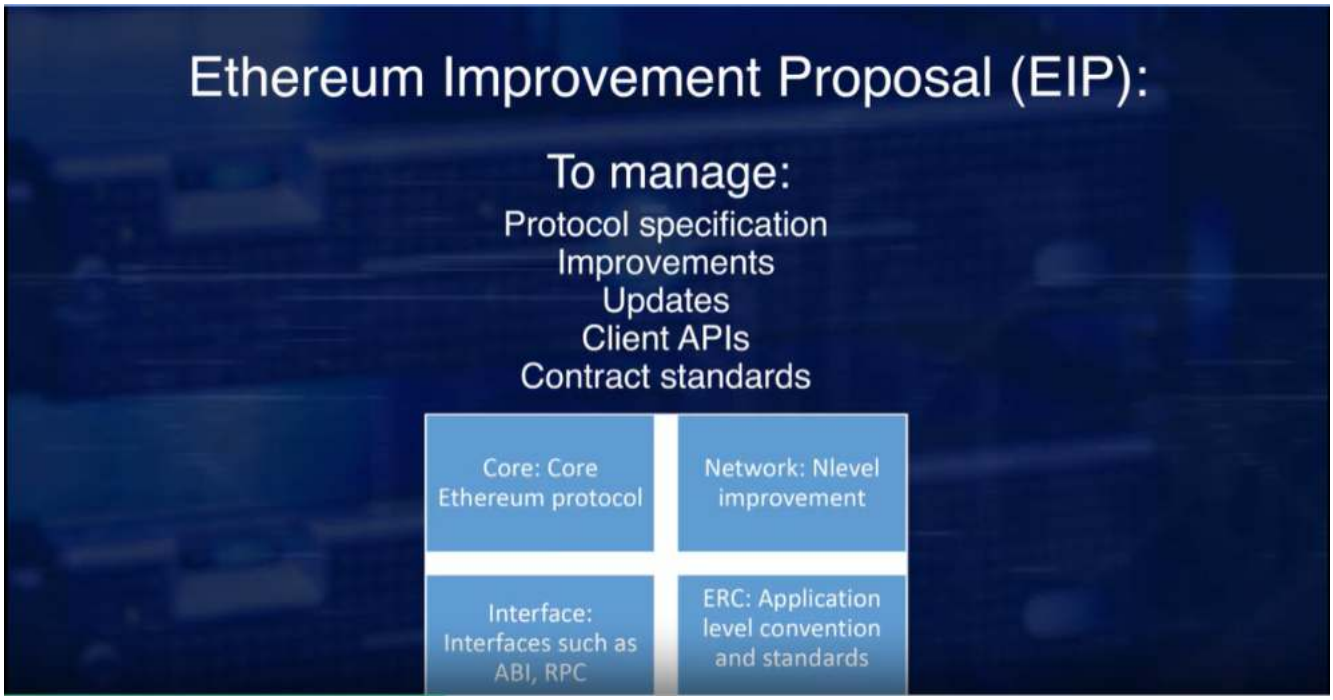
We are going to start this lesson on Dapp model with a hierarchy diagram that gives a high level view of Various Dapp model, starting from the genesis of Bitcoin. In this lesson, we'll review each of these models. We acknowledge that there are many others not included and that this diagram is by no means complete. We encourage you to add other Models to this diagram. After this lesson, you will be able to list some important

models of Dapp and explain the evolution of Dapp models from cryptocurrencies to cryptoassets. Let's explore the models one by one. Bitcoin is the pure cryptocurrency. Every Bitcoin was minted by the protocol, except for the genesis transaction from Satoshi to Hal Finney. Innovation of bitcoin is the decentralized payment system that is still running, though facing issues with scalability and price volatility. Next, the coin and the initial coin offering. Bitcoin code-base is open source and the initial response to Bitcoin from the developer community, were the releases of many new crypto currencies. Then came the innovation of Etherium Smart Contract. It helped automate the crowd funding for startups using a virtual instrument called Koine and the process of initial coin offering. This is similar to an initial public offering IPO of company stocks. Coin ICO use a block chain to record the distribution of the coin, receive funds, specified rules, and to enforce any preconditions and policies. For example a policy of some of the Chinese ICOs is that they would not receive funds from the USA. That is atleast at the time of our recording. In the Dapp hierarchy, we have included only as coined Dapps, Ethereum, and Zcash. However, there are hundreds of ICOs and coin offerings. It has become a popular mode for crowdfunding. Did you know the Zcash focuses heavily on privacy and uses closed-code? We classify any cryptocurrency that raise fronts through ICOs in the coins Dapps. Both Ethereum and Zcash used ICO processes for that initial funding. That's the reason for classification of Ethereum and Zcash in this category. The next Dapp we will look at is Tokens and initial token offering. Sometimes tokens and coins are used synonymously. However, we distinguish between these in our model. Token is like a Dapp coin but, its offering is typically associated with an asset or utility. For example, in the highly popular cryptoKitties, a token is used to represent the kitty or all the rules for the creation lifecycle, breeding etc are written into the immutable Ethereum block chains Smart Contract. At pre-determined times, a certain number of new tokens are released and auctioned off for raising new funds. Assets are Kittys in this case, also appreciate or depreciate in price based on the demand. So the tokens are non-fungible. In other words, value for Tokens can change during its lifetime. Auger is a prediction Dapp that depends on the wisdom of crowd and their reports on reputation of a product to predict markets. These decentralized participants or crowd get paid in REP tokens or reputation tokens. These tokens represent a fraction of the fees collected if the transaction gets executed, based on the crowds reporting. Once again there are so many token Dapps in play, but we have covered only two. In the hierarchy diagram, you will also notice that both Auger and cryptoKitties inherit from standard tokens ERC 20 and ERC 721 respectively. More on this, in the next lesson. Turn your attention to DAO, a Decentralized Autonomous Organization is an early innovation in Dapp. DAO was an investment instrument deployed as a smart contract on Ethereum block chain. Autonomous venture capital Dapp to be exact. The idea was to showcase an autonomous organization without traditional corporate governance structure for a decentralized anonymous Fund-Raising, and automatic investing. DAO was deployed prematurely without extensive testing. But the idea itself was well received and DAO raised funds beyond expectation. However, a vulnerability in the code was exploited by hackers.

This resulted in a digital heist of a significant portion of the collected funds. You can read more in the ether thief, placed in the resource section. Actual events do indeed read like a thriller. It is a nice plot for a movie. The readers digest version of this. This DAO heist and the solution to fix it, resulted in the unexpected hard fork of Ethereum in July 2016. This hard fork split Ethereum to ETH core as we use it now an ETC, the Ethereum classic that still has the DAO bug in it. In spite of all the negative publicity, we feel that the Decentralized Autonomous Organization is indeed an innovative Dapp If done right.

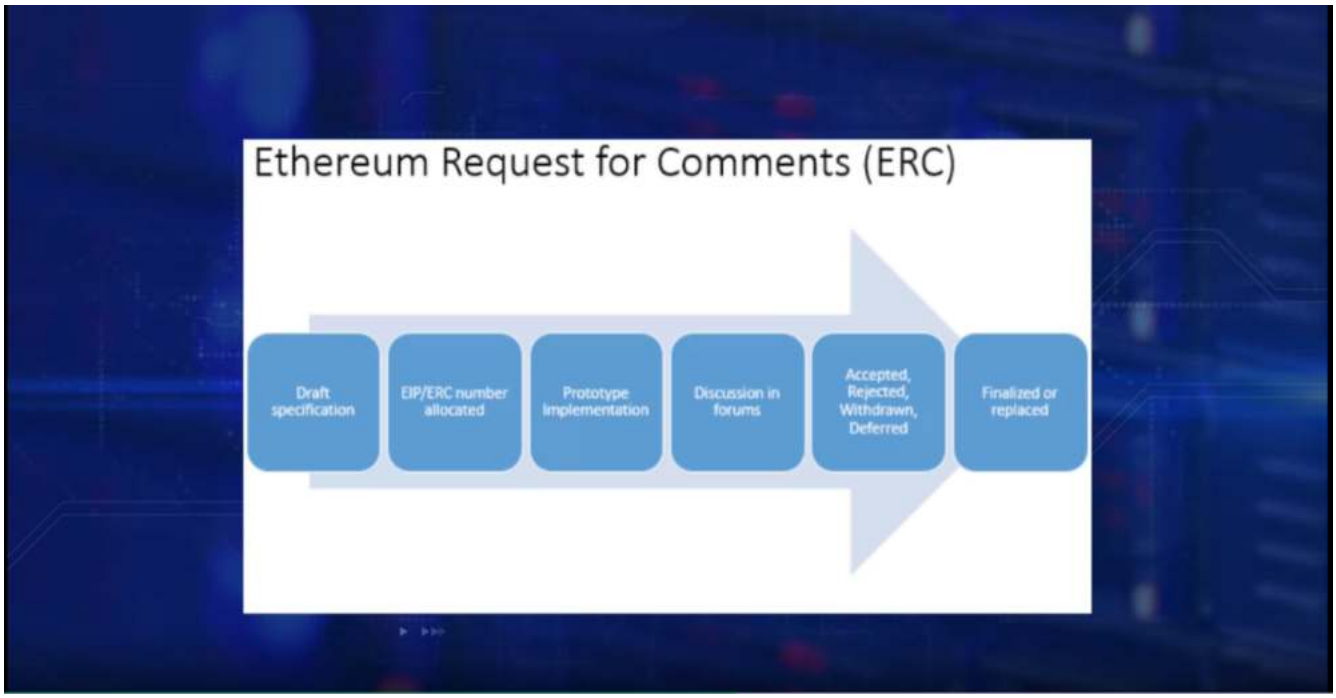
4.1.2 Part 2

The next Dapp model is decentralized marketplace. This is an easy, low hanging fruit of a Dapp. We define marketplace broadly. It could be Dapp for anything from selling goods to education credentialing. Remember, a marketplace facilitates meetings of sellers and buyers. You can build a marketplace platform where decentralized buyers and sellers, those unknown to one another, can meet and exchange items with the payment and transaction recorded on any of the established blockchain technologies such as Bitcoin or Ethereum. You can always use Bitcoin as a payment system for the global marketplace you build.

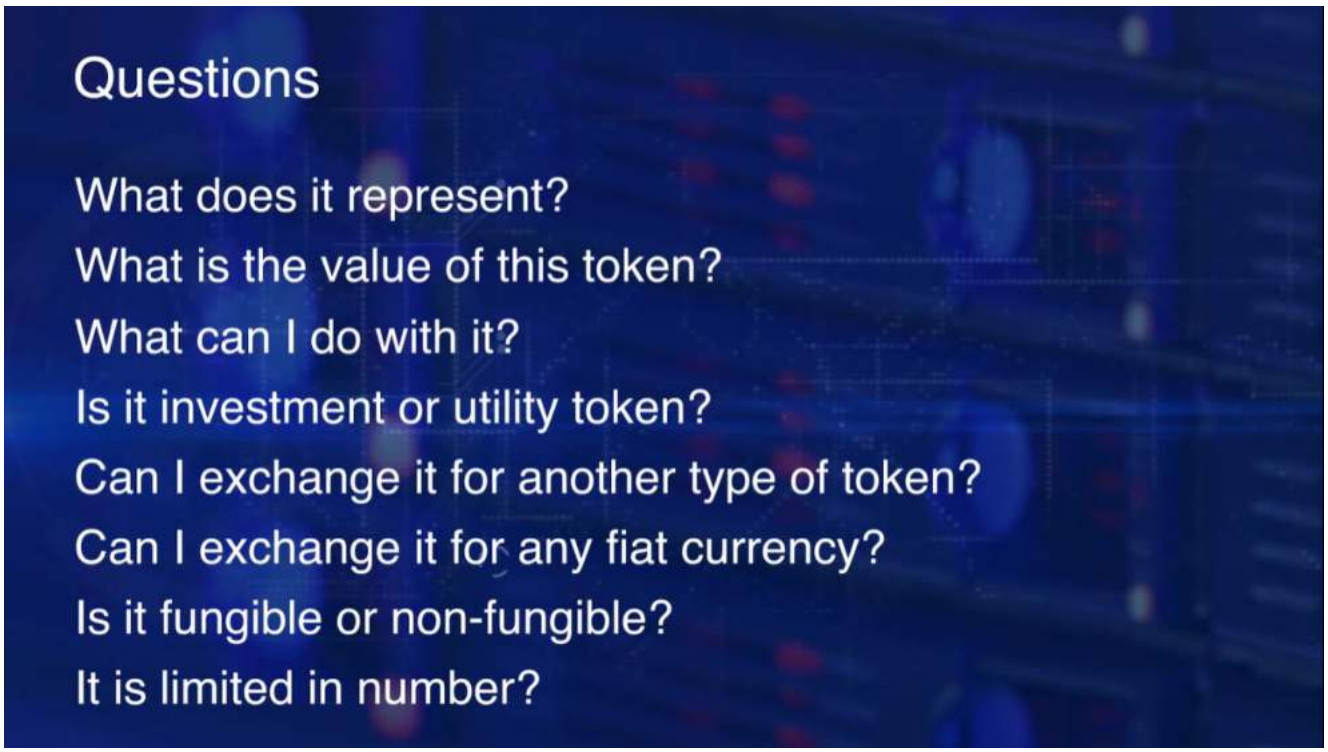


Is there a smart contract code for supporting buying or selling? Yes, there is. An example is safe remote purchase smart contract in purchase.sol in the solidity documentation. One more Dapp to discuss is FinTech.

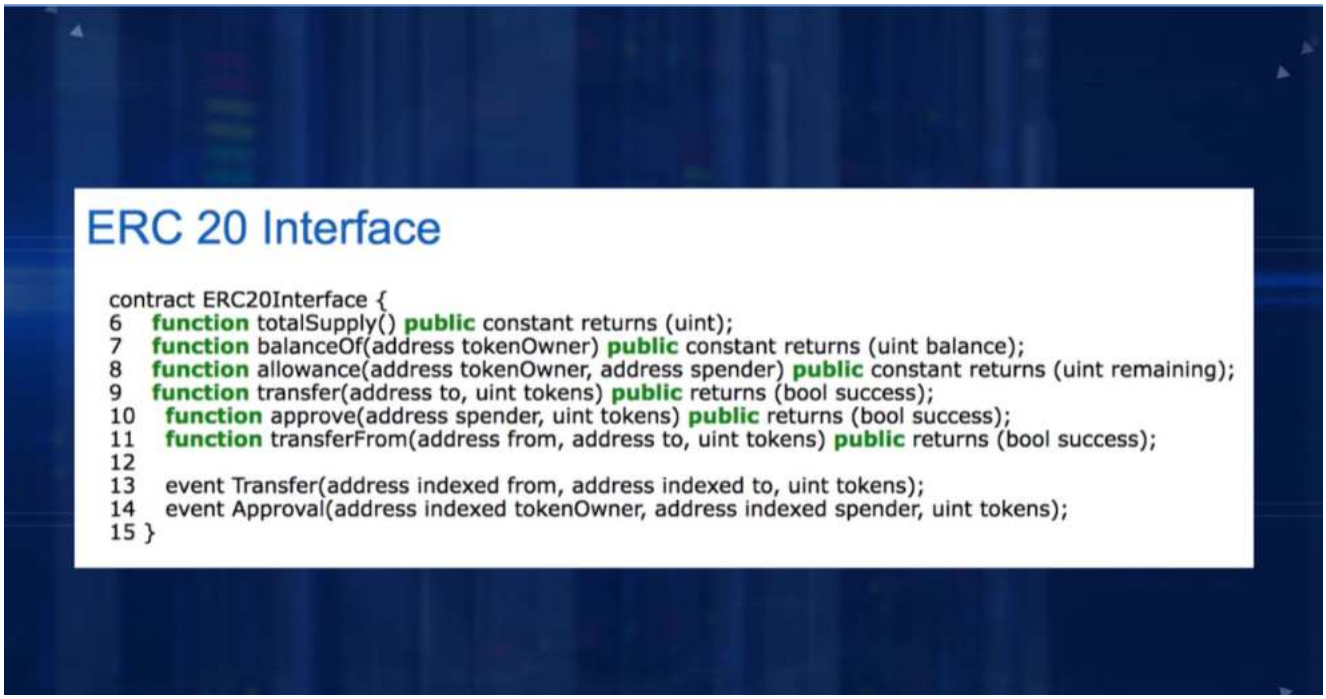
FinTech, short for financial technology, has great potential for innovator Dapps. This domain ranges from decentralized investment instrument to micropayment channels. Currently, a significant issue for FinTech Dapps is the high confirmation times of the transactions compared to the centralized technologies.



At the time of this recording, Dapps cannot meet the sub-second trade confirmation times that the centralized systems can deliver for stock trading.



In this case, scalability is a dominant issue. Privacy is another issue to be considered in this context. Regulation and policies of the US Securities and Exchange Commission, SEC, and the counterpart organization in many countries, have an effect on the FinTech Dapps and their development. Security, privacy, safety, and fraud prevention are major issues.



In fact, tokens, coins, and ICO Dapps provide a backdoor entry for many investors and financial institution into the cryptocurrency markets. In summary, in this lesson, we looked at a few of the Dapp models. We have not covered them all. And many innovative models have yet to be created.

After all, Uber and AirBnB sharing apps came into existence 30 years after the advent of the Internet.

You can use the Dapp models discussed as an inspiration and as a guideline to create a Dapp for your application domain.

4.1.3 Resources

What is An Initial Coin Offering? Raising Millions In Seconds
The Ether Thief

4.1.4 Quiz

1. What kind of Dapp did Bitcoin enable ?

- ◯ Anonymous payment system
- ◯ Dysfunctional payment system
- ◯ Private payment system
- ◯ P2P payment system

2. What is a DAO?

- ○ Distributed Autonomous Organization
- ○ Decentralized Autonomous Organization
- ○ Autonomous Oraclize
- ◯ Decentralized Autonomous Oraclize

3. Is DAO a smart contract?

- ○ False
- ○ True

4. What are ICOs primarily used for ?

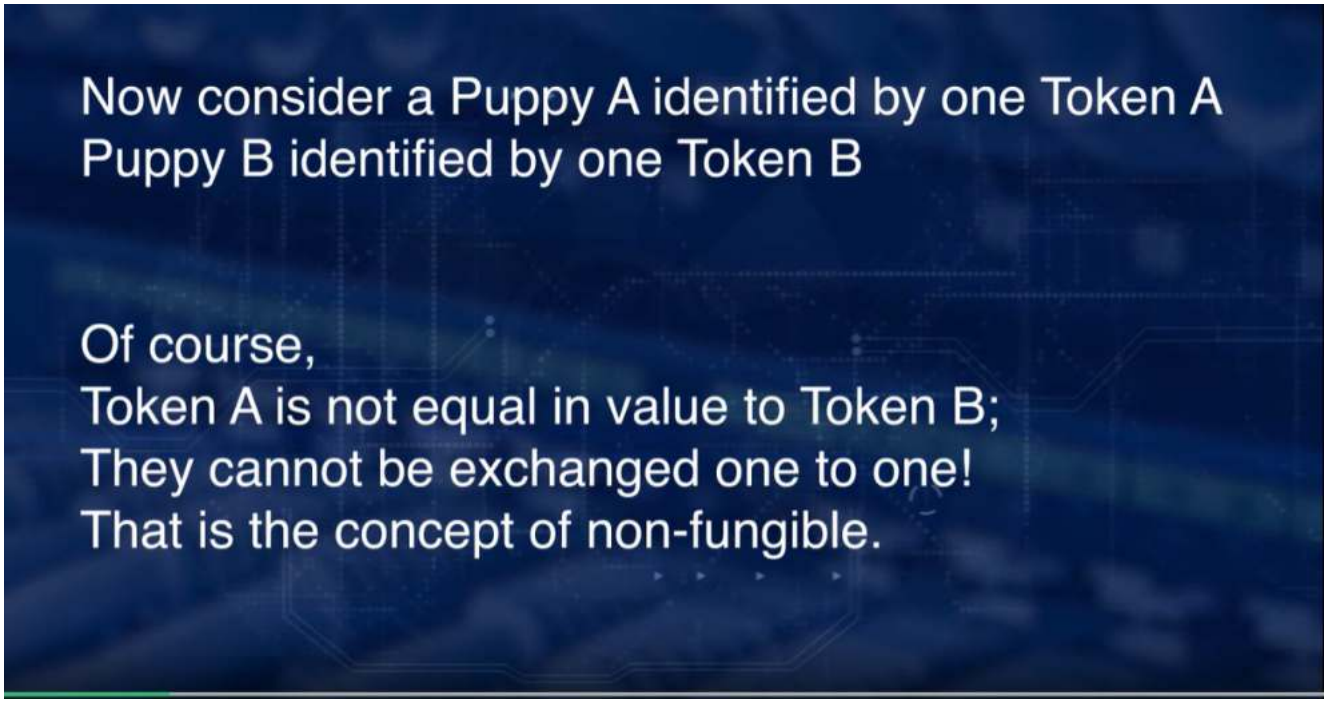
- ○ Trading
- ○ Solving business logics
- ○ Fundraising
- ○ Comparing different coins

4.2 Dapp Standards

4.2.1 Part 1

Our focus in this last lesson is on standards. Anytime the growth of a technology explodes exponentially in many directions, and that too with the deep and broad impact on all walks of life from politics to pet shop, we need to pay attention to bringing some order. This expectation is not unusual. Let's take a look back at the history of computing, when operating systems became a big deal.

Portable Operating System Interface or POSIX standards were introduced for interoperability among different operating systems. IETF, the Internet Engineering Task Force, was established for defining the Internet standards through the Request for Comments, RFCs. Think about it, flights can land in any compliant airport in any country because of those International Standards Organization, or ISO Aviation Standards. Standards bring order, safety, and many more qualitative attributes to the field. Standards can bring regulation and clarity. They are specially imperative for a nascent and high-interest technology such as a block chain.



After this lesson, you will be able to explain the importance of standards to DApps and explain importance of ERC20 and ERC721 tokens. Let’s see how standards are implemented in Ethereum. EIP is an approach to

process improvement in Ethereum ecosystem. EIP or Ethereum Improvement Proposal is a means to manage the protocol specification, improvements, updates, client APIs and contract standards. EIP handles issues at four different categories including; core, or core Ethereum protocol, network, or network level improvement, interface, or interfaces such as ABI and RPC, and ERC, or application level conventions and standards. In this lesson, we are interested in only two ERCs, ERC20 and ERC721 standards. So, what are ERCs? ERC is Ethereum Request for Comments, like RFCs of the Internet. ERC proposal for application level issues are given a proposal ID, thus, become ERCN. Example, ERC20, ERC721, et cetera. The solution draft is proposed by the ERC document and discussed in the community. The ERCs are available on the GitHub and discussed on the guitar and subreddits. The result could be accepted, rejected, withdrawn or deferred. An accepted ERC is finalized and is allocated an EIP number and implemented by the protocol. We'll discuss two of the many ERCs that are making a splash in the DApp run, ERC20 and ERC721. Let's consider coins and tokens. As you might remember, anybody can mint coins and issue tokens. Previously, there were no standards to do this. In the last lesson, we looked at the solidity code for the coin smart contract. You observed that we could easily mint digital coins of any amount we wanted and transfer a coin from one account to another. You experienced how easy it is to issue a coin on your own and transact. You can name your coins something fancy, and deploy it on the Ethereum main chain and issue this digital currency to the whole decentralized world to buy. Anybody with an internet connection can connect and transact using your coin. Given this context, let's consider the issues. What does a token represent? What is the value of this token? What can I do with it? Is it an investment, or utility token? Can I exchange it for another type of token? Can I exchange it for any fiat currency? Is it fungible, or non-fungible? Do you remember non-fungible? Are the tokens limited in number? And many more. These, and many issues are addressed in the standard ERC20 and ERC721. ERC20 is specified as a smart contract interface. It specify a set of rules that allow token DApps to interact with each other to be exchanged with each other, and transact on Ethereum network. The interface looks like this. Given this interface, how do you make a token ERC20 compliant? Of course, you have to implement the functions required by ERC interface. Recall our discussion of interface in the last module. Contract MyToken is ERC20Interface // implement the functions required by ERC20Interface standard // other functions. Can you list some ERC compliant tokens? Augur REP, rep we discussed earlier is a ERC20 token. Here are some ERC20 tokens. You can see many more tokens in this hitscan. We'll know how to distinguish fungible token versus non-fungible tokens.

4.2.2 Part 2

ERC 20 is a fungible token. Dictionary definition of fungible says, able to replace or be replaced by another identical item, mutually interchangeable. One ordinary US dollar bill is equal in value to any other ordinary US dollar bill. One bitcoin is equal in value to any other bitcoin. One ether is equal to any other ether. This is the concept of fungible. Value of this token is intrinsic. Now consider Puppy A, identified by one Puppy Token A. Puppy B, identified by one Puppy Token B. Of course, Puppy Token A is not equal in value to Puppy Token B. They cannot be exchanged one to one. This is the concept of non-fungible. Consider another example. Compare an ordinary penny with a collectible penny. They are one token, or one penny each, physically, but their values are not the same. Another example is Pokemon cards. How about baseball cards? How about real estate? The values differ. In these cases and many more practical examples, a given token value may appreciate or depreciate depending on many factors. This culminates in what is known as the non-fungible token, tokens that cannot be exchanged one-to-one. And ERC 721 is a non-fungible token standard. The non-fungible token concept was brought to prominence by the DApp, CryptoKitties. CryptoKitties is an ethereum ERC-721 compliant tool. It became widely popular, exposing many to the cryptoworld. Of course, since it conforms to ethereum 721 token, it piggybacks on the ethereum blockchain. There was a time it contributed to 30% of the traffic and to the clogging of the Ethereum main net traffic with a backlog of 30,000 transactions. This DApp is hailed as an innovation on many fronts, including but not limited to crypto gaming and crypto collectibles. More importantly, innovation of ERC-721 is in proving that the fixed value for a token may not hold in many practical situations. In this respect, ERC-721 standard has an enormous potential beyond digital cats and collectibles. It has opened up a whole new area of application for DApps. ERC 721 represents just the beginning of many more standards to come. ERC token standards are being proposed for identity, governance, and security. These tokens and EIPs of ethereum are sure to transform Dapp ecosystem into a mainstream application framework, and decentralization into a norm. In summary, we discussed ethereum improvement proposal, and ERC process that are essential for keeping ethereum blockchain updated. We also explored the standards ERC-20 for fungible tokens and ERC-721 for non-fungible tokens. Let's summarize the course as a whole. We opened up the course with a module on essential concepts such as blockchain server, Dapp architecture, and the details of the support APIs. We experience a hands-on,

practical, incremental, end to end development of a Dapp using the Truffle IDE. We then explored important smart contract specific best practices to improve the basic design of a Dapp. We concluded the course with standards that are essential for transforming an emerging technology into a more mature technology. On with this road map of the Dapp development process. You are now ready to chart your path through the emerging decentralized software culture.

4.2.3 Resources

The difference between App Coins and Protocol Tokens
If you don't understand blockchain, maybe these cats can explain it to you
ERC 20(Token Standard)
: ERC 721(Token Standard)
EIP(Ethereum Improvement Proposal)

4.2.4 Quiz

1. How do you make your token ERC 20 compliant?

- ◯ Token smart contract must implement ERC 20 interface functions
- ◯ Use the latest version of solidity compiler
- ◯ Use ERC 20 tokens instead of ethers
- ◯ ERC 20 smart contract must implement Token interface functions

2. Which of the following is true about ERC 20 and ERC 721 tokens?

- ◯ Both are fungible
- ◯ Both are non fungible
- ◯ ERC 20 is non fungible, ERC 721 is fungible
- ◯ ERC 20 is fungible, ERC 721 is non fungible

3. You can use a blockchain token to _____.

- ◯ buy votes to be a chairman
- ◯ enable mining
- ◯ quickly monetize an idea or service

4. Token is a smart contract

- ◯ True
- ◯ False

5. Explain importance of standardization to Dapps.

- ☐ So that they can all be the same
- ☐ So that they can implement certain methods
- ☐ So that they follow certain conventions

6. Which one of the following is ERC 20 compliant ?

- ◯ Exodus
- ◯ Jaxx
- ◯ Coinbase
- ◯ REP

7. On etherscan when you hover over a ERC 20 token such as SALT what message shows up?

- ◯ ERC 721 Token
- ◯ ERC Token Compliant
- ◯ Standard ERC-20 Token