BlockVisors + INMAGINE

BlockVisors + Inmagine Group
Ethereum Dapp + React Development
5- Day Hackathon

Why Blockchain

The promise of a decentralized world.

Less reliance on the middlemen and gatekeepers.

It provides opportunity for a “single version of the truth”

We can verify that something exists or happened

A wakeup call

About Blockvisors

New York based Blockchain consultancy

Token sale Strategy

Development Strategy

Whitepaper Development

Legal + Marketing

Investor stategy

About the Instructor

Dameon Green

17 years Software

Udemy and in Person instructor

Blockchain Developer/ Consultant/Founder

Why This Course?

What are the main goals?

- Quickly up and Running
- Give developers a focused toolkit
- Proof of concept- MVPs
- Push the limits- Test you and test myself.

Course Parts

Theory

Development

Practice

Dive-In

Challenge

Revise

Daily Schedule Breakdown

1. Day 1- How to setup your libraries, wallets and environments? Overview of Ethereum and blockchain topics. Sending and Receiving Ether. How addresses and transactions work in the ecosystem. Review of some projects built on Ethereum.

1. Day 2- How to connect between webapps and blockchain? How to set up project? Truffle deep dive and Solidity/Smart contract intro. Using Remix browser. Going through many smart contract examples. Writing a Smart Contract from scratch. Submitting contract to the network in Remix

1. Day 3- Deep dive into Solidity and Smart contracts / How to issue your own token? Creating token and crowd sale. Performing functions on each. Best practices for Smart Contract development. Using different contract libraries and flattening smart contracts.

1. Day 4- More Solidity / Work through 2 Examples of Distributed Apps. Understanding Web3 and connecting smart contracts with apps. Testing Apps and Smart Contracts in real world context. Connecting to Testnet and Truffle to test real applications.

1. Day 5- Work through 2 Examples of ERC721 applications / Smart Contract security and best practices. Handling a smart contract audit. Data storage on the blockchain.How to deploy on testnet?
Deploy our token from day3 to the real network.

Pre-requisties

Install of all of these for the proper setup.
Its better to do this on day 1 so that we are ready to go when

- Node.js
- Npm and Nvm
- Truffle and Ganache
- Ethereum/Mist
- React/ React Router
- semantic-ui/ others
- VSCode/ Atom
- Chrome/Metamask

Transactions grouped into blocks that are chained together

Database that is decentralized and immutable.

A timestamped registry of access or events

Decentralized transfer of value or information

Cryptographically hashed data formed into blocks that are appended in a sequence.

How Does Blockchain work?

The Next Web

Avoid double-spending of money

Removal of “Trust”

Single version of truth

Replace intermediaries or make their job more efficient

WEb 3=- internet 3.0

Blockchain’s Promise

Hashing values

Privacy and security

SHA256

Security transmutation

On Cryptography

Cryptography is at the center of the whole thing. Then has conceals identity and add a layer of protection.

Cryptography is the science of converting plain simple text into secret, hidden, meaningful text, and vice-versa. It also helps in transmitting and storing data that cannot be easily deciphered using owned keys

Hashing

Hashing is the process of transforming any input data into fixed length random character data, and it is not possible to regenerate or identify the original data from the resultant string data

No matter the size of input string data, the length of its output is always fixed.

Not mathematically feasible to identify two different input strings that will output the same hash. Similarly, it is not possible to computationally and mathematically find the input from the hash itself.

Even a small modification to input generates a completely different hash,

A Platform for decentralized currencies and applications

a world computer which executes its code on multiple machines (nodes) at once.

The system is deterministic which means that there must be no deviations from the results in between those different machines - otherwise we'd have thousands of different blockchains as they would keep diverging in data.

What is Ethereum

Blockchain and Ethereum architecture

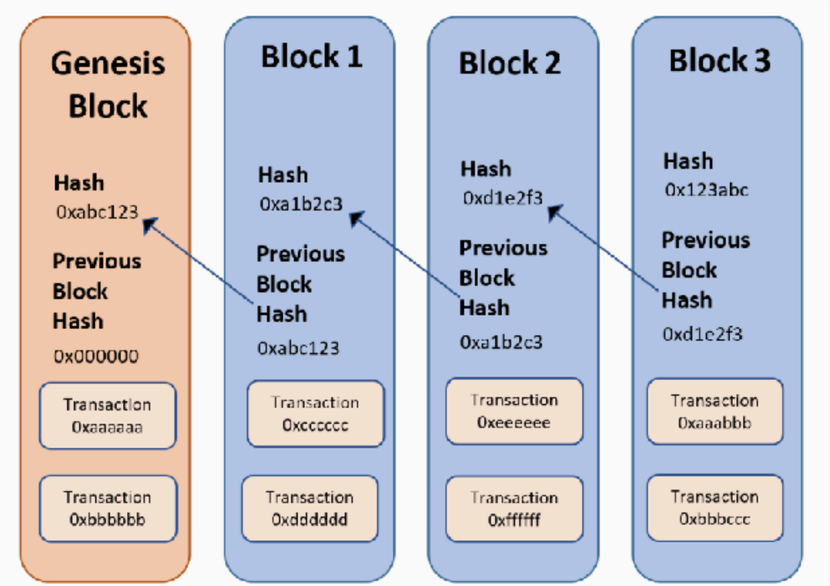
Blockchain is an architecture comprising multiple components and what makes blockchain unique is the way these components function and interact with each other.

Some of the important Ethereum components are Ethereum Virtual Machine (EVM), miner, block, transaction, consensus algorithm, account, smart contract, mining, Ether, and gas]

A blockchain network consists of multiple nodes belonging to miners and some nodes that do not mine but help in execution of smart contracts and transactions.

The Ethereum Architecture

Transactions and blocks



Transactions and Blocks

www.blockviors.net

Mining and nodes

Miners get two types of reward—a reward for writing a block to the chain and cumulative gas and fees from all transactions in the block

Every mining node maintains its own instance of the Ethereum ledger and the ledger is ultimately the same across all miners. It is the miner's job to ensure that their ledger is updated with the latest blocks.

3 miner functions:

Mine or create a new block with a transaction and write the same to the Ethereum ledger
Advertise and send a newly mined block to other miners
Accept new blocks mined by other miners and keep its own ledger instance up-to-date”

Mining

The miner who solves the puzzle first writes the block containing transactions to his own l edger and sends the block and nonce value to other miners for verification.

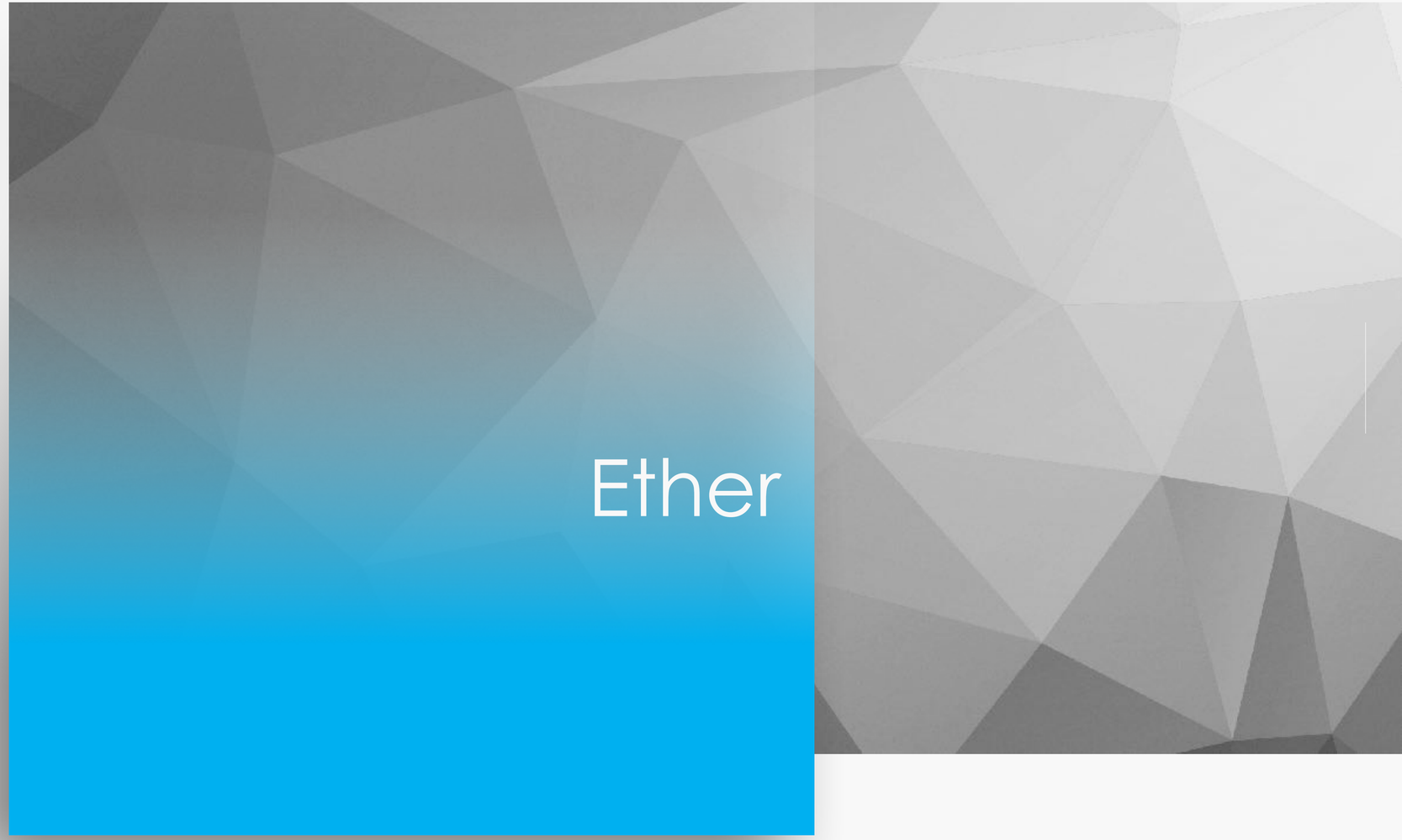
The Currency

Ether is the currency of Ethereum. Every activity on Ethereum that modifies its state costs Ether as a fee, and miners who are successful in generating and writing a block in a chain are also rewarded Ether.

Ether can easily be converted to dollars or other traditional currencies through cryptoexchanges.

Ethereum has a metric system of denominations used as units of Ether. The smallest denomination or base unit of Ether is called wei.
See list of conversion below:

https://github.com/ethereum/web3.js/blob/0.15.0/lib/utils/utils.js#L40:



Runtime environment handling internal stat e and computation.

Every operation on every node

A stack machine of max size 1024 elements for 256 bits

Sandboxed and isolated

External accounts

Contract accounts

The Ethereum Virtual Machine

EtherScan in depth

The blockchain explorer

Deploy, verify, track, watch

Link from Remix

See token data

See deployed code

Link to GitHub

Etherscan

Transactions

All or nothing

cant be altered

Cryptographically signed by sender

Money sent

Operations performed

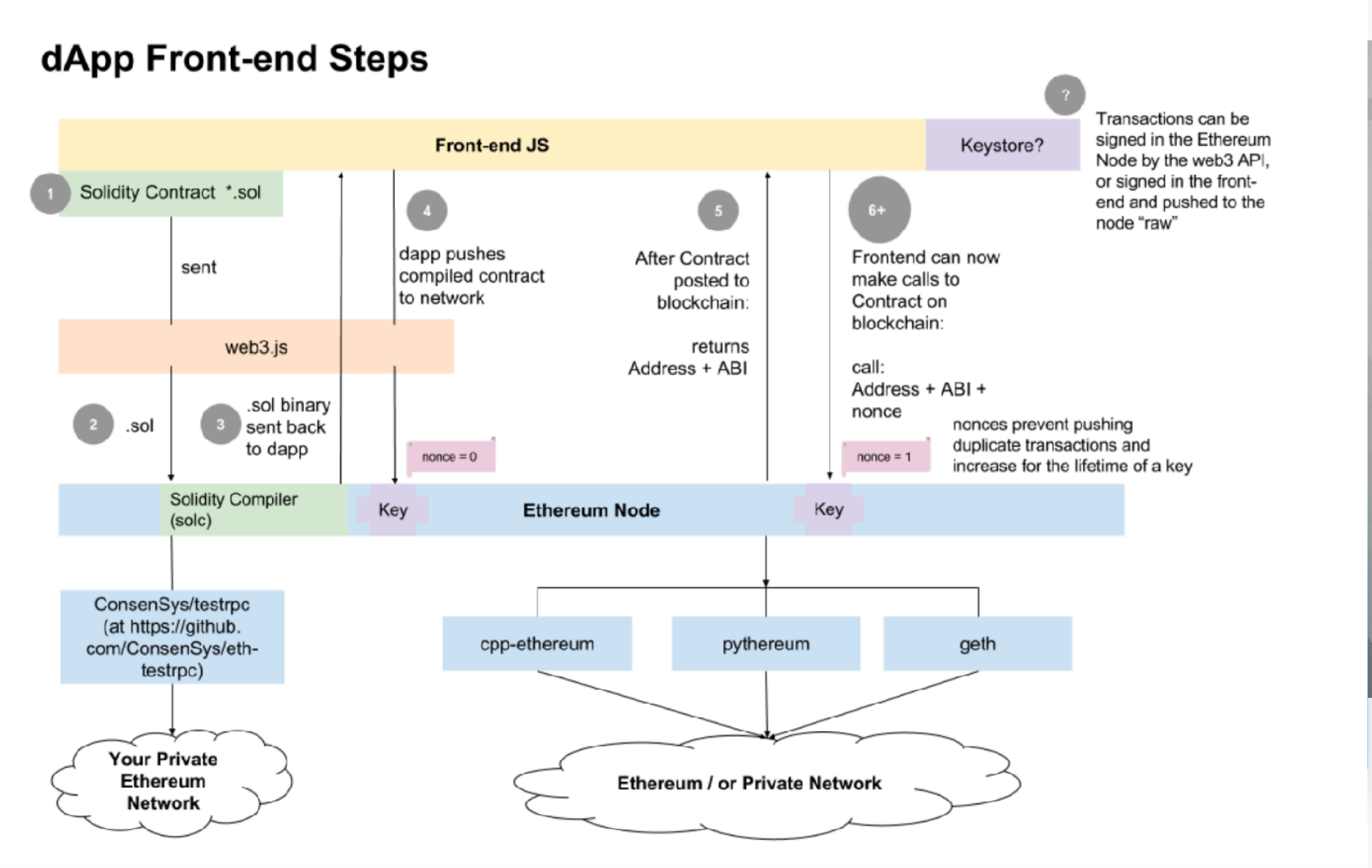
Ethereum Transactions

Trusted Network. The World Decentralized computer

The community has grown considerably in a few years. It continues to be a go- to network for raising funds and proving decentralized application Use Cases.

More than a crypto currency …a platform for innovation

Why Ethereum Ecosystem



How Ethereum Dapps work

Current and Future Uses

State of the Dapp
www.stateofthedapps.com
dappradar.com

CryptoKitties

Tokens

Ampliative Art /Ratio

WeiFund

DNN- decentralized news network

uPort

LeeRoy

Ethereum
What's
Possible

More Dapps

Ethlance

PayNinja

IDEX

Sap-radar

OpenSea

Aragon

dappBoard

ICOs, STOs, Token sales galore!!!

Ethereum What's Possible (continued)

Shortcomings

Transaction fee not feasible for some uses

Blockchain technology is actually more advancing..with things like sharding..etc

Founders have many other projects

Gets negative press with some bad-actor ICOS

Proof of Work probably wont be the around for a long time

Slow transaction speed

Regulatory uncertainty around tokens

Ethereum Shortcomings

Proof of Work

A transaction is sent to the network and waits on confirmation and addition to a block,

Essentially a computation problems is solved and a node is given a reward.

Each complete back is broadcast to all nodes if they check out in order.

If descrepancy- longest chain needed.

The transaction caller is charged a Gas fee…in Ether..which is the reward.

Consensus Algorithm

Public vs. Private

Public by nature

Private for some uses and sensitive data

Consortium or hybrid

Private has many uses and possibility

Private has many uses and possibility

Blockchain Types

Ether powers the Network

The concept of gas deals with the change of state or data being written to the network.

There is Gas price and Gas Limit

Gas limit is the maximum amount of gas we're willing to spend on a transaction.

Gas cost is the GWei price per unit of gas.

total cost is the amount of necessary gas X the price in GWei per gas unit. This is the maximum transaction fee we'll pay - any extra gas is refunded

How does Gas Work

Ether powers the

Operation Name	Gas Cost	Remark
STEP	1	Default amount per execution cycle
STOP	0	Free
SUICIDE	5000	Permanently kill contract
SLOAD	200	Load word from permanent storage
SSTORE	5000	Put word into permanent storage
MLOAD	3	Load word from memory
MSTORE	3	Put word into memory
BALANCE	400	Balance of given account
CREATE	53000	Changed in homestead from 21000
LOG0	375+8*b	Log b bytes of data
SHA3	30+6*w	Encode w words of input

Gas cost is the GWei price per unit of gas.

total cost is the amount of necessary gas X the price in GWei per gas unit.

This is the maximum transaction fee we'll pay - any extra gas is refunded

www.yourcompany.com

Gas Costs

How Gas is Used

Every account has balance in Ether(1 Ether = 10^18 Wei)

Every transaction requires Gas to execute

maxTransCost=gasLimit * gasPrice

gasLimit: Amount of gas purchased by the sender, default value 4712388(need at least 21000 for the trans fee)

gasPrice: Specified by transactor, a criteria

For selection by miner, typically 0.5 GWei and max 50 GWEi = 10^9 Wei

Wei convert to ether…usually a few cents..and more expensive if busy

Transaction execution depletes gas according to rules:

if gas runs out..it reverts

If any is left …it is returned

Message calls to sub-cpontract also o have a gas limit.

If that gas runs out..parent still continues

More on Gas

Accessing Ethereum

0- Olympic- Ethereum public pre-release testnet

1- Frontier, homestead, Metropolis, the Ethereum public main network

1: Classic, the(un(forced public Ethereum Classic main network, chain id 61

1 - Expanse, and alternative Erhereum implementation, chain ID 2

2: Morden, the public Ethereum testnet, no Ethereum Classic testnet

3: Ropsten, the public cross-client Ethereum testnet

4: Rinkeby, the public Geth Ethereum testnet

42: Kovan, the public Parity Ethereum testnet

77: Sokol, the public POA testnet

99: POA, the public Proof of Authority Ethereum network

www.yourcompany.com

Ethereum Networks

Ethereum Wallets

Code

Gui

Hot vs cold

Mobi and mobile wallets

Hardware

Paper wallets

Wallets Overview

Address Types

2 main types

Externally owned accounts controlled by public-private key pairs(usually humans) they have balance and storage

Contract accounts
: have code storage , and are controlled by that code

www.yourcompany.com

Ethereum Addresses

The Wallet and Browser

Your main wallet accounts

Syncing with the network

Sending and Receiving Ether

Using the Test Code

Creating Tokens

Ethereum and Mist

Ethereum Dapp Development

Wallet for Ether and other tokens

Off exchange cold

Decentrallized

Many ways to access

Contribute to sales

View Tokens

How they are created

Private key : 64 hex cars randomly generated

Public key: 128 hex chars generate from private key using the Elliptic Curve Digital signature algorithm

Public address: last 40 har s of 64 char hash of public key

0x denotes it hex

It is created— by generating address from creator address and nonce(number of transactions sent from that address)

www.yourcompany.com

Creating Ethereum Addresses

What are they?

Externally owned account- like what people have to hold ethereuem

These have public and private key

This public key is generally of 256 characters, however, Ethereum uses the first 160 characters to represent the identity of an account.

Contract accounts have function and state

They hold dither

They don’t have a private key..only a public address like the external accounts

www.yourcompany.com

Ethereum Accounts

Transactions

Message from one account to another

Can include binary data (payload) and ether

Can go between Externally owned account and contracts

Eternally owned account can only accept ether

Smart contracts can active and return data(like a function call)

www.yourcompany.com

Ethereum Transactions (general)

Transactions

Transfer of Ether from one account to another: The accounts can be externally owned accounts or contract accounts. Following are the possible cases:

An externally owned account sending Ether to another externally owned account in a transaction

An externally owned account sending Ether to a contract account in a transaction

A contract account sending Ether to another contract account in a transaction

A contract account sending Ether to an externally owned account in a transaction

www.yourcompany.com

Ethereum Transactions (cases)

Transactions

Deployment of a smart contract: An externally owned account can deploy a contract using a transaction in EVM.

Using or invoking a function within a contract: Executing a function in a contract that changes state is considered a transaction in Ethereum. If executing a function does not change a state, it does not require a transaction.

A contract account sending Ether to another contract account in a transaction

www.yourcompany.com

Ethereum Transactions (cases..more)

Transaction Properties

From account property- sends gas or ether..can be external or a contract

To account property- who the ether, benefits, or token goes to

Value- the value of ether sent

Input- the compiled byte code that is deployed

blockhash- this refers tot he hash of the block that was deployed

Gas- amount of gas supplied by the sender

gasPrice- the price per gas that the sender was willing to pay

www.yourcompany.com

Ethereum Transaction Properties

Transactions

Message can contain: source, target, data payload, ether, gas, and return data

Special type of message: a delegate call is executed in the context of the calling function (much like a library or subroutine)

Special operation: sefl-destruct call removes contractfrom the blockchain

Self-destruct removes from being active on blockchain…cant delete it..

www.yourcompany.com

Ethereum Transactio ns(continu

Ethereum Storage and Memory

Storage

Pemanent

Persistent memory inside a contract

Costly to read or modify

Key/value store (each 32B) in space format

Memory

Temporary

Fresh every contract transaction

Expanded by 32B chunks as it is accessed

Cost quadratically grows with access

The Swiss army knife development toolkit

Development and Test framework

10 fake accounts

Fast to test..but no real network

Truffle , Ganache, Drizzle

Ganache and Ganache-CLI

Truffle console

Truffle boxes- scaffolding

Truffle

Boxes

The Truffle Framework is a development environment, testing framework and asset pipeline for Ethereum. It's the most complete set of tools, with the most traction among the Ethereum developers. Truffle's ecosystem includes a number of Truffle boxes — boilerplates that include front-end JavaScript code, Solidity contracts and workflow utilities like a boilerplate webpack project with its toolset — migrations, tests, build pipeline etc. Truffle Boxes can contain entire starter dapps."

Truffle Boxes

Ethereum Dapp Development

Gui based Truffle

Easy to use and setup

Replaces the old test roc

Instantly process transactions

Private setup for testing

Easy to see accounts and link to metamask

Alternatively use Geth

More simulation
Than really but gets you developing fast

Ganache

Using Metamask

Add-on to Chrome

Can switch networks

More secure- encryption

Verity transaction, gas, ether and amount before approve

Use with main and test networks

Makes chrome an ethereum brwhere

Given seed phrase- can restore to other browsers

Allows browser to visit the ethereum network and web3

www.yourcompany.com

Metamask

Getting Started- Geth

Go- Ethereum

acts as an Ethereum client to connect to public and test networks. It is also used to create the mining and EVM (transaction nodes) for private networks..

Geth is a command-line tool written in Go for creating a node and miners on a private chain.

It can be installed on Windows, Linux, and Mac as well.

Ethereum Client

Allows you to get an actual node

Geth

Geth steps

Geth can be downloaded from the https://ethereum.github.io/go-ethereum/downloads/ page. It is available for both 32 and 64 bit machines.

Follow install step then go to command line

Use home-brew if Mac

Brew install geth

Geth account new

It gives you an account

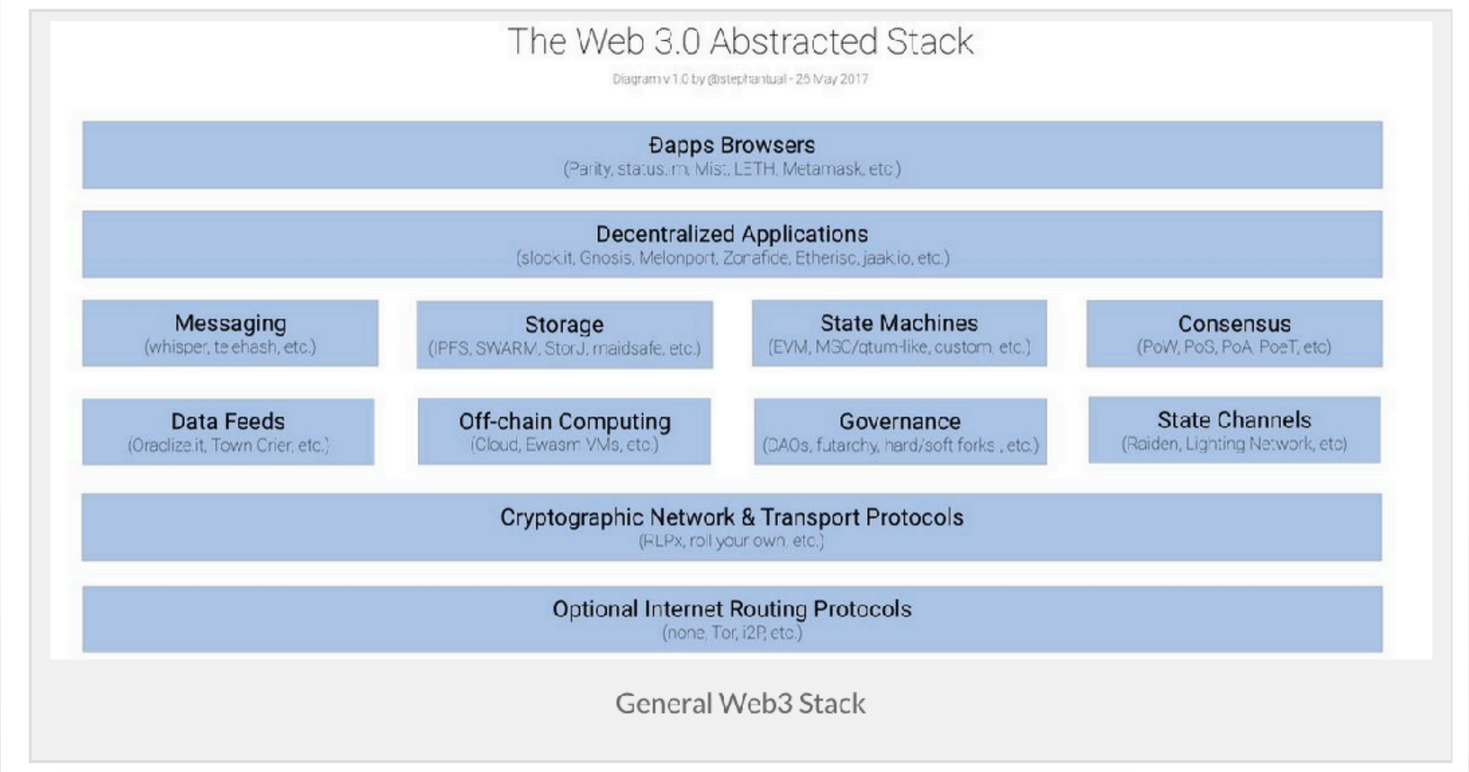
Just type geth..to run the server

What is web3

web3.js - Ethereum JavaScript API.

web3.js is a collection of libraries which allow you to interact with a local or remote ethereum node, using a HTTP or IPC connection.

shifting from centralized to partially decentralized to fully decentralized apps- WEb3 is at the core.



Web3

More on Web3

RPC protocol talks to blockchain..ethereum
A peer to peer set of nodes

Can interact from command line

Web3.eth.Contract-
This is how we interact with our contract

We need to know what kind of interface
and what kind of functions are there

The JSON interface

The contract has a Solidity file that we write
Constract has json representation
ABI

Connecting to Web3

Infura is a remote node—-over son rpc- quickly
Sign up and create a project then get key

Npm install web3

After- open node console
Type:

node:
Var url = “infura link”
Var url = “https://mainnet.infura.io/v3/ad09b34ad0d848aa9cedee653f16b88c"
 Var Web3 = require(‘web3)
Type web3

web3.eth.getBalance(address, (err, bal) => { balance = bal })

Web3

Infura

Container service that gives access to blockchain vis web3

Deploy contracts easily to the tinwork

Create an account

Install hd-wallet provider via npm

Use api key or link in truffle.js

Smart Contracts + Solidity

Ethereum + React Dapps

The Smart Contract

Custom logic and code deployed within blockchain environment like Ethereum

Rules that define interaction between accounts

Transfer and handle digital assets

Basically a small program with function that allow you verify information, facts ,data or balances

Similar to a real world contract in that they are centered around transactions

What is a Smart Contract

Smart Contract development

A program that is converted to bytecode. They interact with the network to add data and change state.

- Develop, debug, and compile
- Dapps give a front-end for interaction
- Allow us to automate
- They come with a cost
- Business logic and rules

- Use oracle for accessing external data
- Tokenize valuable assets
- Record membership or confirm
- Hold in escrow
- Enforce limits or hard rules for things like withdrawal
- Alternative money

Smart Contract Use Cases

Understand its use cases so that we can mintage weakness and play to their strengths.

What makes a good smart contract

Accurate: Executes as intended

Efficient: minimizes cost upon deployment when invoked

Not too much code..or unnecessary loops

Secure: Immune to attacks

Better Smart Contracts

Smart Contract Specifics

written in solidity

Contracts are usually stored in contracts folder

CamelCase or CapWords formatting

End with .sol file extension

Example : myFavoriteContract.sol

Pragma assures that breaking change doesn't break the contract

Creating a Smart Contract

Smart Contract Structure

A contract consists of the following multiple constructs:

State variables- vars declared in a contract but not apart of a function

Structure definitions - custom defined datatypes

Modifier definitions- these modify existing code

Event declarations- fired from other function so that any can watch for an event

Enumeration def:- essentials assigning a var number to a list of values

Function definitions- functions create transaction and alter state

Steps

1. > truffle init
2. Update truffle.js with the development network
3. Create contracts
4. Update 2_deploy_contracts.js referencing the contracts
5. > ganache-cli &
6. > truffle migrate --reset
7. > truffle test

Creating and Testing Smart Contracts

www.yourcompany.com

Creating and Testing Smart Contracts

Steps

1. > truffle init
2. Update truffle.js with the development network
3. Create contracts
4. Update 2_deploy_contracts.js referencing the contracts
5. > ganache-cli &
6. > truffle migrate --reset
7. > truffle test

Set network in truffle.js

```
ubuntu@ip-172-31-15-184:~/solidity-examples$ more truffle.js
module.exports = {
  // See <http://truffleframework.com/docs/advanced/configuration>
  // to customize your Truffle configuration!
    networks: {
        development: {
            host: "localhost",
            port: 8545,
            network_id: "*" // Match any network id
        }
    }
};
```

Testing

Written in mocha or chai framework-see format below

In the test directory…written as a js file

We call it async as it is a unit test framework

```
contract('GetAndSet2TestAsync', async (accounts) => {

        it("should be zero if unset", async () => {
                let instance = await GetAndSet2.deployed();
                let value = await instance.getStoredData(1);
                assert.equal(value.valueOf(), 0, "Unset parameter was not zero");
            });

        it("should store the specified value", async () => {
                var specifiedIdx = 0;
                var specifiedVal = 'howdy';

                let instance = await GetAndSet2.deployed();
                await instance.setStoredData(specifiedIdx, specifiedVal);
                let storedData = await instance.getStoredData(specifiedIdx);
                assert.equal(storedData, specifiedVal, "Set parameter was not returned");
            });

    });
```

Go into truffle console..type test..it compiles and runs test

Setup test to fail and run again

Testing

www.yourcompany.com

Testing

Use the test.folder

CapWords naming

Test + Name of the contract

Test one contract per test

Import test library for helpers

We essentially use a Test contract

```
// test/TestMerchandise.sol
pragma solidity ^0.4.23;

import "truffle/Assert.sol";
import
"truffle/DeployedAddresses.sol";
import "../contracts/Merchandise.sol";

contract TestMerchandise {

}
```

Testing
Best
Practice

www.yourcompany.com

Deployment

Compelling the contract

2 artifacts are generated

ABI definition

Contract bytecode

Think of the Application Binary Interface (ABI) as an interface consisting of all external and public function declarations along with their parameters and return types. The ABI defines the contract and any caller wanting to invoke any contract function can use the ABI to do so.

Bytecode is what represents the contract as it is deployed in the Ethereum ecosystem. Byte is required for deployment and the ABI is need for invoking the functions

This deployment is a contract itself

Contract
Deployment
Technical

Deployment Steps

Deploy to testnet work first always

Change the network information in Truffle.js

Specify whatever network you want

Use Geth- we cant use truffle

Need to setup and unlock for certain length of milliseconds

Parity and kovan fast..but 15 min to sync

1. Update truffle.js with the live test network
2. > geth --rpc --rpcapi "eth,net,web3" console
3. Wait for blockchain sync
4. geth> eth.accounts (if none, use "personal.newAccount()")
5. geth> personal.unlockAccount(web3.eth.coinbase, "password", 15000)
6. > truffle migrate --network live

Make sure to save account password information!

Deploying to Network

The Solidity language

Ethereum Dapp Development

A static-typed Smart contract language

Contract Classes, Functions and conditionals

Inheritance and abstract contracts

Libraries

Types, Arrays, Structs and Mappings

Global Variable

Solidity Basics

Solidity Details

Probably the first “contract-oriented language”(but similar to javascript)

Much like other OO languages

Contains perssistent data in state variable and functions to modify vars

Contains common object-oriented concepts like inheritance, abstract contracts , and interfaces

Calling a function on a different Contract will perform an EVM function call and this switch the context such that the calling contract state vars are inaccessible form the called contract

About Solidity

Solidity Details

Only runs on the blockchain

We access data using this.balance and block.number

Use events to see results..not logging to console

It is immutable…stays on blockchain

Each call and execution costs money

Still new…and changing

www.yourcompany.com

More Solidity

Solidity Details

Starts with Pragma mark which name of the version you wish to compile to

Contracts are similar to classes in structure

Comments like javascript..natspec format

State variable in storage..parameters in memory

Many new keyword: view..etc

Still new…and changing

www.yourcompany.com

More Solidity(cont)

Solidity Documentation

https://solidity.readthedocs.io/en/v0.4.24/

Voting contact

Blind Auctions

Safe remote purchase

Micropayment

Solidity docs

Solidity Sample contract

```
/// @title Simple example of setting and getting storage data
/// @author Dev Name
/// @dev Storage is costly and should only be used for critical data
contract GetAndSet {
  uint16[3] storedData;

  function setStoredData(uint8 n, uint16 x) {
    storedData[n] = x;
  }

  function getStoredData(uint8 n) public view returns (uint16) {
    return storedData[n];
  }
}
```

Still new…and changing

Solidity Sample

Solidity Sample contract

```
/// @title Simple example of setting and getting storage data
/// @author Dev Name
/// @dev Storage is costly and should only be used for critical data
contract GetAndSet {
  uint16[3] storedData;

  function setStoredData(uint8 n, uint16 x) {
    storedData[n] = x;
  }

  function getStoredData(uint8 n) public view returns (uint16) {
    return storedData[n];
  }
}
```

Still new…and changing

Solidity Sample

Solidity Type Declarations

bool: Boolean can be either true or false

int/uint: signed and unsigned integers from 8 to 256 bit. Storage is expensive so u specify Without specification…its assumed the entier 256

fixed/ufixed: signed and unsigned fixed point numbers of various sizes

Address: 160 bit non -arithmetic value. Has member function “balance” “transfer, (send, call, delegate call”)

bytesN: N butes (bytes is alias for bytes1)

Bytes: is a dynamicly-sized array- costs more

String: Dynamically-sized UTF-* encoded string

www.yourcompany.com

Solidity var and function modifiers

Constant: Deprecated for function means var is a constant at compile time

pure: Does not read or change the storage state

view: Like “constant” only reads the storage state

Payable: Can receive Ether through send or transfer, needs fallback function

Visibility

external: can be accessed from other contracts, efficient for large arrays of data

Internal: can only be accessed from inside the contract, default

Public: Can be accessed outside the contract , default

Private: not in derived contract, but information still visible to all external observers

www.yourcompany.com

Solidity Modifiers

Solidity var and function modifiers

Public State Variables

Declaring state variable as public basical adds a getter. For example:

```
uint256 public value1;
```

Makes an implicit function:

```
function value1() returns (uint256) {
    return value1;
}
```

Accessed internal manner: value1

Accessed external manner: this.value1()

www.yourcompany.com

Solidity Modifiers In use

Solidity var and function modifiers

```
enum State { Created, Locked, Inactive }

modifier isOwner() {
   if (msg.sender != owner) { throw; }
   _; // continue executing rest of method body
}

modifier isState(State _state) {
   require(state == _state);
   _;
}

modifier cleanUp() {
  _;  // finish running method body
  // clean up code
}

doSomething() isOwner isState(State.Created) cleanUp {
  // code
}
```

Solidity Modifiers

Modifiers - they can prevent re-entrancy attacks

www.yourcompany.com

Function modifiers

```
/// @title Simple example of setting and getting storage data with owner and se\
lfdestruct
/// @author Dev Name
/// @dev Storage is costly and should only be used for critical data
contract GetAndSet2 {
  address public owner;
  string[2] storedData;

  modifier notDum(string x) {
    require(!compareStrings(x,"Dum"));
    _;
  }

  function setStoredData(uint8 n, string x) notDum(x) {
    storedData[n] = x;
  }

  function getStoredData(uint8 n) public view returns (string) {
    return storedData[n];
  }

  function getOwner() public returns (address) {
-UU-:----F1  GetAndSet2.sol    Top L11    Git:master  (solidity Abbrev) ----------
```

```
function setStoredData(uint8 n, string x) notDum(x) {
  storedData[n] = x;
}

function getStoredData(uint8 n) public view returns (string) {
  return storedData[n];
}

function getOwner() public returns (address) {
  return owner;
}

function GetAndSet2() public {
  owner = msg.sender;
}

function goodbye() public {
  selfdestruct(owner);
}

function compareStrings (string a, string b) view returns (bool) {
U-:----F1  GetAndSet2.sol    41% L34    Git:master  (solidity Abbrev) -----
```

Solidity Function Modifiers

www.yourcompany.com

Modifiers - they can prevent re-entrancy attacks

Error Checking

Dont use “throw” it uses gas

Require

Use “require”- it refunds the reamaing

Use tot validate inputs or state

Can return value

Most often used towards beginning of function

require(msg.sender== owner);

Revert

Refunds remaining gas to caller

Use same as require, but for more complex logic

Can return a vault

if(msg.sender != owner) {revert();}

Assert

Uses gas

User foer owerflow/under flow

should never be reached…and error on the ode

assert(this.balance > totalSupply

Solidity Error Checking

Assert

test for internal errors

Check invariants

Assert that a balance is what it should be
Can return value

Most often used towards beginning of function

Property function code should never assert

Performs a revert for require -style exeptions

Executes an invalid operation for alert-style exceptions

All state changes are reverted in both

The operation wasn't successful…thus it has to unwind everything

Assert is expensive..thus u use it in testing … require stops the code when requirement is not met

Solidity
Assert

Events

Use of EVM logging

Inheritable - contract that inherit other inherit the events

Indexed topics

Can work with client-side listeners

Stored int he blockchain

Example- defining

```
event AddListing(
        address indexed seller,
        string name,
        uint price
);
```

The keyword “indexed” is the topic

Example- using

Must use emit

```
function addItem(string name, string description, uint price) public returns(uint) {
    require(online == true);
    require(price > 0, "Price must be greater than zero");
    uint initialLength = items.length;
    items.push(Item({
        // abbreviated for clarity
    }));
    emit AddListing(msg.sender, name, price);
    assert(items.length == initialLength + 1);
    return items.length + 1;
}
```

Solidity Events

Event transaction log

When you analyze the transaction log…you will see something like this below


```
[
    {
        "from": "0x08970fed061e7747cd9a38d680a601510cb659fb",
        "topic":
"0x271a9d337f855516cc23940a051d5bac633dae3335001c96c1951710c7e44a7a",
        "event": "AddListing",
        "args": {
            "0": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c",
            "1": "Mountain Bike",
            "2": "3",
            "seller": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c",
            "name": "Mountain Bike",
            "price": "3",
            "length": 3
        }
    }
]
```


Notice seller name and price

This is how you log and write out date to EVM

Solidity
Events
Transaction
log

www.yourcompany.com

Killing a contact- Why?

Cease business operations

Extreme security measure

It takes an address and sends remaining ether to a designated target

Self-destruct is the only way to remove from blockchain

Archive notes may keep code and storage indefinitely

Funds sent after self-destruct are lost forever

```
function kill() public {
    require(msg.sender == owner);
    selfdestruct(owner);
}
```

Normal function..set to public…require that the executor of the contract is the owner….also specify that the owner receives the funds.

Solidity Kill Function

Constructor

here we have access to the owner

Runs on creations

It is not mandatory and should not have a lot of code

Define:
Contract MyContract {
address owner;

```
constructor() public {
    owner = msg.sender;
}
```

Here we forever define the contract owner

Now u can do as below

```
function kill() public {
    require(msg.sender == owner);
    selfdestruct(owner);
}
```


Solidity
Contact
Constructor

Inheritance

It is the process of defining multiple contract that are related to each other

Parent child relationship

The contract that is inherited is called the parent contract and the contract that inherits is called the child contract. Similarly, the contract has a parent known as the derived class and the parent contract is known as a base contract.

Solidity supports multiple types of inheritance, including multiple inheritance.

Solidity copies the base contracts into the derived contract and a single contract is created with inheritance. A single address is generated that is shared between contracts in a parent-child relationship.

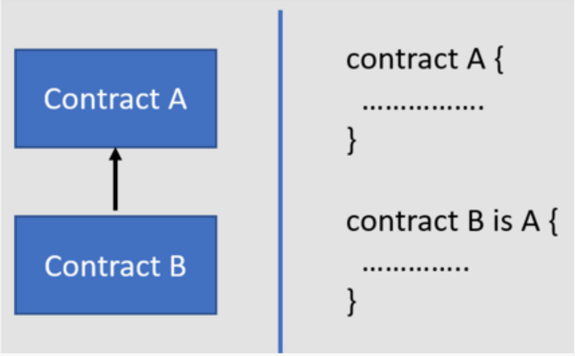
There is a is-a relationship between base and derived contracts and all public and internal scoped functions and state variables are available to derived contracts.

Solidity
Contract
Inheritance

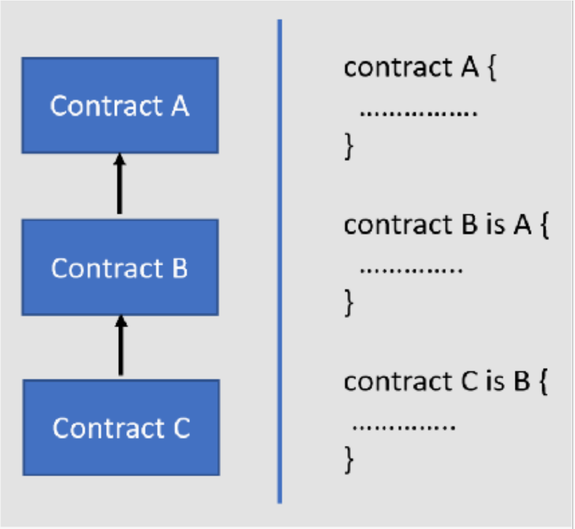
Solidity Contract Inheritance (continued)

Single inheritance

Parent child relationship



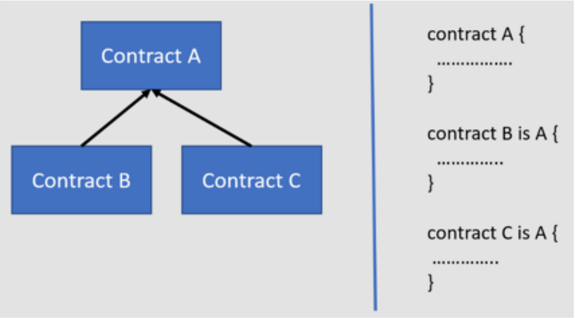
Multi-level inheritance



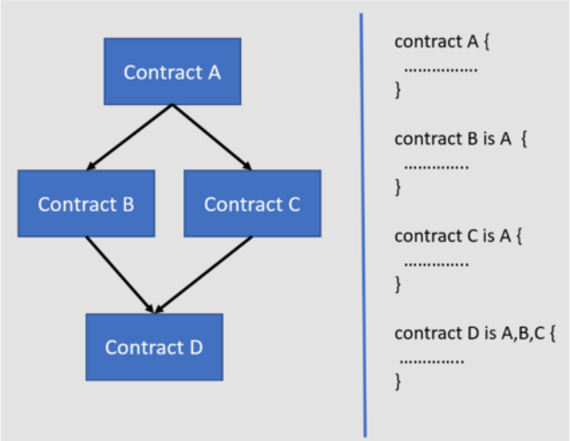
Solidity Contract Inheritance (continued)

Hierachical

Parent child relationship



Multiple inheritance



```
pragma solidity 0.4.19;

contract ParentContract {
    uint internal simpleInteger;

    function SetInteger(uint _value) external {
        simpleInteger =  _value;
    }
}


contract ChildContract is ParentContract  {
    bool private simpleBool;

    function   GetInteger() public view returns (uint) {
        return simpleInteger;
    }

}



contract Client  {
    ChildContract pc = new ChildContract();

    function workWithInheritance() public  returns (uint) {
        pc.SetInteger(100);
        return pc.GetInteger();
    }

}
```

Solidity Contract Inheritance sample code

Overriding methods

Child contract inherits functionality

Child provides its own implementation

```
pragma solidity ^0.4.19;

contract ParentContract {

    uint internal simpleInteger;

    function SetInteger(uint _value) public {
        simpleInteger =  _value;
    }

    function GetInteger() public view returns (uint) {
        return 10;
    }
}

contract ChildContract is ParentContract  {

    function GetInteger() public view returns (uint) {
        return simpleInteger;
    }
}

contract client  {

    ParentContract pc = new ChildContract();

    function workWithInheritance() public returns (uint) {
        pc.SetInteger(100);
        return pc.GetInteger();
    }
}
```

Solidity
Contract
Method
Overriding

www.yourcompany.com

Toggle values for access

Online indicator

require(online == true);

Place in all functions like below

```
contract MyContract {
    address owner;
    bool online;
    function claimFunds() public returns(bool) {
        require(online == true);
    }
}
```

Here we forever define the contract owner

Create parameter up top
bool: online;

```
function setOnline(bool status) public {
    require(msg.sender == owner);
    online = status;
}
```

Other security measures

www.yourcompany.com

Fallback Function

Cannot take arguments but can access mags. Data

Used when just enter is sent to the contract. Need this function to be payable to receive the Ether!

Avoids the default behavior of throwing an exception

Blockchain data

Blockchain data accessible from contract

Now

this.balance, <address> balance

block.number, block.timestamp

Msg.value, msg.sig, msg.data

tx.gaprice, tx.orign

gasleft()

Solidity features

Web IDE for solidity

Can use locally ore in browser

Copy and paste contracts from other side

Run Contracts and functions

Check and compile against various versions

Test deployment to Test or main net

Connect to metamask and other accounts

Simulate with javascript Web3

Remix IDE

Block

block.coinbase (address)- same as etherbase—miners address

Block.difficulty (uint) - Difficutly level of current block
Of type address

block.gasLimit. (uint) - Gas limit for current block

block.number (uint) - Block number in sequence

Block.timestamp (uint) time when the block was created
Msg.sender is the caller of the contract

msg.value is the value in Wei of the contract

msg.data (bytes) information about the function and its parameters

www.yourcompany.com

Transaction and message global vars

Message

Msg.sender is the caller of the contract

msg.value is the value in Wei of the contract

msg.data (bytes) information about the function and its parameters

Msg.gas (uint) - Gas inquest after execution of a transaction

Msg.sig (bytes4) - function identifier using first four bytes after hashing function signature

Now (uint) - current time

www.yourcompany.com

Transaction and message global vars

Debugging

No printing to console

Logging isn't really supported

Sample

Use events

Event myLogMessage(indexed d, uint val)

myLogMessage(msg.sender, quality);

Code client to wat

Var event = myContract.myLogMessagfe();

event.watch(function(error, result){/*some callback*/});

Debugging in Remix IDE

Smart Contract Deep Dive

Ethereum + React Dapps

Fix old code

Search the net work solidity contracts

Some will be old - Update them

Add to them…modifiers..etc

Noticed deprecated features

Get good with practice

Find and Fix Method

Good example

Putting solidity to work

The fallback

Require statements

Variable and declarations

Random number generation

Public vs private access

Modifiers and events

HashMarket example

Solidity Code Example

Use in big projects

The fallback

Require statements

Variable and declarations

Public vs private access

Modifiers and events

Multi-Sig Wallet

Tokens vs Coins

ERC- 20 Token Standard

The primary token

Traded easily on exchanges

Many wallet options

Tried and tested token version

Core functions

Familiarity to investor and platform users

ERC-20 Features

Core Functions

```
// ----------------------------------------------------------------------------
// ERC Token Standard #20 Interface
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
// ----------------------------------------------------------------------------
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

Source: https://theethereum.wiki/w/index.php/ERC20_Token_Standard

ERC-20 Functions

www.yourcompany.com

Utility

Traded for services

Many wallet options

Tried and tested token version

Core functions

Familiarity to investor and platform users

Security

Traded easily on exchanges

Many wallet options

Tried and tested token version

Core functions

Familiarity to investor and platform users

ICOs generally

Major source for fundraising

Tokens generated for sale and pre-sale round to public or private

Security, utility, dividend…or hybrid

Tokens transferred to wallets offline

Traded on exchanges or peer to peer

Initial Coin Offerings and Token Sales

ICO Smart Contracts

The Token

The crowdsale

The functions

Contributing

Wallets and the multi-sig wallet

ICOs- How they work

Token

The Token

The crowdsale

The functions

Contributing

Wallets and the multi-sig wallet

Tokens- Creating our own

Crowdsale

The Token

The crowdsale

The functions

Contributing

Wallets and the multi-sig wallet

Open Zeppelin

Mainly token contracts

Tested and updated

Easy to use

Good practice to reference

Along with Token Market…most commonly used

www.yourcompany.com

Open Zeppelin

Open Zeppelin

ZeppelinOs is an open-source, decentralized platform of tools and services on top of the EVM to develop and manage smart contract applications securely.

In effect, this is a kind of middleware layer on top of EVM, which would be a step further than the current OpenZeppelin framework. What Zeppelin Solutions promise here is enabling developers to opt-in to mutability for their deployed code through upgradeability patterns.

Up until now, in order to “update” a contract one would have to deploy two contracts — one as a proxy with the interface, and the other as the implementation.

Open
Zeppelin
More

Diving into Development

Development Approach

The Command line

Versatile and to the point and heavily used for dependencies

Project Overview and main points

Highlight w hat we need to learn

Practice makes perfect

Certain steps we will repeat because we have to get accustomed to doing them

Think about how we can take thing to the next level

Keep learning

Typical Setup

Navigate to our directory

Create new directory

Run create script or install individual packs

Check our install version and folders

Check our scaffolding

Test run server

Decide.plan of action

Development Setup

The React Library

React Details

Single page- single load

Many setup boxes

Big support from community

Emphasis on handling state..much like solidity

Not as opinionated as other options

Can install and use other libraries

Can use redux for states management

www.yourcompany.com

React and React Routing

Ethereum Dapp Development

React Flow

Component- represent UI pieces

Actions- function

That do thing

Reducer- handle state by watching for changes in events

The then update the state…and the UI

Can use redux for states management

www.yourcompany.com

React Dataflow

Setup details

Command line install

React material UI

Create-react-app

Create-react-dapp

React truffle boxes

www.yourcompany.com

React Setup

Javascript Promises

How they are used

Chaining functionality

Why they are useful

Options and alternatives

Blockchain waits on confirmations

Javascript Promises

React Conversion

Making components

Setting up routes and routing

Handling state

Deciding our blockchain solution

Pluggin into web3

www.yourcompany.com

Converting from Vanilla to React

React Demo Projects

Ethereum + React Dapps

React Portfolio Site

01

Basic React

Get the layout and structure

02

Portfolio

Create a portfolio or directory site

03

Extendable Project

Create something simple that is extendable

React Demo Project

01

Basic React

Get the layout and structure

02

Portfolio

Create a portfolio or directory site

03

Extendable Project

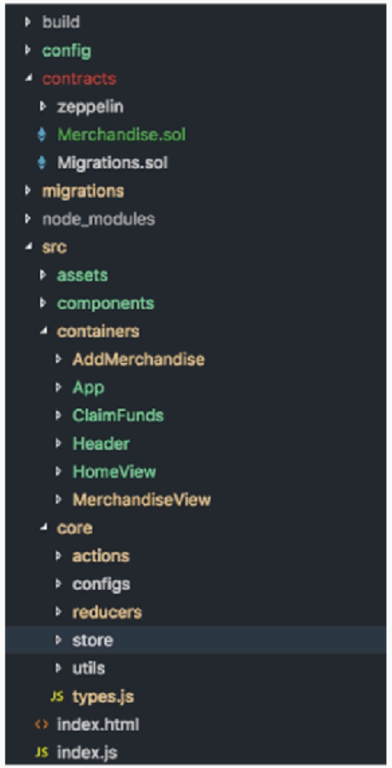
Create something simple that is extendable

React + Ethereum Dapp Demos

Ethereum Dapp Development

Folder Structure

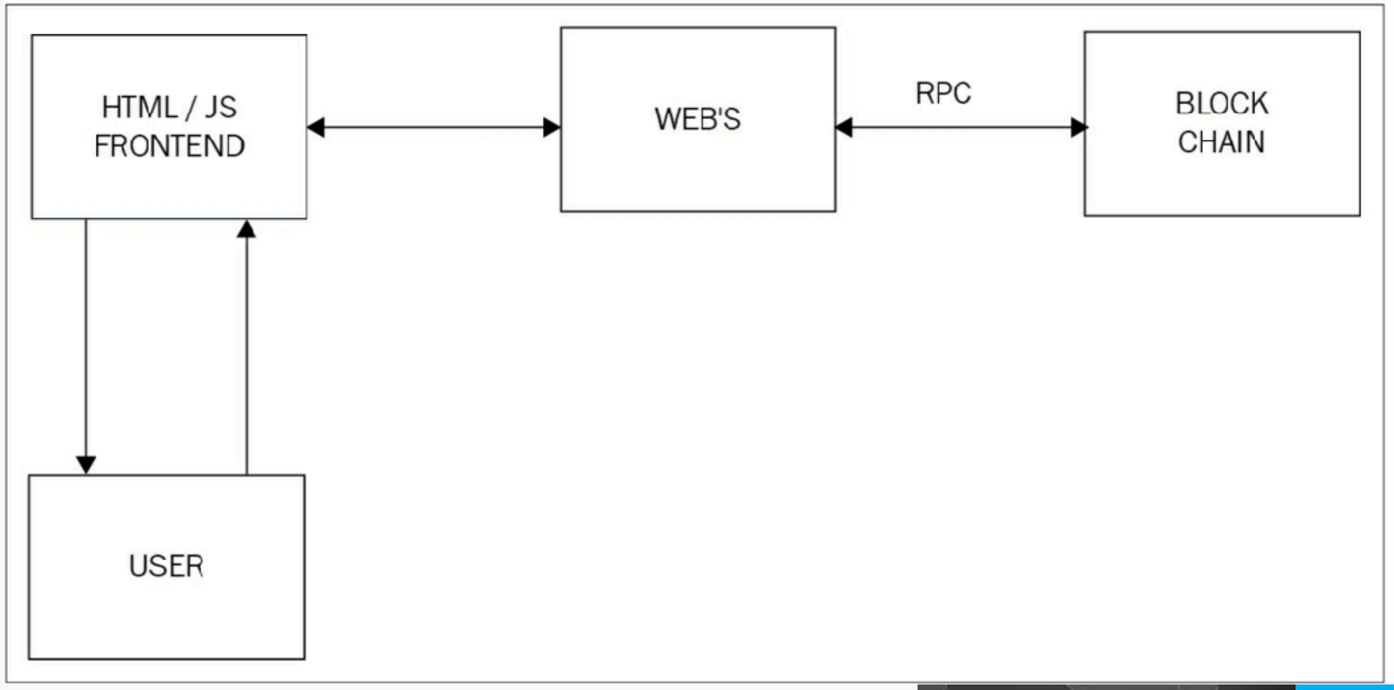
Folder Layout



dApp Project setup

dApp Project High level view

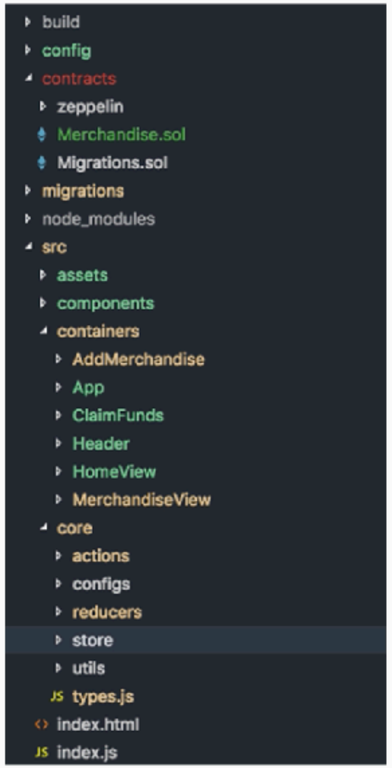
It is desirable to interact with the contracts in a user-friendly manner via a web page. It is possible to interact with the contracts using the web3.js library from HTML/JS/CSS-based web pages. The HTML content can be served using any HTTP web server, whereas web3.js can connect via local **RPC** to the running Ethereum client (`geth`) and provide an interface to the contracts on the blockchain. This architecture can be visualized in the following diagram:



Ethereum Dapp Development

Folder Structure

Folder Layout



www.yourcompany.com

dApp Folder Structure

Ethereum Dapp Development

Election Smart contract Dapp

Use basics of solidity

Handle cases

Use conditions

create basic dapp

Explore how we can extend it

Leverage React

www.yourcompany.com

Voting/ Election Project

Marketplace

List ,buy, sell items on blockchain

Create user friendly UI

Claim funds using metamask

create basic dapp

Explore how we can extend it

Implement authentication

www.yourcompany.com

Marketplace dApp

Item struct

- Seller's address
- Buyer's address
- Item Id

```
struct Item {
    address seller;
    address buyer;
    uint itemId;
    string itemName;
    string itemDesc;
    uint itemPrice;
    bool sold;
    bool shipped;
    bool received;
}
```

www.yourcompany.com

Marketplace dApp- Code

Ethereum Dapp Development

Arrays and Functions

Arrays hold the items

Functions do things

Function myFunc () public {

```
Item[] public items;

function addItem(string name, string description, uint price) public returns(uint) {
    items.push(Item({
        itemId: items.length + 1,
        seller: msg.sender,
        buyer: address(0),
        itemName: name,
        itemDesc: description,
        itemPrice: price,
        sold: false,
        shipped: false,
        received: false
    }));
    return items.length + 1;
}
```

Marketplace dApp- Arrays & Functions

www.yourcompany.com

Ethereum Dapp Development

Arrays and Functions

getItems- gets the items and returns all values

Retrieve from array

View- keyword—we are reading data…not writing anything

WE are returning our struct..essentially as a tuple

```
function getItem(uint itemId) public view returns(
    uint, string, string, uint, bool, bool, bool
) {
    return (
        items[itemId].itemId,
        items[itemId].itemName,
        items[itemId].itemDesc,
        items[itemId].itemPrice,
        items[itemId].sold,
        items[itemId].shipped,
        items[itemId].received
    );
}
```

Marketplace dApp- Arrays & Functions(cont)

www.yourcompany.com

Ethereum Dapp Development

Arrays and Functions

Buy item: here were are buying items

Payable keyword in action

Setting the buyer to the caller of the function==msg.sender

sold- Boolean to true

```
function buyItem(uint itemId) public payable {
    items[itemId].buyer = msg.sender;
    items[itemId].sold = true;
}
```

```
function shipItem(uint itemId) public {
    items[itemId].shipped = true;
}

function receiveItem(uint itemId) public {
    items[itemId].received = true;
}

function claimFunds(uint itemId) public {
    msg.sender.transfer(items[itemId].itemPrice);
}
```

Marketplace dApp- Arrays & Functions(cont

ompany.com

Ethereum Dapp Development

Using require and assert

```
function addItem(string name, string description, uint price) public returns(uint) {
    require(price > 0, “Price must be greater than zero”);
    uint initialLength = items.length;
    items.push(Item({
        itemId: items.length + 1,
        seller: msg.sender,
        buyer: address(0),
        itemName: name,
        itemDesc: description,
        itemPrice: price,
        sold: false,
        shipped: false,
        received: false
    }));
    assert(items.length == initialLength + 1);
    return items.length + 1;
}
```

```
function buyItem(uint itemId) public payable {
    require(msg.value == items[itemId].itemPrice;
    items[itemId].buyer = msg.sender;
    items[itemId].sold = true;
}
```

Require that the item is sold for the price of the particular items

Marketplace dApp- updating with requires and assert

Ethereum Dapp Development

Using require and assert

```
function shipItem(uint itemId) public {
    require(items[itemId].sold == true);
    items[itemId].shipped = true;
}
```

Ship only if the item has been sold

```
function receiveItem(uint itemId) public {
    require(items[itemId].shipped == true);
    require(msg.sender == items[itemId].buyer);
    items[itemId].received = true;
}
```

Needs to be sold and
Marked by the buyer as received

```
function claimFunds(uint itemId) public {
    require(msg.sender == items[itemId].seller);
    require(items[itemId].received == true);
    msg.sender.transfer(items[itemId].itemPrice);
}
```

Only after buyer has confirmed receipt

Marketplace dApp- updating with requires and assert

any.com

Ethereum Dapp Development

Testing

setup before all

```
contract TestMerchandise {

    uint public initialBalance = 10 ether;
    Merchandise merchandise;

    function beforeAll() public {
        merchandise = Merchandise(DeployedAddresses.Merchandise());
    }

}
```

Look at afterAll and afterEach()

It gets complied and run on network..we get details and then the test is run

Isolate dependencies

Run in order

```
function testAddItem() public {
    uint numberItems = merchandise.addItem("Test Item", "Test Description", 4 ether);

    uint expected = 1;

    Assert.equal(numberItems, expected, "It should contain 1 item");
}
```

Marketplace dApp- Testing

Ethereum Dapp Development

More Testing

```
function testGetItem() public {
    uint expectedItemId = 1;
    string memory expectedName = "Test Item";
    string memory expectedDesc = "Test Description";
    uint expectedPrice = 4 ether;

    uint returnItemId;
    string memory returnName;
    string memory returnDesc;
    uint returnPrice;
    bool returnSold;
    bool returnShipped;
    bool returnReceived;

    (returnItemId, returnName, returnDesc, returnPrice, returnSold, returnShipped, returnReceived) = merchandise.getItem(0);

    Assert.equal(returnItemId, expectedItemId, "The ID should be zero");
    Assert.equal(returnName, expectedName, "The name should be Test Item");
    Assert.equal(returnDesc, expectedDesc, "The description should be Test Description");
    Assert.equal(returnPrice, expectedPrice, "The price should be 4 ether");
    Assert.isFalse(returnSold, "The sold status should be false");
    Assert.isFalse(returnShipped, "The shipped status should be false");
    Assert.isFalse(returnReceived, "The received status should be false");
}
```

www.yourcompany.com

Memory says store in memory—-destory after use

Storage- persists…and staying

Notice the assertions and the property mapping

isFalse allows us to save code and use buiilt-in booleans

Marketplace dApp- Testing

Continued

More Testing- buy item

```
function testBuyItem() public {
    bool item = merchandise.buyItem.value(4 ether)(0);

    Assert.isTrue(item, "The function should have returned true");
}
```

Value method with ether..funds specified are from initial funds…has to be less than that total

Include required parameters in buyItem

Go to console and write :

Truffle test

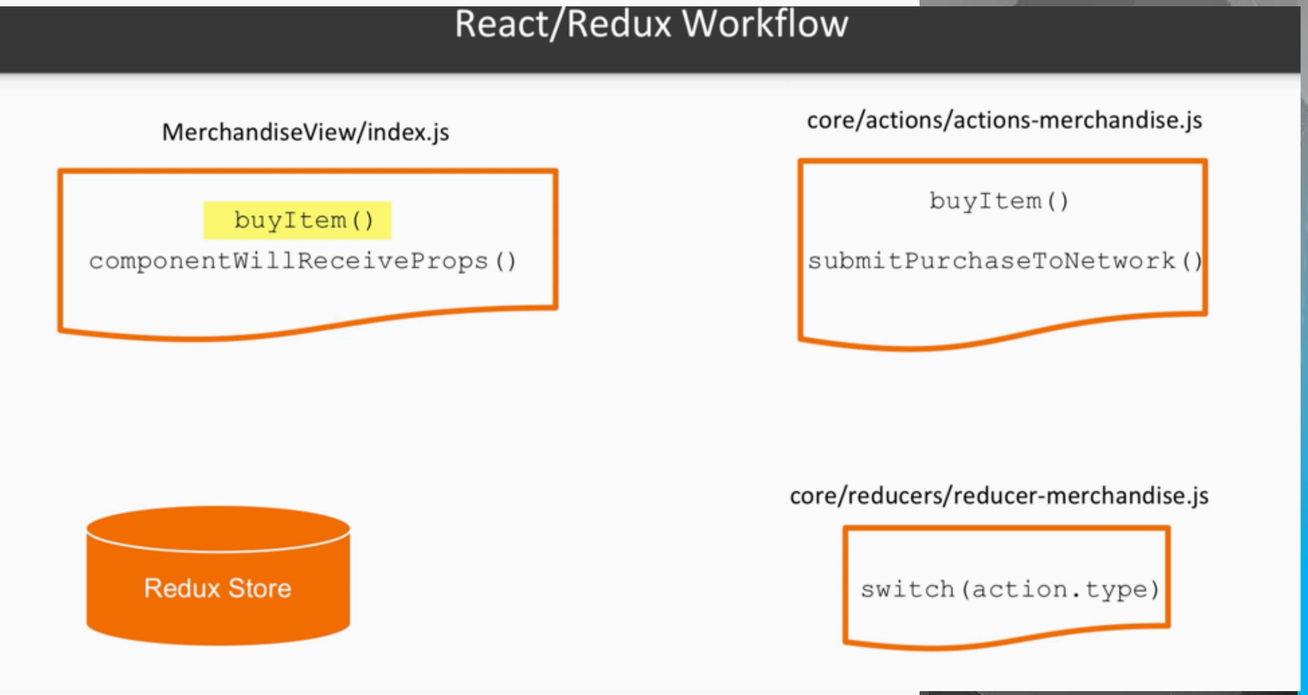
www.yourcompany.com

Marketplace dApp- Testing

Continued

Marketplace dApp- React/ Redux workflow

Front End



Payment app

Internal: only accessible from within the contract and other contracts derived

Private:only accessible within contract

Not accessible in contacts derived from it

External: call from other contrast via transaction

Public : like external but can be called from within contract

www.yourcompany.com

Project: Decentralized Payment App

Todo coin

Create a coin

Transfer it to other addresses as rewards

Use multiple accounts

Using web3 connection

Extend to where we can allow others to transfer

www.yourcompany.com

Project: Todo Coin app

Ethereum Dapp Development

HashMarket Challenge

We have used this contract and know what it does

Build a UI using js

Connect to Web3 and the smart contract and its functions

Manage tranactions

Test the UI and functionality

www.yourcompany.com

Project: HashMarket Challenge

Ethereum Dapp Development

Payroll app

Build our smart contracts

Setup the UI

Test function and Ui

Deploy

Analyze our tranactions

www.yourcompany.com

Project: Payroll app

ERC- 721 Non-Fungible Token Standard

Ethereum + React Dapps

www.blockvisors.net

Fungible vs Non-Fungible

Fungible

Traded for anything

Like currency

Divisible

Most ICOs and Tokens

Non-Fungible

Non-divisible

Can hold specific data

Collectibles

Not traded.on typical exchange

Value usually represents something tangible

Standard Functions

```
  event Transfer(address indexed _from, address indexed _to,
uint256 _tokenId);
  event Approval(address indexed _owner, address indexed
_approved, uint256 _tokenId);
  event ApprovalForAll(address indexed _owner, address indexed
_operator, bool _approved);

  function balanceOf(address _owner) public view returns
(uint256 _balance);
  function ownerOf(uint256 _tokenId) public view returns
(address _owner);
  function exists(uint256 _tokenId) public view returns (bool
_exists);

  function approve(address _to, uint256 _tokenId) public;
  function getApproved(uint256 _tokenId) public view returns
(address _operator);

  function setApprovalForAll(address _operator, bool _approved)
public;
  function isApprovedForAll(address _owner, address _operator)
public view returns (bool);

  function transferFrom(address _from, address _to, uint256
_tokenId) public;
  function safeTransferFrom(address _from, address _to, uint256
_tokenId) public;
  function safeTransferFrom(address _from, address _to, uint256
_tokenId, bytes _data) public;
}
```

www.yourcompany.com

More ERC 721

More ERC 721

Standard Functions

http://erc721.org/
https://github.com/cryppadotta/dotta-license

https://github.com/fulldecent/erc721-example

Sample marketplaces:
https://opensea.io/
https://rarebits.io/
https://www.emoon.io/

https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC721/ERC721BasicToken.sol

www.yourcompany.com

ERC 721 Details

According to standard each is 256 bit positive integer-= the pair of address and unisigned 256 bit integer is unique id

Can represent deeds or valuable assets

Track license and ownership

Internal asset exchanges

Labeling products

Decentralized asset exchange

Game collectibles

Proof of ownership or royalties

www.yourcompany.com

ERC -721 Discussion

Ethereum Dapp Development

Non-Fungible Tokens for Sale

Front end UI

ERC 721 Contract

Ability to Purchase

Differences between ERC20 and ERC 71

Collectibles

Not traded.on typical exchange

Increasing trade value

www.yourcompany.com

Gaia Tree Project

Ethereum Dapp Development

On-demand platform access

Front end UI

ERC 721 Contract

Ability to Purchase

Differences between ERC20 and ERC 71

Collectibles

Not traded.on typical exchange

Increasing trade value

www.yourcompany.com

Non-Fungible Project

Ethereum Dapp Development

Registration - Festival

Front end UI

ERC 721 Contract

Purchase levels of access

Can trade tickets

Increasing trade value

www.yourcompany.com

Non-Fungible Project: Event registration

Ethereum Dapp Development

New standard in progress

New standard for Cryptocomposables

Transferring one 998 means that all the ERC 721 and ERC 20 tokens underneath would also change ownership

My project- Upreal

Example crytptcokitties-its represented by a 721 but it could have tradable accessories like fungible food tokens

Not traded.on typical exchange

Increasing trade value

Collectibles

www.yourcompany.com

ERC 998 Non-Fungible

Storing Data

Computationally expensive

Not the intended purpose

Attaching data…but keeps growing

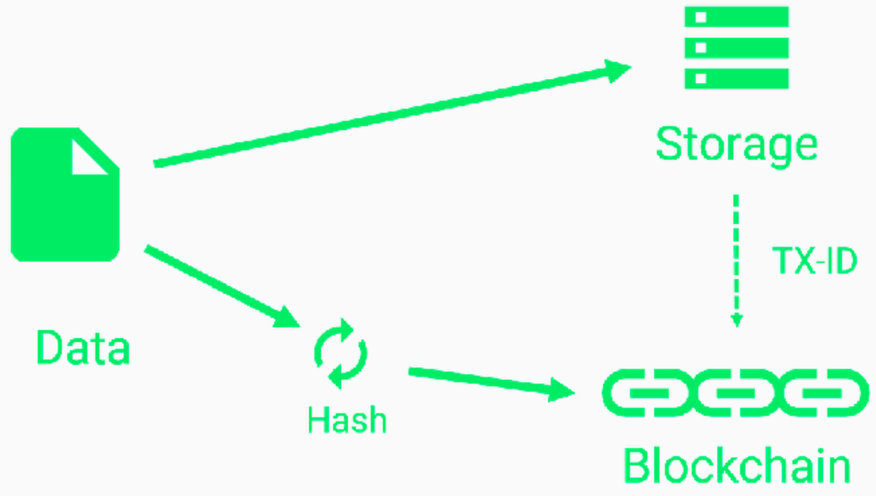
Limited by address size in some cases

Better ways

Mix of blockchain and distributed system

Storing Data on Blockchain

Storing Data



www.yourcompany.com

we assign the id (hash) of the blockchain transaction to our raw data.
In a relational database we would add another column to store the transaction id.

Storing Data on Blockchain

Storing Data

advantages of traditional storage mechanisms (like queries) while getting the tamper-evidence of the blockchain.

We could…in addition..store some subset of the data…

www.yourcompany.com

Storing Data on Blockchain

IFPS

Also, some distributed file-systems like IPFS (Inter-Planetary File System) are meant as a community effort. Just like with blockchain there is a public network where everybody can store their files. The difference here is, that not everybody has to store a copy of the data. A redundancy of x copies is considered enough.

Pros

- Storage for large amounts of data compared to blockchain
- Redundancy of the data
- Transparency
- Decentralized

Cons

- No query capabilites

www.yourcompany.com

IFPS

IFPS

At the time of writing it is approximately 98,225 non-zero bytes of data. This costs between $1.05 and $12.66 depending on the gas price that you set.

There is the block gas limit of currently 3,141,592 gas which can be spent maximum per block.

www.yourcompany.com

Storage on Ethereum

Public Relations Worldwide

Smart Contract Audits

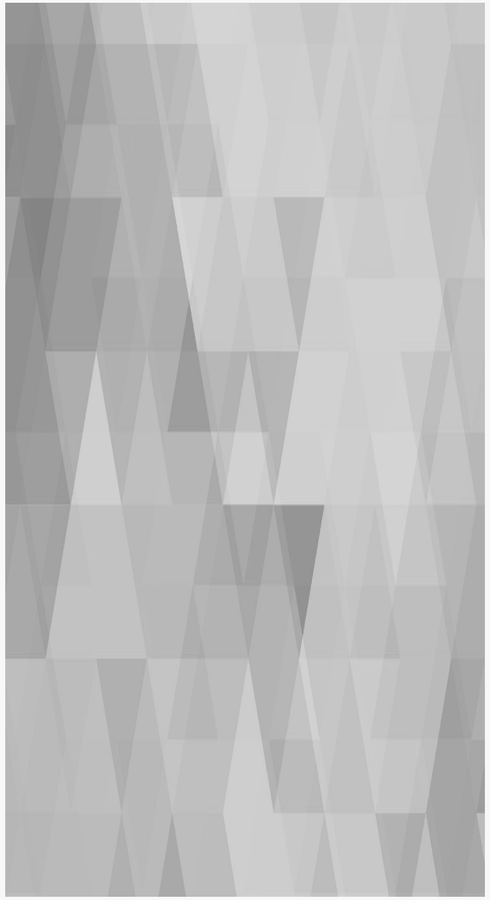
Why Smart Contract Audits

www.yourcompany.com

160

- Leverage the community
- Solve bugs and issues before deployment
- More eyeball and opinions
- Get it right…deployment is final

Smart Contact Attacks



Ethereum React Dapp Development

Source: https://blockgeeks.com/guides/audit-smart-contract/

01

Race Conditions

are a general system's behavior where events do not occur in the intended order. In Smart Contracts, Race Conditions may arise from calling external contracts that take over control flow.

02

Reentrancy

describes one version of Race Conditions whereby some function is called repeatedly before the first function invocation is completed. For example, one of the DAO attack bugs resulted from different invocations of functions interacting in unintended ways. The key solution is to block concurrent calls from happening in certain functions, especially scrutinizing external calls.

03

Cross-Function Race Condition

attack of two functions sharing the same state, with the same solutions. One example is an attacker externally calling transfer() before the user balance has been set to 0, or although the attacker has already received the withdrawal:

04

Transaction ordering Dependence(TOD) / Front Running

Front Running is Timestamp Dependence, so you must scrutinize the uses of the timestamp, especially in cases where transaction time is financially important – such as in a betting contract

Another condition is Integer Overflow and Underflow, a uint value reaching the maximum circles to zero or a uint value less than zero gets set to its maximum.

Ethereum Dapp Development

The Steps for a Full Smart Contract Audit

1.Ensure the Audit Will be Completed on a Deployed Smart Contract

2: Your audit should be performed on a release candidate (RC), or the final Smart Contract.

Provide A Legal Disclaimer:

Explain Who You Are:

Explain Your AuditProcess:

Conduct Attack Vulnerability

Tests: Detail Vulnerabilities Found and

Concerns:

Smart Contract Audit Steps

The Steps for a Full Smart Contract Audit

Analyze Contract Complexity:

Analyze Failure Preparation:

Analyze Code Currency:

Analysis of Re-used Versus Duplicated Code:

Analyze External Calls

Are Untrusted Contracts Marked?

Smart Contract Audit Steps(continued)

Source: https://blockgeeks.com/guides/audit-smart-contract/

The Steps for a Full Smart Contract Audit

Are errors in external calls handled correctly?

Do external calls favor push over pull?

Initial Balance Analysis:

Analyze security of on-chain data:

Analyze N-party Contracts:

Smart Contract Audit Steps(continued)

Source: https://blockgeeks.com/guides/audit-smart-contract/

Solidity Specific Variants

Are Invariants Enforced? Is Integer

Division Conducted?

What happens if Ether is forcibly sent? •
Is tx.origin used?

Timestamp dependence:

Offer Next Steps:

Suggest fixes to the vulnerabilities found and steps moving forward.

Smart Contract Audit Steps(continued)

Source: https://blockgeeks.com/guides/audit-smart-contract/

Sample Audits

Looking at New Alchemy sample

Look at the original contract

What did we change

What was voided

What did we learn

What can we keep as a best practice

Smart Contract Audit Examples

Source: https://blockgeeks.com/guides/audit-smart-contract/

High Profile Ethereum Hacks

Mt Gox

Problems- no good version control
No testing policy

No organizational efficiency, Poor management

The signature data that goes with the input data can be manipulated to look as though the recipient didn't reactive from sender and he gets paid twice.

Not an ethereum project…but a case of poor testing infrastructure and poor organization

Source: https://blockgeeks.com/guides/audit-smart-contract/

High Profile Ethereum Hacks - DAO

DAO Hack

The potential of the DAO and the flexibility, control and complete transparency that it offered was unprecedented, people leaped in to get their share of the pie.

Normally to exit the DAO then they can do so by sending in a request. The splitting function had 2 steps:

- Give the user back his/her Ether in exchange for their DAO tokens.
- Register the transaction in the ledger and update the internal token balance.

What the hacker did was they made a recursive function in the request, so this is how the splitting function went:

- Take the DAO tokens from the user and give them the Ether requested.
- Before they could register the transaction, the recursive function made the code go back and transfer even more Ether for the same DAO tokens.

The origin of the hard fork to Ethereum Classic

Source: https://blockgeeks.com/guides/audit-smart-contract/

www.yourcompany.com

High Profile Ethereum Hacks - Parity

Parity

Knows as the parity muilti-sig frozen funds

Accidental kill of the contract

Parity employed library driven smart contract development for their multi-sig wallets. Meaning, all multi-sig wallets referenced this single library contract for all their functionality. In essence, all Partiy Mulit-Sig wallets had a single point of failure and that address was in the wallet library solidity code:

The user was basically able to initialize the library itself as a wallet, claiming owner rights for it, including the right to kill it altogether.

when the user killed the wallet, they basically killed the library and all the logic functions associated with it. That means, all the wallets that were linked to the library have become useless and ~$150 mil has been frozen up.

Source: https://blockgeeks.com/guides/audit-smart-contract/

www.yourcompany.com

High Profile Ethereum Hacks - Nice Hash

Nice Hash

On December 6th, 2017 at 00:18 GMT, Slovenian based mining company NiceHash was hacked for 4700 BTC which is ~$80 million.

Not a lot of information given about what happened except that an employees information was compromised

Shows how vulnerable a centralized exchange can be

Ironic in this industry

Once you've uploaded your private keys to an exchange to make trading easier, they're at the mercy of that site's security. The sites can be hacked, via social engineering or other means, and the keys can be stolen.

www.yourcompany.com

Hack- a - Thon Projects

Ethereum + React Dapps

Advanced Topics

Reverse Engineering Projects

Ethereum + React Dapps

Other projects

Learn project structure

Become expert

Jumpstart your ideas

State of the Dapp

Help others in open source projects

Github

Reverse Engineering

Getting Work

Need solidity

Find old contracts and debug and upgrade

Learn the included library and interfaces

Keep up with the docs

Need js/backend language - Java, C++, python and solidity

Find remote jobs

Becoming a blockchain developer

Closing Thoughts

Set aside time

Practice and figure
Out what you don't know

Start a personal project…Now

Work in groups

Do the work….it will pay off

Impossible is nothing….challenge yourself

Closing Thoughts

The End

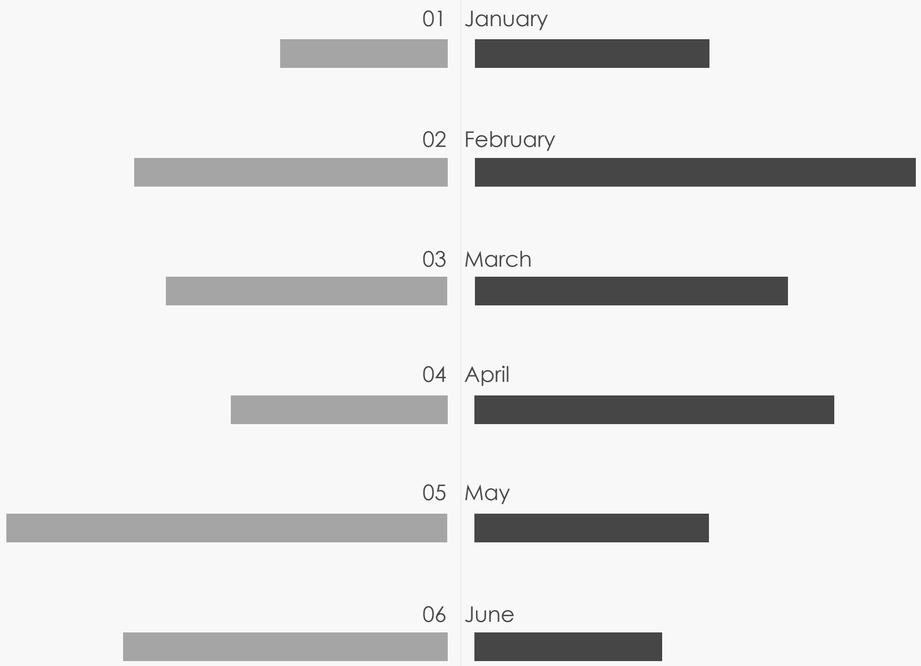
www.yourcompany.com

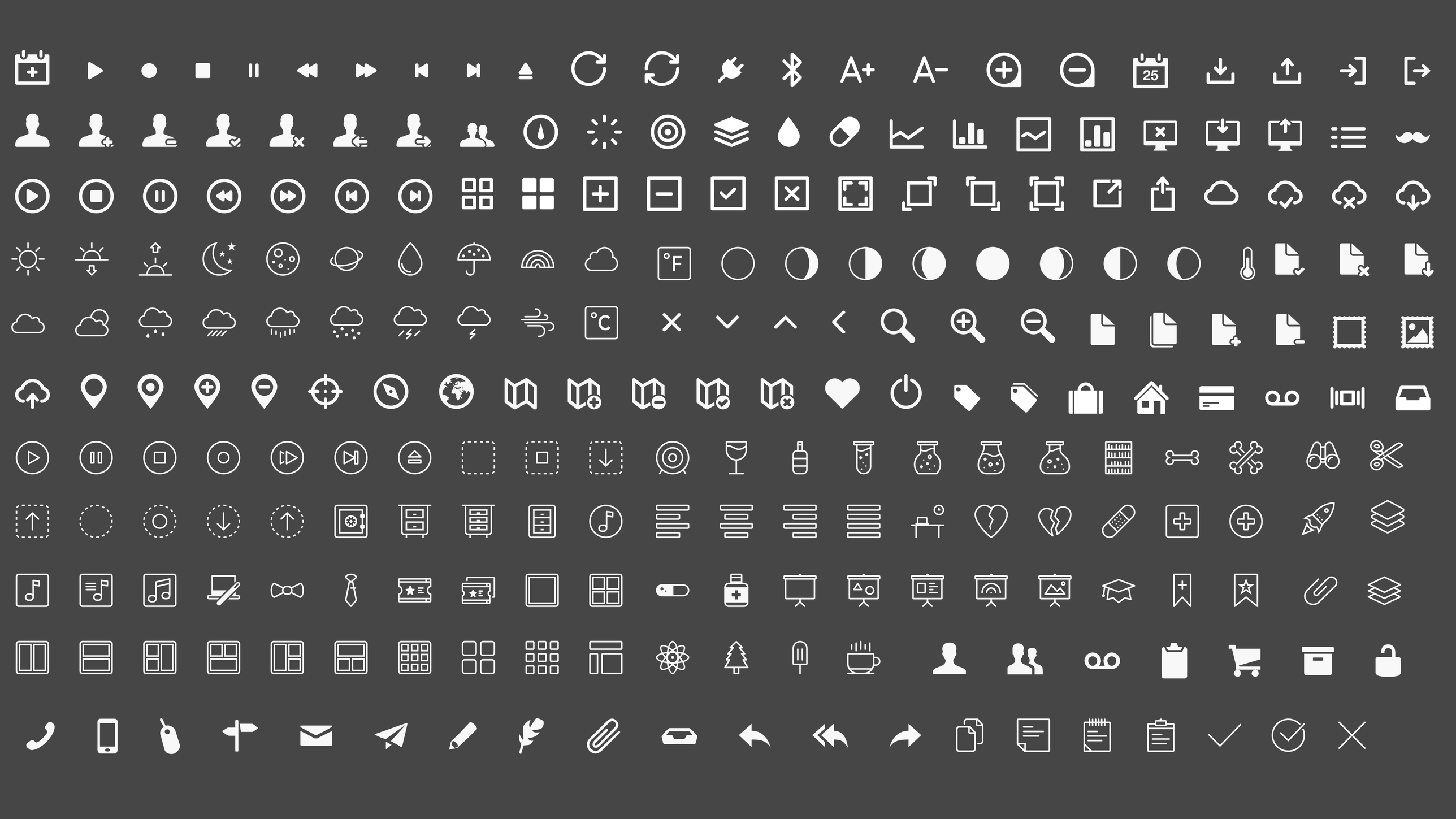
177

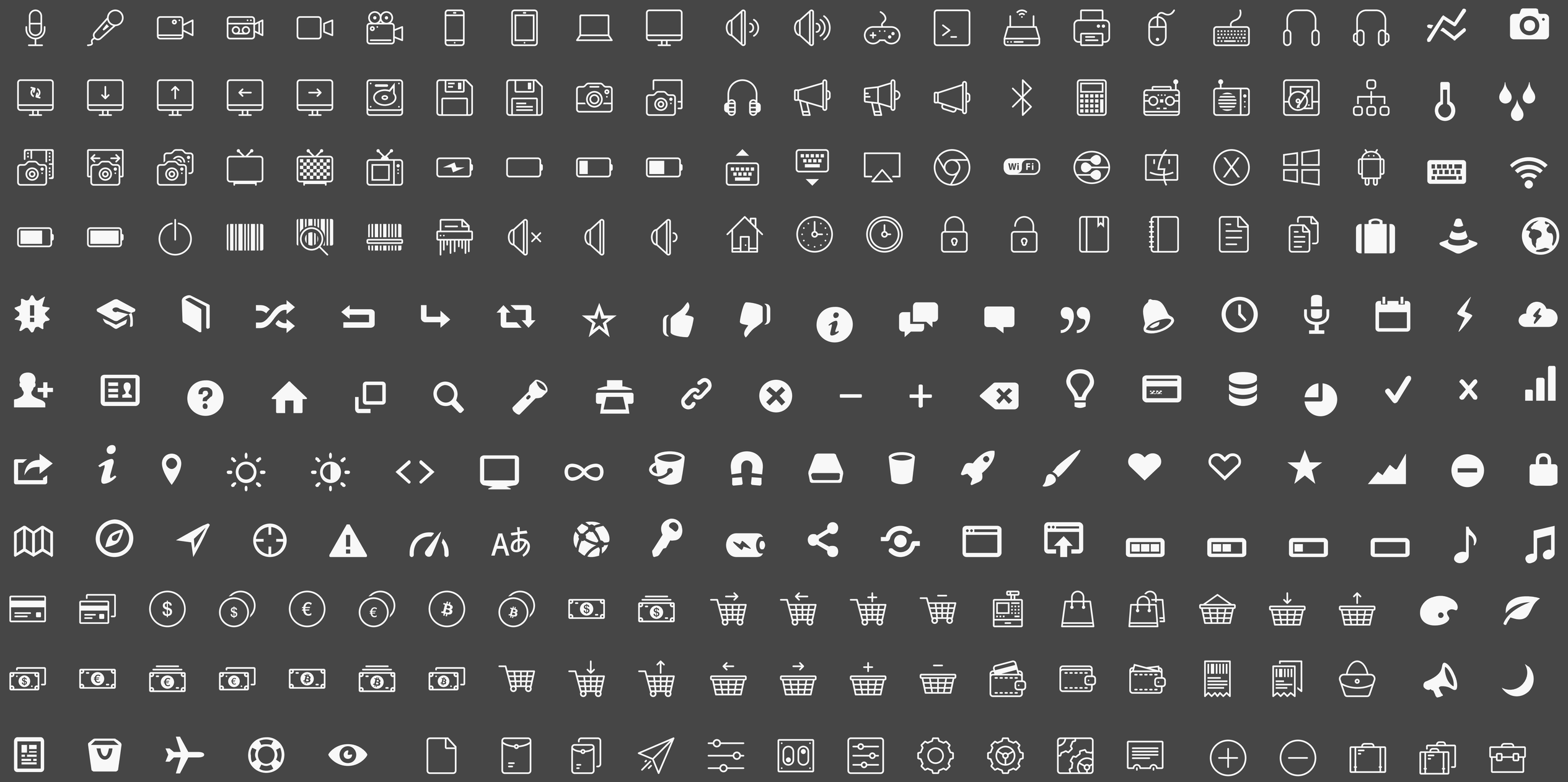
www.blockvisers.net

+! 212-256-1485

put your title here.






Wi Fi
Aあ