Notes for Project 2 – Dapp Diagram

First, build website using JS, HTML, CSS. We will be running it using node.js (Javascript runtime Environment)

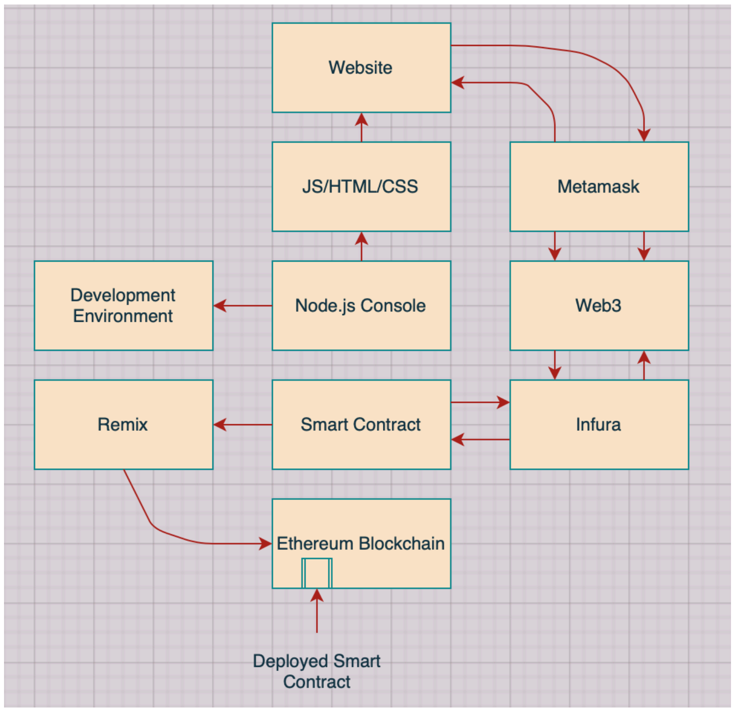
For the front-end section, you can do whatever you like, or use a Framework like React, Angular, Vue, or stick with Vanilla JavaScript. Or build website from scratch.

Two pieces we build into the application to make it a Dapp is:

- Smart Contract:
  - Piece of code I write that determines the logic or interactions the users are allowed to take
- Ethereum Blockchain:
  - Stores the different states that occur from the interaction's users take with the application

Graphic: https://app.diagrams.net

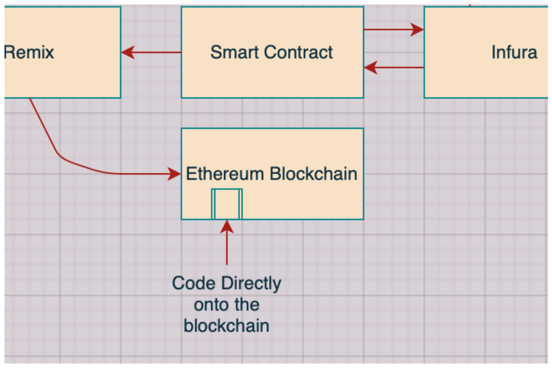
- Down the middle are different layers of the stack
- Down the sides are different tools you can use to help connect the layers



https://app.diagrams.net

While writing my smart contract:

- It’s only a piece of code that the Ethereum blockchain has no idea it exists but what is important to understand is that to deploy your smart contract, you are putting your code directly on the blockchain and then it is a part of the Ethereum Environment.
- Similar to how you can store transactions in a block, you can also push pieces of code directly onto the Ethereum Blockchain.



https://app.diagrams.net

- Once that code is on the Blockchain, it gets assigned a *hash* value and becomes a part of the immutable set of records that make up the blockchain which is important because it’s a huge difference between standard app development and decentralized app development.

- In Dapp development, once you push your code live, it can’t be changed. You can’t fix bugs that come up or change things based on how other people decide to use your code and anybody that wants to can use your code.

Tools:

- In the graphic, Smart Contract does not actually touch Node.js or the Ethereum Blockchain. That’s because node.js can’t run the Smart Contract code. It needs to be compiled using other tools such as Remix
- The Smart Contract sits in Remix (in browser integrated development environment) and gets compiled by their interpreter and then can be deployed to the blockchain from there. (Deploying the contract isn’t directly tied to node.js but you can still bring your contracts into your local environment and work with them from there which will help bridge the gap between front and back-end… more on how to do that later)
- Web3 is the API we will need to interact with the blockchain. Everything that goes onto the Ethereum blockchain is done using a transaction. These transactions are written and read using the Web3 library. (think of it as a Window to access the Ethereum Network).
- Using Web3, you have to ability to search the Ethereum Blockchain, but we don’t have a connection to the Network yet. You’ll need to provide a connection using an endpoint URL.

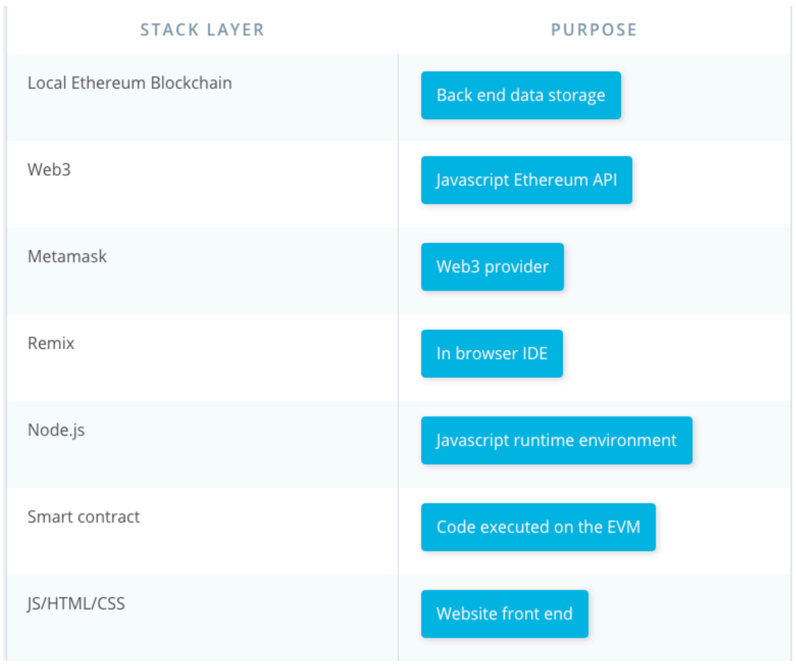
- Infura is a quick and powerful way to make this connection.

    - Infura allows you to connect to the Ethereum blockchain without running a full node. It's a lightweight alternative to downloading the entire blockchain to your local device. It makes the connection that allows you to use the functionality provided by web3.
    - Infura is an API instead of a dev tool which provides secure reliable access to Ethereum and IPFS. It is scalable. IPFS can decentralize the storage much like where Ethereum can decentralize the computation across a network.

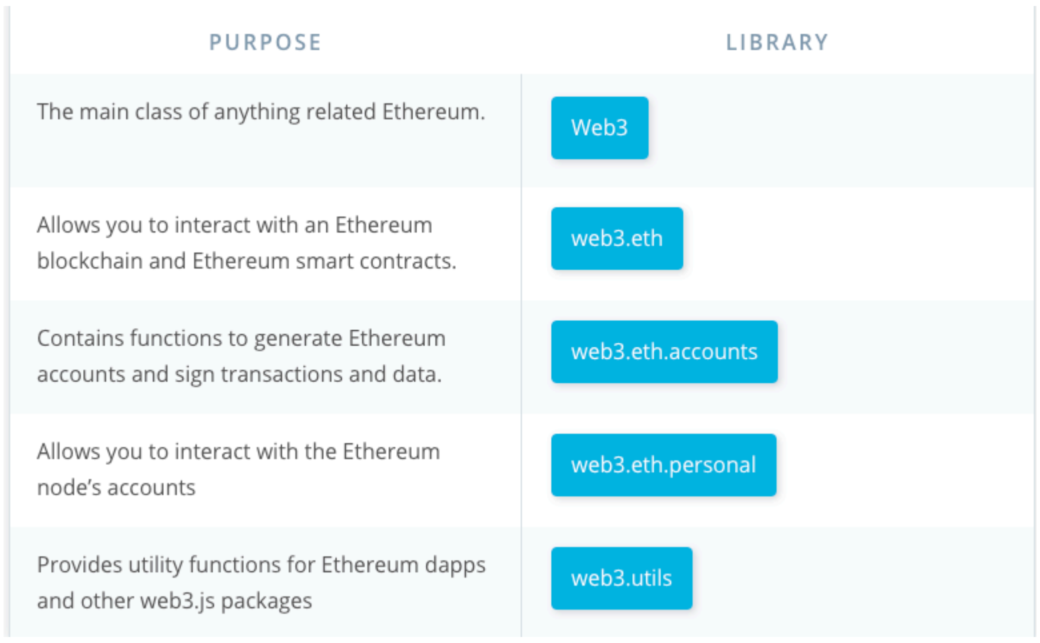
- Metamask is the tool that we use to interact with any existing Dapp. It gives applications access to the funds you store on your wallet so that you can effectively make transactions on the network.
    - Metamask: (Graphic: Connected to both Web3 and Infura)
        - Connected to Web3 because to make this connection to the blockchain for its users, it needs to read and write data to the Ethereum Blockchain.
        - Web3 is used to read and write data, so Metamask uses this to provide this functionality to its users.
        - Metamask also runs on Infura. It uses this to bridge that final connection between Web3 and the Ethereum Blockchain.

From here, the data can climb all the way back up the chain through Infura, Web3, and Metamask again, all the way to the website in order to display this to our users.

Stack Layer and Purpose:



STACK LAYER	PURPOSE
Local Ethereum Blockchain	Back end data storage
Web3	Javascript Ethereum API
Metamask	Web3 provider
Remix	In browser IDE
Node.js	Javascript runtime environment
Smart contract	Code executed on the EVM
JS/HTML/CSS	Website front end



PURPOSE	LIBRARY
The main class of anything related Ethereum.	Web3
Allows you to interact with an Ethereum blockchain and Ethereum smart contracts.	web3.eth
Contains functions to generate Ethereum accounts and sign transactions and data.	web3.eth.accounts
Allows you to interact with the Ethereum node's accounts	web3.eth.personal
Provides utility functions for Ethereum dapps and other web3.js packages	web3.utils

Web3 - Create a project using terminal: (Web3 is the Only way programmatically to connect with the world of Ethereum)

- In the terminal, cd into the directory you want to be in.
- Make a new folder for your Web3 dependency ex: mkdir project2
- cd into this folder
- initialize this project: npm init – Select all default for package.json to get started by pushing enter
- Next, save web3 as dependency for the project by installing and saving: npm install web3 --save
- If you want, type clear

Start Here

- Now open node in terminal window by typing: node
- Web3 class contains all the other modules and web3 is the package.
- Set var to Web3 and then require web3: var Web3 = require('web3');
- Now just call the web3 variable we set: Web3
- This prepares you to connect to the Ethereum Network but Web3 is just an API.

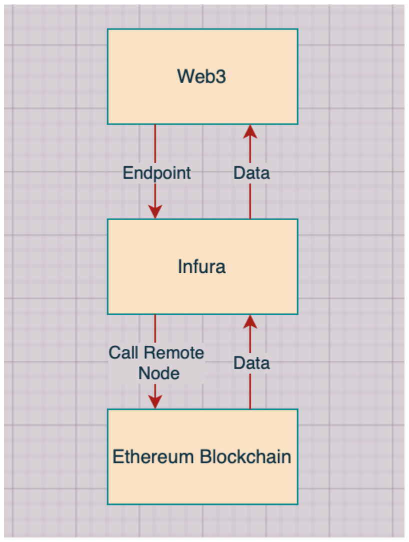
Web3: Web3 is the only programmatic way that can connect any front-end application to the Ethereum world

Unitmap: Denominations:

```
noether: '0',
wei: '1',
kwei: '1000',
Kwei: '1000',
babbage: '1000',
femtoether: '1000',
mwei: '1000000',
Mwei: '1000000',
lovelace: '1000000',
picoether: '1000000',
gwei: '1000000000',
Gwei: '1000000000',
shannon: '1000000000',
nanoether: '1000000000',
nano: '1000000000',
szabo: '1000000000000',
microether: '1000000000000',
micro: '1000000000000',
finney: '1000000000000000',
milliether: '1000000000000000',
milli: '1000000000000000',
ether: '1000000000000000000',
kether: '1000000000000000000000',
grand: '1000000000000000000000',
mether: '1000000000000000000000000',
gether: '1000000000000000000000000000',
tether: '1000000000000000000000000000000'
```

Infura Operation:

- Infura provides a way to allow you to connect to a node remotely through endpoint urls rather than having to download the full local node to your system which will take up a large amount of storage due to the many gigabytes of data.
- To make this happen, they set up endpoint URLs that, when provided as the connection to Web3, gives you access to the network specified which is similar to how you would do in your own local node.
- This makes it possible to have access to a full node without having to actually download the full node to your own computer.
- You will need an internet connection for this to work.
- To make it work, set up an Infura account and get the main network and point that they provide.
- Then in the terminal, connect to Web3 – Follow the steps above in (*Infura – Connecting to the Ethereum Network*)
- Within Web3, a connection is created to the URL endpoint that infura provided.
- From there, make a Web3 call, which uses the Infura connection to reach a remote Ethereum node and that Ethereum node will return the Blockchain data that you are requesting.
- This gives you full access to the entire Ethereum Blockchain.



https://app.diagrams.net

Infura – Connecting to the Ethereum Network:

Rather than downloading a node with GETH, we can use infura which provides a way to connect to a node directly saving space on our computer.

- Infura.io – set up account
- Set up connection endpoint – select the network we want to use i.e. Testnets: Ropsten, Rinkeby, Kovan, Goerli or use: Mainnet (Main Ethereum Network)
- Open your terminal window and navigate to the project2 folder you created from above (follow steps in Web3).
- In your terminal, type: node in order to enter node console
- Create var Web3: var Web3 = require('web3');
- Next, create a variable and set it to the URL endpoint from infura website:

var url = 'https://ropsten.infura.io/v3/a5ce2af95ecc46d2bdde6acec5d4160f'

- Now create a new variable named web3 and set it equal to Web3 that points to the URL: var web3 = new Web3(url);
- Call the variable to see if the connection is working: web3
- Now we can make calls to a remote node that's using the infura API that's connected to our Web3.

Read data on terminal instead of on Etherscan.io:

- Go to https://ropsten.etherscan.io/
- Select an address
- Set a variable named address equal to that address: var address = '0x78a076b8a75de532a79a86bc01356838b3731beb'
- Then type: web3.eth.getBalance(address, (err, bal) => { balance = bal })
- Now type: balance
- Note that balance may show up as wei and on Etherscan it may show in ether.

        - Make sure to select the correct Network on etherscan.io (i.e. Ropsten Network) in order to check the transactions
- If you want to view the balance in wei in terminal, you have to make the conversion yourself using the web3.utils module.

More on the code here: https://web3js.readthedocs.io/en/v1.2.0/web3-eth.html#getbalance
- https://web3js.readthedocs.io/en/v1.2.0/web3-utils.html#fromwei

To change from Wei to Ether:
- Code: web3.utils.fromWei(number [, unit])
- Specify what value you want to change Wei from in the second value i.e. web3.utils.fromWei(balance, 'ether')
- May need to type code in terminal rather than copy and paste.

You can try to get the transaction count by:
- Web3.eth module
- Function named getTransactionCount: web3.eth.getTransactionCount(address [, defaultBlock] [callback])
- You can use the callback function as they show you just like before, but we can do it differently
- We can use .then which is another way to return a promise using JavaScript
- If we use .then to console.log the result, we will get back the transaction count when we first run the command rather than needing to call the variable afterwards, i.e. web3.eth.getTransactionCount(address).then(console.log);

Accounts:

EOA: Externally owned accounts
- Tied to private keys, doesn't hold code, maintains ether balance, can send transactions
- Individuals holding a private key manage EOA's.
- As a user on the ETH network, this is the type of account you would own
    - Contents:
        - Account Balance
        - Transaction count
    - Abilities:
        - Send transactions
        - Initiate/call a smart contract
        - Transfer value from its wallet
- In EOA's there's no code so the code hash field is an empty string

- **Summary:** Tied to private keys, doesn't hold code, maintains ether balance, can send transactions

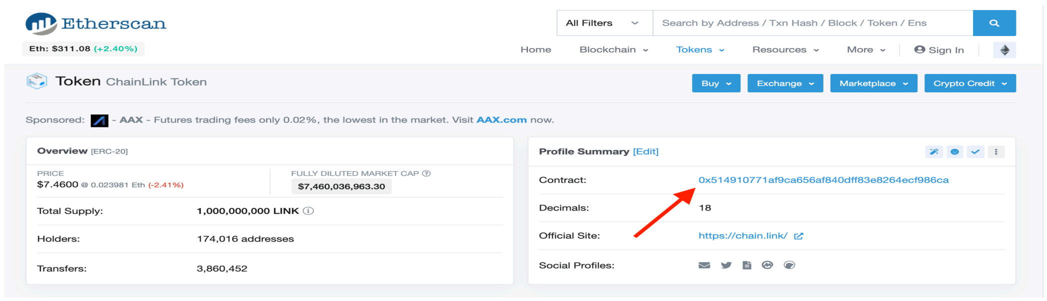
CA: Contract accounts
- Controlled by the code within a Smart Contract
    - Contents:
        - Account balance

            - Transaction count: Transaction count in CA’s refers to the number of times this contract has deployed other Smart Contracts
            - Smart Contract code
        - Abilities
            - Transfer value
            - Initiate/call another Smart Contract
            - Execute Smart Contracts
            - Manipulate storage

- There are a few fields that determine the state of these accounts:
    - **Nonce**
        - Number of transactions on the account
        - Depends on EOA or CA
        - EOA: Number of transactions sent from the accounts address
        - CA: Number of contracts created by the account
    - **Account Balance**
        - Total value of Ether available on the account in Wei (Wei a small denomination of Ether: 1 Eth = 10e18 Wei)
    - **Storage Hash**
        - Root node of the Patricia tree
        - The Patricia tree is the data structure that Ethereum uses
        - This tree stores the hash containing the contents of the account
    - **Code Hash**
        - Hash of the code within the Smart Contract
        - Executes every time a call is made to the contract
        - This hash cannot change
        - This is the reason that Smart Contract code cannot change after it’s deployed
- **Summary:** Has code, maintains ether balance, executes code when triggered by transactions or messages

Accounting Contracts – Using Mainnet:

- Get details about the contract by going to Etherscan.io (Make sure to choose Mainnet)
- Navigate to ERC 20 Top Tokens under the Nav Tokens Link
- Next, select an ERC20 Token such as Chainlink (LINK)
- Select on the contract address (Top right)



Eth: $311.08 (+2.40%)

Token ChainLink Token

Sponsored: - AAX - Futures trading fees only 0.02%, the lowest in the market. Visit AAX.com now.

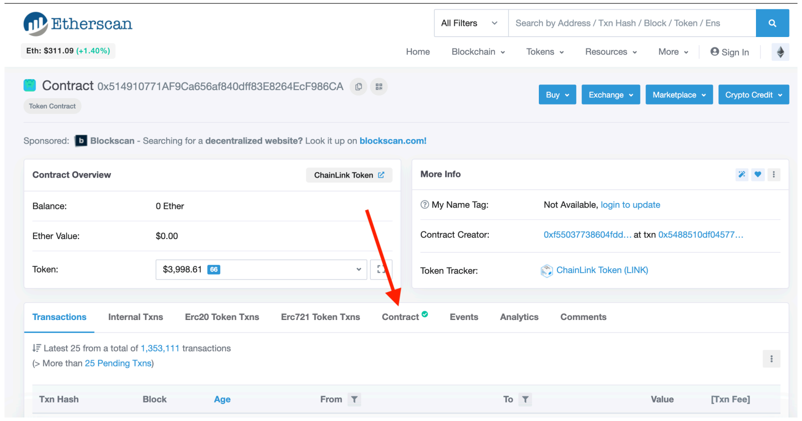
Overview [ERC-20]

PRICE	$7.4600 @ 0.023981 Eth (-2.41%)
FULLY DILUTED MARKET CAP	$7,460,036,963.30
Total Supply:	1,000,000,000 LINK
Holders:	174,016 addresses
Transfers:	3,860,452

Profile Summary [Edit]

Contract:	0x514910771af9ca656af840dff83e8264ecf986ca
Decimals:	18
Official Site:	https://chain.link/
Social Profiles:	

- Next, navigate to the contract code by selecting Contract:



Eth: $311.09 (+1.40%)

Contract 0x514910771AF9Ca656af840dff83E8264EcF986CA

Sponsored: Blockscan - Searching for a decentralized website? Look it up on blockscan.com!

Contract Overview

Balance:	0 Ether
Ether Value:	$0.00
Token:	$3,998.61 66

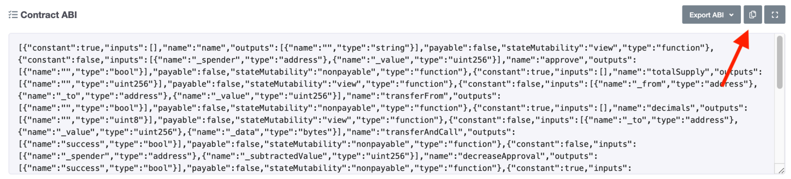
More Info

My Name Tag:	Not Available, login to update
Contract Creator:	0xf55037738604fdd... at txn 0x5488510df04577...
Token Tracker:	ChainLink Token (LINK)

Latest 25 from a total of 1,353,111 transactions
(> More than 25 Pending Txns)

Txn Hash	Block	Age	From	To	Value	[Txn Fee]

- Scroll down to Contract ABI and copy it buy pushing the copy icon



```
[{"constant":true,"inputs":[],"name":"name","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"},
{"constant":false,"inputs":[{"name":"_spender","type":"address"},{"name":"_value","type":"uint256"}],"name":"approve","outputs":
[{"name":"","type":"bool"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name":"totalSupply","outputs":
[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"name":"_from","type":"address"},
{"name":"_to","type":"address"},{"name":"_value","type":"uint256"}],"name":"transferFrom","outputs":
[{"name":"","type":"bool"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name":"decimals","outputs":
[{"name":"","type":"uint8"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"name":"_to","type":"address"},
{"name":"_value","type":"uint256"},{"name":"_data","type":"bytes"}],"name":"transferAndCall","outputs":
[{"name":"success","type":"bool"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":false,"inputs":
[{"name":"_spender","type":"address"},{"name":"_subtractedValue","type":"uint256"}],"name":"decreaseApproval","outputs":
[{"name":"success","type":"bool"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":
```

- The **ABI** is the **Application Binary Interface**
- In terminal, create a variable abi: var abi = past abi copied from Etherscan here
- Now go back to Etherscan and copy the Contract Address
- Back in the terminal, create a new variable called contractAddress: var contractAddress = ‘0x514910771AF9Ca656af840dff83E8264EcF986CA’
- Now we can interact with the contract since we have both of these variables
- We can find a function in the Web3.eth module that allows us to read the contents of the contract
- Pass in the variables we created as shown in the documentation: var contract = new web3.eth.Contract(abi, contract_address)
- Now we can see the details of the contract by typing: contract
- To see all of the methods in the contract, we type: contract.methods
- This will give us back a list of all methods we can call that have been written into the contract
- Try calling the name of the token: contract.methods.name().call((err, result) => {console.log(result)})
- In this case, it will return the name of the token: Chainlink Token

Ganache:

- Comes preloaded with ETH in 10 accounts of 100 ETH each. It reduces the amount of time to test your blockchain by deploying your own local blockchain. Allows for testing quickly and efficiently.
- Provides a solution in order to make test transactions between multiple accounts without having to worry about stocking up on test ether or paying for real ether.

- Install ganache in terminal: npm install -g ganache-cli
- Run ganache with cmd: ganache-cli

Connecting Metamask to Ganache and importing Private Keys:

Linked Video: https://www.youtube.com/watch?v=jaTnIeWjAg0

- Note: Instead of writing truffle develop in the terminal as shown in the video, type: ganache-cli
- From there, follow the rest of the steps provided in the video link

Connecting to Local Blockchain – Ganache:

- Create your folder, i.e. cd desktop/Udacity_Videos
- mkdir Ganache_Dev
- cd Ganache_Dev
- npm init -y (-y accepts defaults)
- npm install web3 –save
- node
- Open Ganache GUI
- Open a second terminal and type ganache-cli to open the cli
- Once running go back to the main terminal with node running
- Connections in Ganache and in cli (terminal) have different IP's. Just make sure you're connected to the right URL
- Back in the terminal with node running:
    - > var Web3 = require('web3')
    - > var web3 = new Web3('http://127.0.0.1:7545')
- If all is well, you are now connected to your local Blockchain using web3
- To check, we can call web3
    - > web3
    - Now try to list accounts
        - web3.eth.getAccounts().then(accounts => console.log(accounts));

Connecting to Local Blockchain – Ganache with VSCode:

- In VSCode, navigate to your folder and open it up i.e. Ganache_Dev
- Create new file: index.js
- Paste this code below:

```
var Web3 = require('web3')
var url = 'HTTP://127.0.0.1:7545'
var web3 = new Web3(url)
web3.eth.getAccounts().then(accounts => console.log(accounts));
```

- Open terminal and run: node index.js
- Also try specifying one address:

```
var Web3 = require('web3')
url = 'HTTP://127.0.0.1:7545'
web3 = new Web3(url)
web3.eth.getAccounts().then(accounts => console.log(accounts[5]));
```

Connecting Ganache with Metamask:

- https://www.youtube.com/watch?v=BWslsP6E7XA
- This video also discusses how to specify two parameters for a deterministic mnemonic sentence to Ganache-cli so that each time you run Ganache, you don't lose your accounts or private keys.

Ethereum Network Performance

- https://ethstats.net/
- Will come in handy when we start building and implementing projects

Note: GETH

- https://geth.ethereum.org/
- Allows you to download full node on your local computer. (It will take up a lot of storage)

Connecting remix to Ganache and VSCode:

- Got to your contract in remix: I used google chrome, code should be saved already: http://remix.ethereum.org/#optimize=true&evmVersion=null&version=soljson-v0.4.26+commit.4563c3fc.js
- Ganache is opened, as well as VSCode editor
- Compile the contract to make sure remix is good with your code
- Go to the Run tab and select Web3 provide
- Connect to your Ganache-cli by entering in: HTTP://127.0.0.1:7545 in the Web Provide Endpoint textbox.
- Deploy the contract
- Ganache will show it as a Contract Creation
- In remix you can set and get messages
- In your VSCode editor, create a file called for ex: connect-to-remix.js
- you can copy the code below:

```
// connecting to remix

var Web3 = require('web3')
var web3 = new Web3('HTTP://127.0.0.1:7545')

web3.eth.getTransactionCount('0xb6294Fe48ce5D61C575c530085E5EA02C8138af1').then(console.log);
```

- In terminal run the file: node connect-to-remix.js
- The output should match what Ganache is displaying

Truffle:

- To install truffle, open your terminal and type: npm install -g truffle (-g installs it globally)
- Create a new folder ie: truffleProject
- In your terminal, navigate to the folder you just created: cd truffleProject
- To initialize a truffle project, type: truffle init
- Check out your code editor to see the files created.

- Select your truffle.js file or truffle-config.js file. The link below for configuration will provide you the code that you need or you can manually input the information from the starter code.
- Also, in the truffle.js file, make sure you configure your compilers:

```
-   // Configure your compilers
-     compilers: {
-       solc: {
-         version: "0.5.16",    // Fetch exact version from solc-bin (default:
    truffle's version)
```

- Back in the terminal type: truffle console
- Next type: compile
- Now type: migrate
- In your code editor, you can see the new folder/files created. Migrations.json is your ABI.
- Under your contracts folder, create a new file ie: myMessage.sol
- Past the code we have been using from remix. Note – make sure you are using the correct pragma version. At this date, truffle has been using 0.5.16 so in your myMessage.sol file you want to do the same.
- I found that in my code editor, I had to include memory in the parameter of my functions which is highlighted below.

```
-   pragma solidity ^0.5.16;
-
-   contract Message {
-       string myMessage;
-
-       function setMessage(string memory x) public {
-           myMessage = x;
-       }
-
-       function getMessage() public view returns (string memory) {
-           return myMessage;
-       }
-   }
-
```

- Now again in the terminal, type: compile
- Then type migrate
- You should see Message.json appear in the build/contracts folder.
- https://www.trufflesuite.com/
- https://www.trufflesuite.com/docs/truffle/reference/configuration
- https://www.trufflesuite.com/boxes
- https://www.trufflesuite.com/tutorials/pet-shop
- Install Ethereum from terminal: brew install ethereum
- In terminal, you can run the rinkeby network by typing: `geth --rinkeby --syncmode "fast"`

Learning Solidity: Arrays

- https://solidity.readthedocs.io/en/v0.5.3/types.html#arrays
- Array contract ex: https://github.com/udacity/nd1309-work-code/blob/master/Course_Identity_And_Smart_Contracts/Smart_Contracts_With_Solidity/arraysContract.sol

OpenZepplin:

- https://github.com/OpenZeppelin/openzeppelin-contracts
- Choose **contract** folder and then open the **token** folder
- Choose the **ERC20** folder
- Now choose one of the ERC20 contracts

Creating Contract Project:

- Open your terminal
- Make sure truffle is installed on your system: npm install truffle
- Create a folder for your projects: i.e.: mkdir truffle_token
- Navigate to the folder you just created
- Now type: truffle init (installs node module folders and initial contract migrations file and migrations contract)
- Now two packages need to be installed.
- Type: `npm install @truffle/hdwallet-provider`
- Type: npm install openzeppelin-solidity
- OpenZepplin - This is the library with the encode that we can import from for the contracts that you can import all of the different token contracts
- Note: I had to run: `npm i openzeppelin-solidity@2.5.1` in order to for ERC20Details to be called

Steps to run local Etheruem Network:

```
1) Open a Terminal window, and make sure you are inside your
project directory

2) Run the command `truffle develop` (to run a local ethereum
network)

3) Use the command `compile` (to compile your solidity contract
files)

4) Use the command `migrate --reset` (to deploy your contract to
the locally running ethereum network)
```

Deploy contract to Rinkeby Testnet:

- In the truffle.js file, under: networks:{
- Include:

```
-   // Rinkeby Testnet
-       rinkeby: {
```

```
-         provider: () => new HDWalletProvider(mnemonic,
   `https://rinkeby.infura.io/v3/${infuraKey}`),
-         network_id: 4,       // rinkeby's id
-         gas: 4500000,        // rinkeby has a lower block limit than mainnet
-         gasPrice: 10000000000
-       },
```
- },
- This will deploy your contract onto the Rinkeby testnet and then you will receive a contract address. Use this contract address in metamask
- Of course, in truffle.js you want to set a *const mnemonic* phrase to the metamask mnemonic you have attached to your metamask account
- In addition, you can set *const infuraKey* to the infura project ID you receive on infura.io
- In addition, if you play around with your contract, you can always reset and migrate by the following cmd: `truffle migrate --reset --network rinkeby`

Metamask: Add Token:

- Open your metamask toolbar. Add an account if you do not already have one set up.
- At the bottom select: Add Token
- Select Custom Token from the tabs
- From your deployed contract to the Rinkeby testnet, copy the contract address
- Past the contract address in Token Contract Address.
- You should see the Token symbol populate and the number of decimals
- Click next and your token should be added into the Assets portion of Metamask
- You can open another account and follow the same Add Token procedure and then transfer tokens to the new account to test the functionality

Setting up Truffle Project using webpack:

- Again, in terminal if you haven't set up truffle, type: npm install truffle
- Or check if you have truffle installed by typing: truffle -v
- Make a new directory i.e..: mkdir starNotaryv1
- Change directory into the directory you created: cd starNotaryv1
- Type: truffle init
- Then type: `npm install @truffle/hdwallet-provider`
- `Now type:` npm install openzeppelin-solidity or use: npm i openzeppelin-solidity@2.5.1
- This will ensure you have your packages installed. Do these steps before unboxing webpack
- Create your project in terminal by typing: truffle unbox webpack
- In the project folder directory: truffle develop
- Check what URL address truffle started in i.e.: http://127.0.0.1:8545/
- Now to connect metamask, open your metamask from the toolbar and to select a network, create: Custom RPC
- Name the network i.e.: Live Server
- Paste the URL into the New RPC URL and then save it
- Your metamask should now be connected

- In Remix, you can now write your Smart Contract and deploy it.
- Under deployed contracts, you can select the arrow to view the functions your contract has available
- For starNotaryv1 contract, you can press claimStar
- Then click the starOwner and see that the owner address has changed to your account address selected in Remix

Mocha and Chai:

- Both are available as npm packages
- Can be used as both front end and back end development packages for Dapps
- Tests are grouped underneath the describe keyword, and test cases are grouped under the it keyword

Write Smart Contracts in Truffle Project: Move Smart Contract to Truffle Project

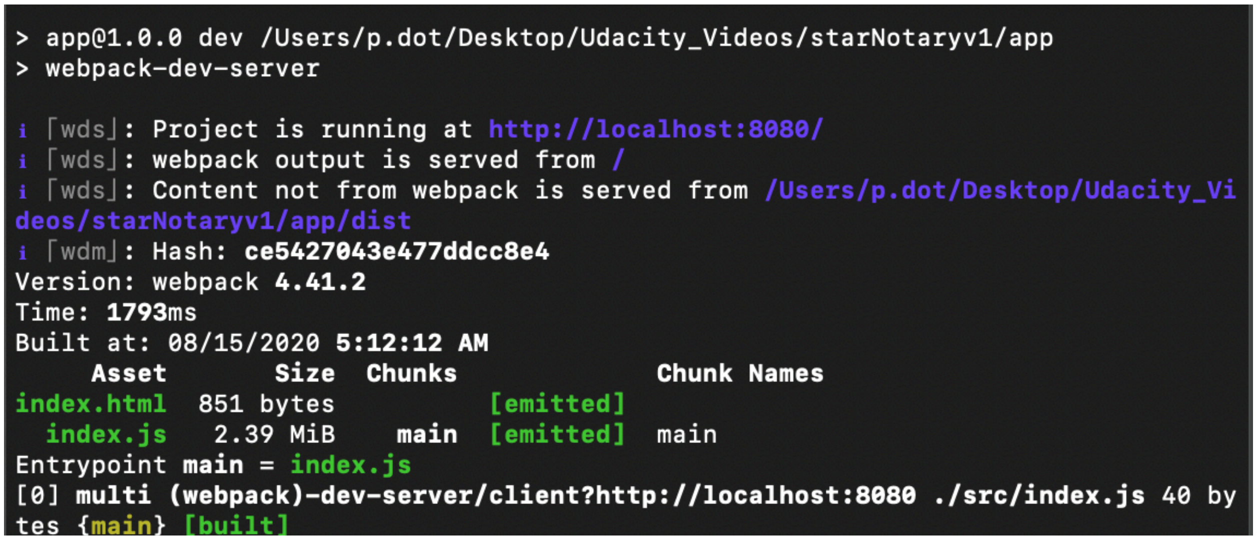
- Clean up boilerplate code that we got from truffle box
- Open your project with your code editor i.e.: VSCode
- In the contract folder, Delete everything except for the Migrations.sol file
- Create a new file in the contracts folder called: StarNotary.sol
- Copy the code from Remix and paste it into StarNotary.sol
- In the migrations file, you will edit 2_deploy_contracts.js to suit your project
- In the test folder, delete metacoins.js and the TestMetacoin.sol file
- Create TestStarNotary.js file in your test folder where you can call functions and test your project

Testing:

- Set up your test using the **it** block in your code editor in your test file.
- Go to your terminal, you should already be in your project directory and in truffle develop.
- Type: compile
- Then type: test. (You should see it pass i.e.: 1 passing). Number of passing depends on how many it blocks you have in your testing file
- The cmd: `migrate --reset` will migrate and reset creating a fresh clean Smart Contact session
- To run the test cases, type: test
- This is the backend development

Open a second terminal window:

- Change directory into your project folder i.e.: cd starNotaryv1
- Chance into your app directory: cd app
- Now you can run your development console which is a front-end by typing: npm run dev
- Now you can check and see the server your project is running on



```
> app@1.0.0 dev /Users/p.dot/Desktop/Udacity_Videos/starNotaryv1/app
> webpack-dev-server

i ｢wds｣: Project is running at http://localhost:8080/
i ｢wds｣: webpack output is served from /
i ｢wds｣: Content not from webpack is served from /Users/p.dot/Desktop/Udacity_Vi
deos/starNotaryv1/app/dist
i ｢wdm｣: Hash: ce5427043e477ddcc8e4
Version: webpack 4.41.2
Time: 1793ms
Built at: 08/15/2020 5:12:12 AM
     Asset       Size  Chunks             Chunk Names
index.html  851 bytes          [emitted]
  index.js   2.39 MiB    main  [emitted]  main
Entrypoint main = index.js
[0] multi (webpack)-dev-server/client?http://localhost:8080 ./src/index.js 40 by
tes {main} [built]
```

- Copy the project server / port: http://localhost:8080/
- Open up your browser and past it into the URL
- Make sure you have your Metamask RPC network selected that we created earlier with one of your addresses imported (Found in **Setting up Truffle Project using webpack** Section)
- Also make sure your truffle.js file is configured correctly as we discussed above
- Note: My Metamask is setup with Google Chrome. When you past the project server/port URL into your browser, use the same browser you have your Metamask set up in
- Reset your account in Metamask if you run into Nonce Errors

For starNotaryv2:

Again, Steps I used: create directory:

- In terminal: mkdir starNotaryv2
- cd starNotaryv2
- truffle init
- npm install @truffle/hdwallet-provider
- npm i openzeppelin-solidity@2.5.1
- truffle unbox webpack
- Cleanup your code and fix 2_deploy_contract.js in migrations folder
- Create StarNotary.sol in contracts folder
- Create TestStarNotary.js file in test folder
- Rename truffle-config.js to truffle.js and fix network server/port
- Create index.html and index.js files in your app/src folder
- Open two terminal windows
- Backend – navigate to starNotaryv2
- Type: truffle develop
- Make sure your port is set in the truffle.js file to the port listed in terminal
- Now: Compile
- Then: test

- Your functions should all pass
- Frontend – Navigate to your starNotaryv2 folder
- cd app
- npm run dev
- copy URL link and paste in a browser (Google Chrome that has Metamask toolbar)
- If you have not connected accounts yet on metamask, do so with private key from backend
- Now test your website