BCDV1011 Design Patterns for Blockchain

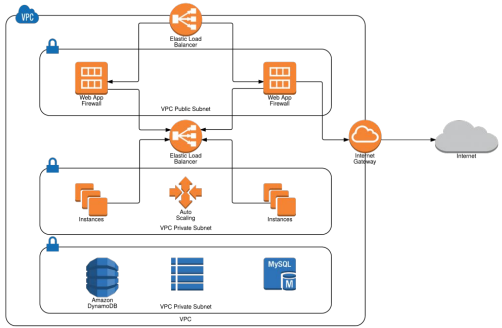
Presenting architectures and building a project plan

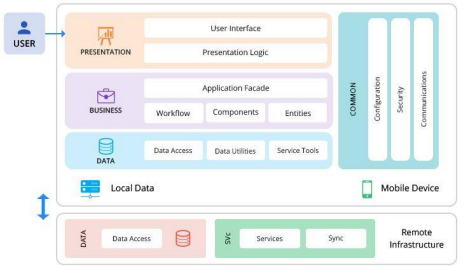
Presenting an architecture

- Architectural document
- Flow diagrams
- State diagrams
- Data definitions

Architectural document

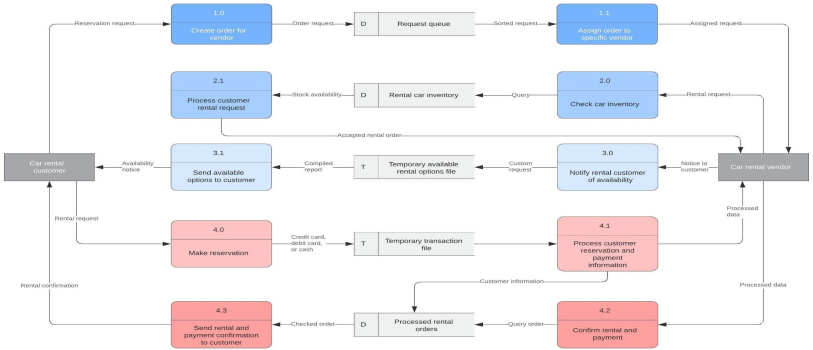
- Describe goals of architecture
- Start from high-level and work down
- Architectures for
  - Smart contract - data (types, names) and functions (parameters, modifiers)
  - Web/mobile app - data, functions
  - Server - microservices, integrations
- Tech stack - technologies, modules, libraries
- Hosting - cloud services, availability

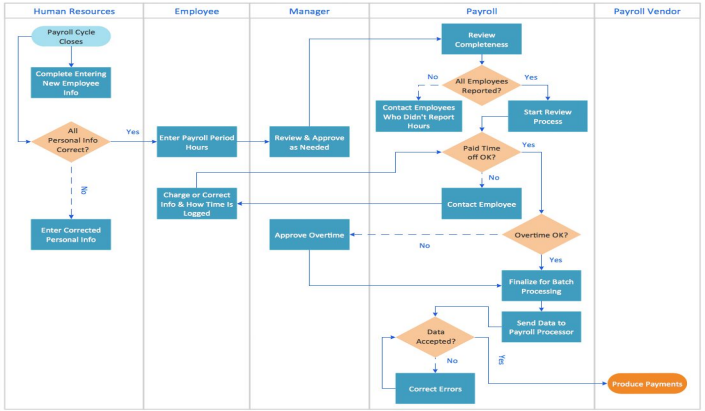




Flow diagram

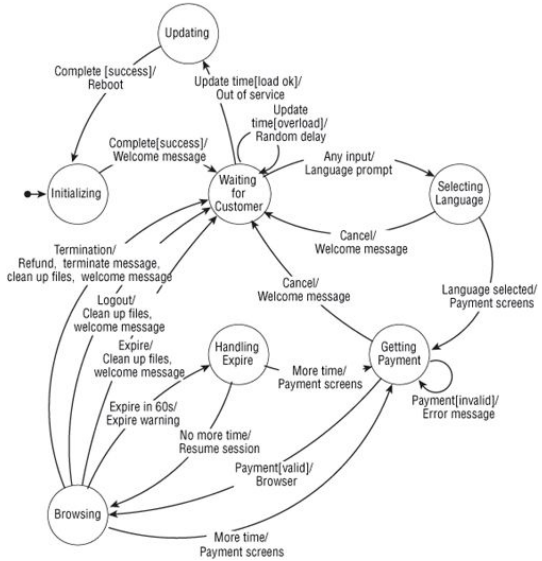
- This is the start of user stories and UX design
- Explains the steps the data/user needs to go through





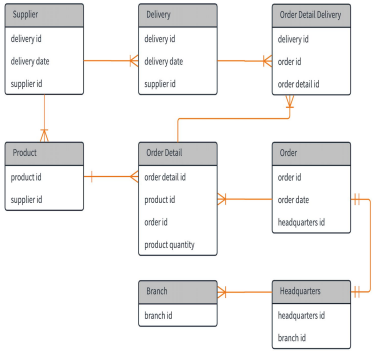
State diagrams

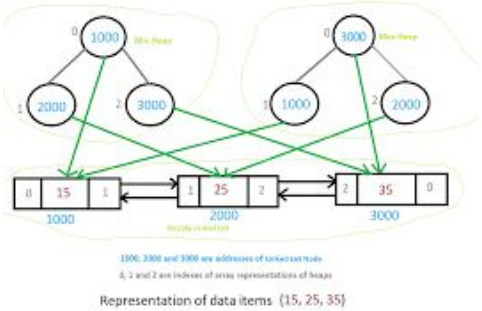
- Identifies the states that the state data is in and the transitions to move between them



Data definitions

- State data types and names
- Data structure explanations
- Database data dictionary
- Entity relationship diagram





Representation of data items {15, 25, 35}

Data Dictionary

Data Dictionary outlining a Database on Driver Details in NSW

Field Name	Data Type	Data Format	Field Size	Description	Example
License ID	Integer	NNNNNN	6	Unique number ID for all drivers	12345
Surname	Text		20	Surname for Driver	Jones
First Name	Text		20	First Name for Driver	Arnold
Address	Text		50	First Name for Driver	11 Rocky st Como 2233
Phone No.	Text		10	License holders contact number	0400111222
D.O.B	Date / Time	DD/MM/YYYY	10	Drivers Date of Birth	08/05/1956

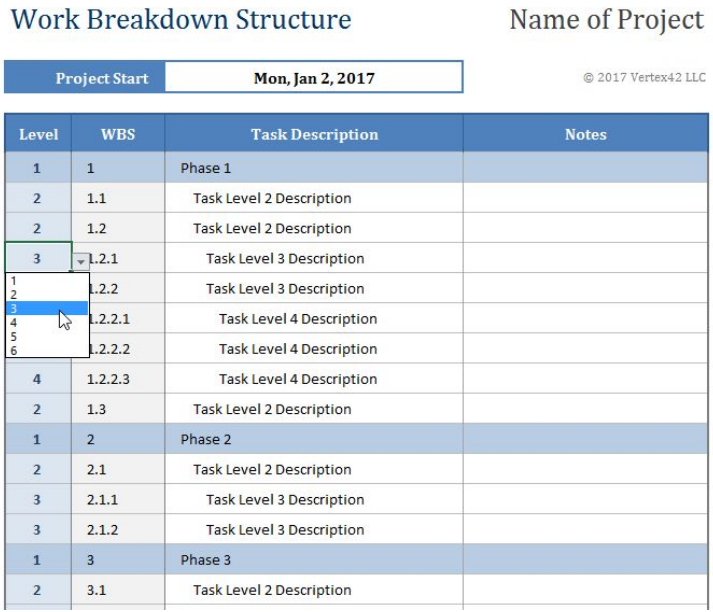
Screencast-O-Matic.com

Creating a project plan

- Project decomposition
- Time estimates
- Staffing
- Dependencies
- Cost estimate

Project decomposition

- List out all of the major tasks, fill in sub tasks
- Breakdown of tasks based on architecture, technology, develop type



Work Breakdown Structure

Name of Project

Project Start	Mon, Jan 2, 2017

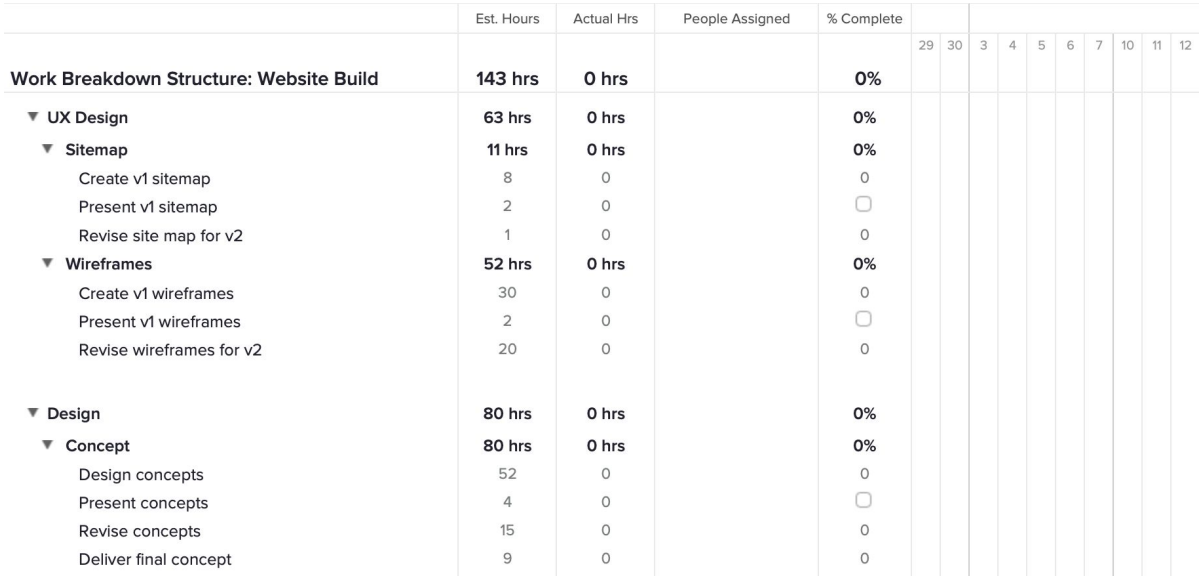
© 2017 Vertex42 LLC

Level	WBS	Task Description	Notes
1	1	Phase 1	
2	1.1	Task Level 2 Description	
2	1.2	Task Level 2 Description	
3	1.2.1	Task Level 3 Description	
	1.2.2	Task Level 3 Description	
	1.2.2.1	Task Level 4 Description	
	1.2.2.2	Task Level 4 Description	
4	1.2.2.3	Task Level 4 Description	
2	1.3	Task Level 2 Description	
1	2	Phase 2	
2	2.1	Task Level 2 Description	
3	2.1.1	Task Level 3 Description	
3	2.1.2	Task Level 3 Description	
1	3	Phase 3	
2	3.1	Task Level 2 Description	

1
2
3
4
5
6

Time estimates

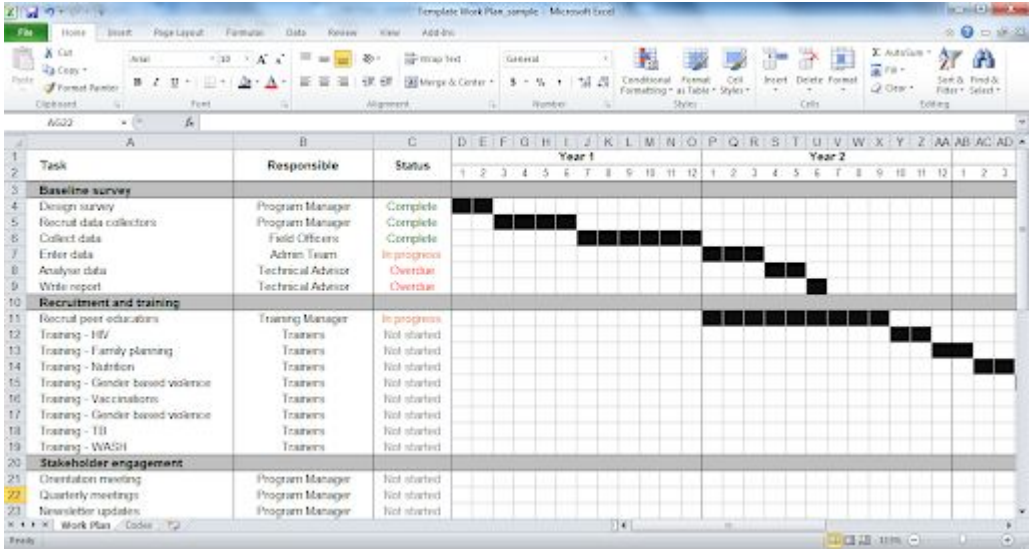
- Estimate how long each sub task will take



	Est. Hours	Actual Hrs	People Assigned	% Complete	29	30	3	4	5	6	7	10	11	12
Work Breakdown Structure: Website Build	**143 hrs**	**0 hrs**		**0%**										
▼ **UX Design**	**63 hrs**	**0 hrs**		**0%**										
▼ **Sitemap**	**11 hrs**	**0 hrs**		**0%**										
Create v1 sitemap	8	0		0										
Present v1 sitemap	2	0		☐										
Revise site map for v2	1	0		0										
▼ **Wireframes**	**52 hrs**	**0 hrs**		**0%**										
Create v1 wireframes	30	0		0										
Present v1 wireframes	2	0		☐										
Revise wireframes for v2	20	0		0										
▼ **Design**	**80 hrs**	**0 hrs**		**0%**										
▼ **Concept**	**80 hrs**	**0 hrs**		**0%**										
Design concepts	52	0		0										
Present concepts	4	0		☐										
Revise concepts	15	0		0										
Deliver final concept	9	0		0										

Staffing

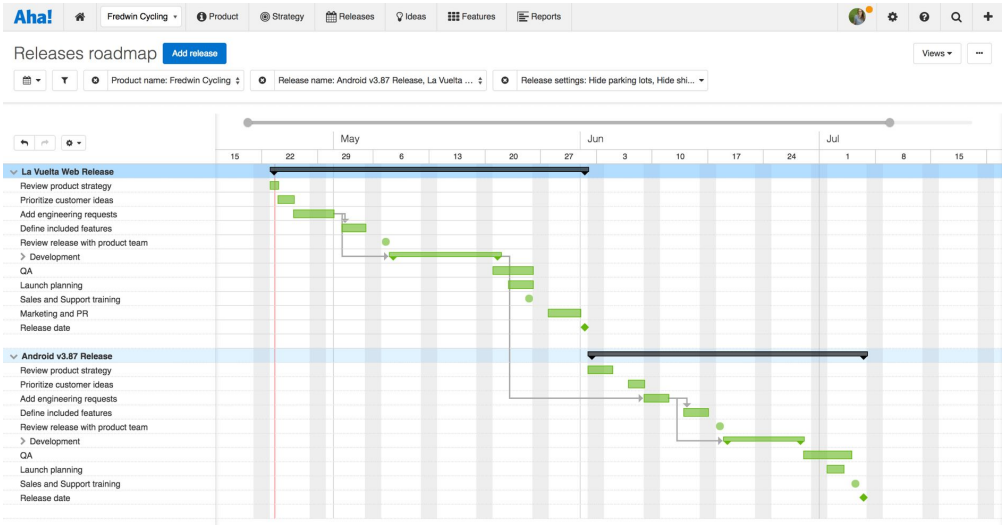
- Assign roles to tasks
- Assign people to roles



Task	Responsible	Status
Baseline survey		
Design survey	Program Manager	Complete
Recruit data collectors	Program Manager	Complete
Collect data	Field Officers	Complete
Enter data	Admin Team	In progress
Analyse data	Technical Advisor	Overdue
Write report	Technical Advisor	Overdue
Recruitment and training		
Recruit peer educators	Training Manager	In progress
Training - HIV	Trainers	Not started
Training - Family planning	Trainers	Not started
Training - Nutrition	Trainers	Not started
Training - Gender based violence	Trainers	Not started
Training - Vaccinations	Trainers	Not started
Training - Gender based violence	Trainers	Not started
Training - TB	Trainers	Not started
Training - WASH	Trainers	Not started
Stakeholder engagement		
Orientation meeting	Program Manager	Not started
Quarterly meetings	Program Manager	Not started
Newsletter updates	Program Manager	Not started

Dependencies

- Assign tasks in order of dependencies
- Look for parallel development opportunities
- Find a balance in size of staff and length of time for project
- Groups of 10 are a good size to limit to



Cost and time estimate

- You now know
  - What tasks need to be done
  - How long each task takes
  - The order of tasks based on dependencies
  - Who is doing each task
- From this you can give a time and cost estimate (based on what you are paying people)

* Software development			
Contractor labor estimate	3000	$150	$450,000
Project team member estimate	1920	$75	$144,000
Total labor estimate			$594,000
Function point estimate	Quantity	Conversion Factor	Function Points
External inputs	10	4	40
External interface files	3	7	21
External outputs	3	5	15
External queries	3	4	12
Logical internal tables	6	10	60
Total function points			148
Java 2 languange equivalency value			46
Source lines of code (SLOC) estimate			6,808
Productivity *KSLOC^Penalty (person months)	30.266		
Total labor hours (160 hours/month)	4,842.551		
Cost/labor hour ($120/hour)	$120		
Total software development estimate	581,106.09		

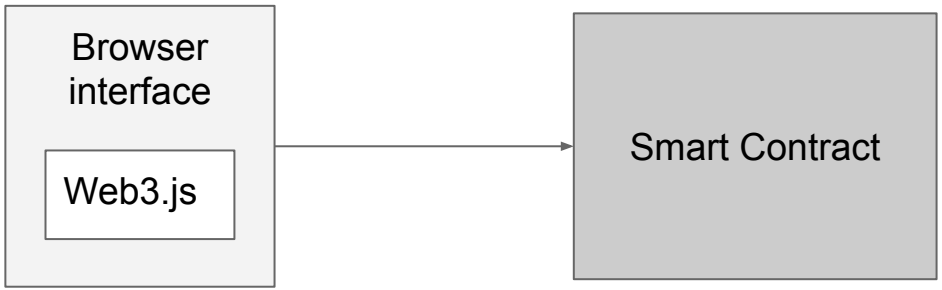
BCDV1011 Design Patterns for Blockchain

Common architectures

Common architectures for Ethereum dApps

- Simple browser dApp
- Simple mobile dApp
- dApp with server
- Hybrid database/dApp
- Simple contract
- Token
- Non-fungible token with factory
- IPFS
- Oracle
- Custodial vs Non-custodial

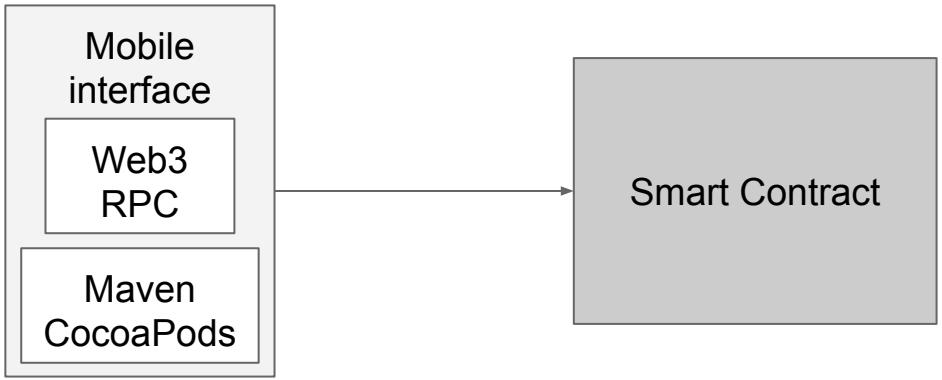
Simple browser dApp



Simple browser dApp

- Browser interface using web3.js to interact with a smart contract
- Easy to setup and deploy
- Uses Metamask to handle transaction signing

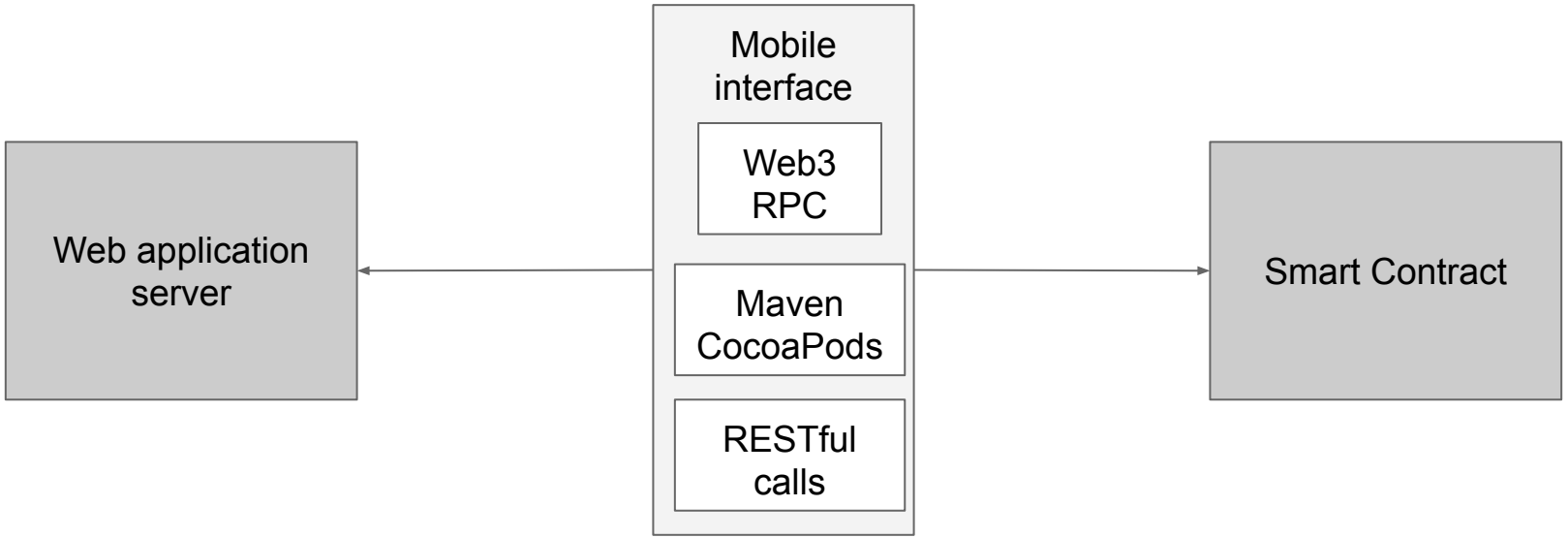
Simple mobile dApp



Simple browser dApp

- Mobile UI, go-etheruem mobile libraries, Web3 RPC
- iOS and/or Android
- Uses libraries for private key management and signing transactions
- Does not require central web application server

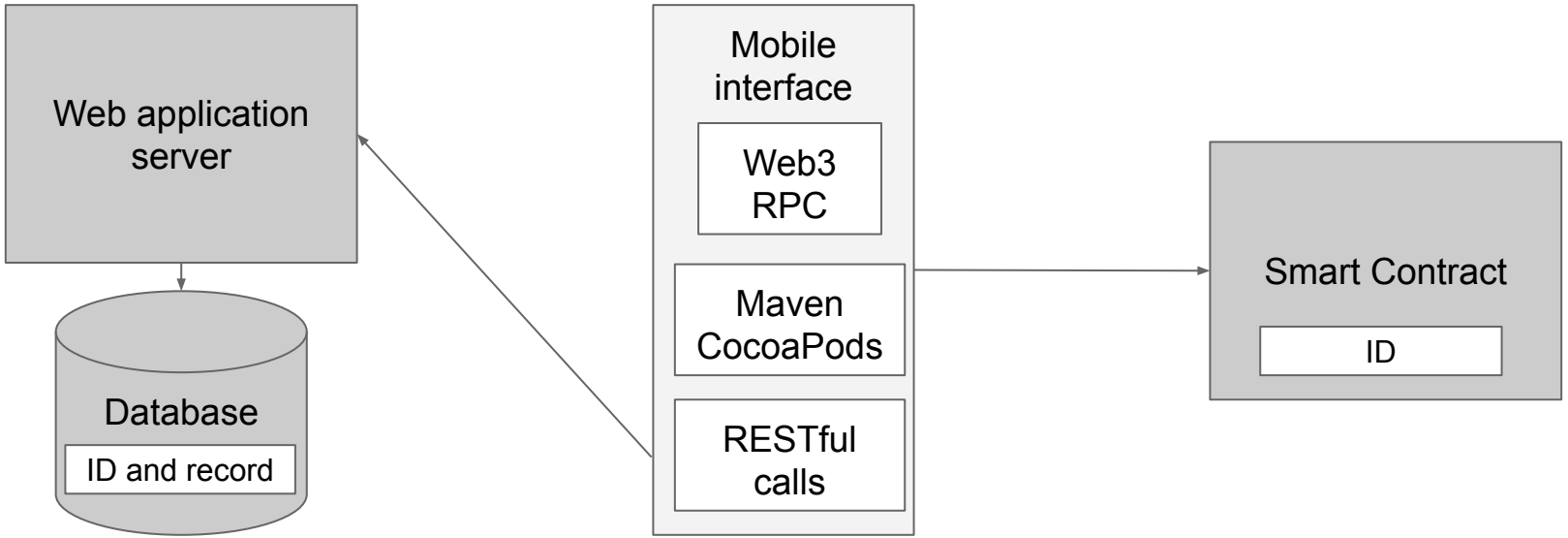
dApp with server



dApp with server

- Mobile or web interface with backend server
- RESTful interface with server
- Server provides
  - More advanced calculations
  - Integration to other systems
  - Access to non-mobile libraries
  - Authentication to services

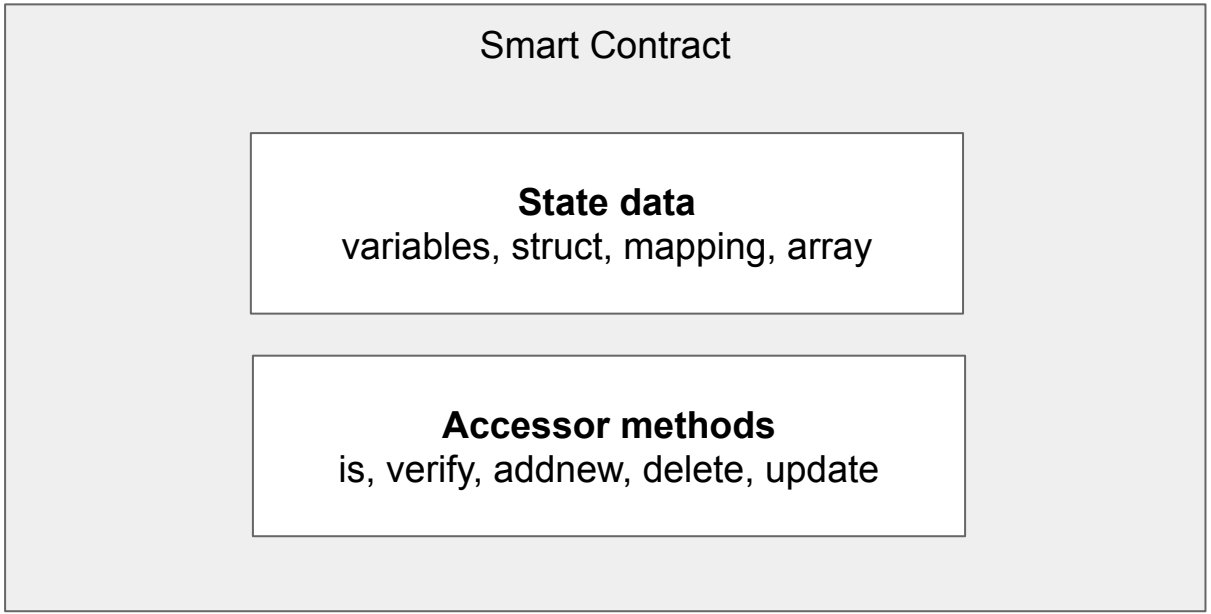
Hybrid database/dApp



Hybrid database/dApp

- Server with database
- ID is stored on blockchain
- Record is stored in central database
- Central database storage is cheaper
- Smart contract does not have access to data record
- NOT decentralized

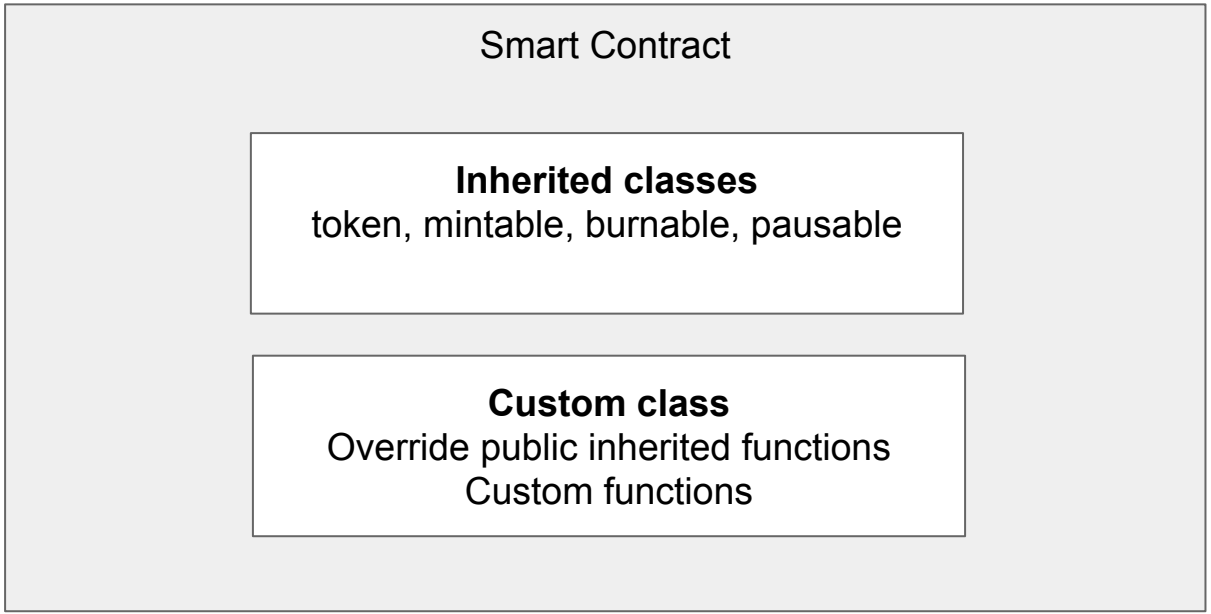
Simple contract



Simple contract

- Custom data storage
- Custom interface

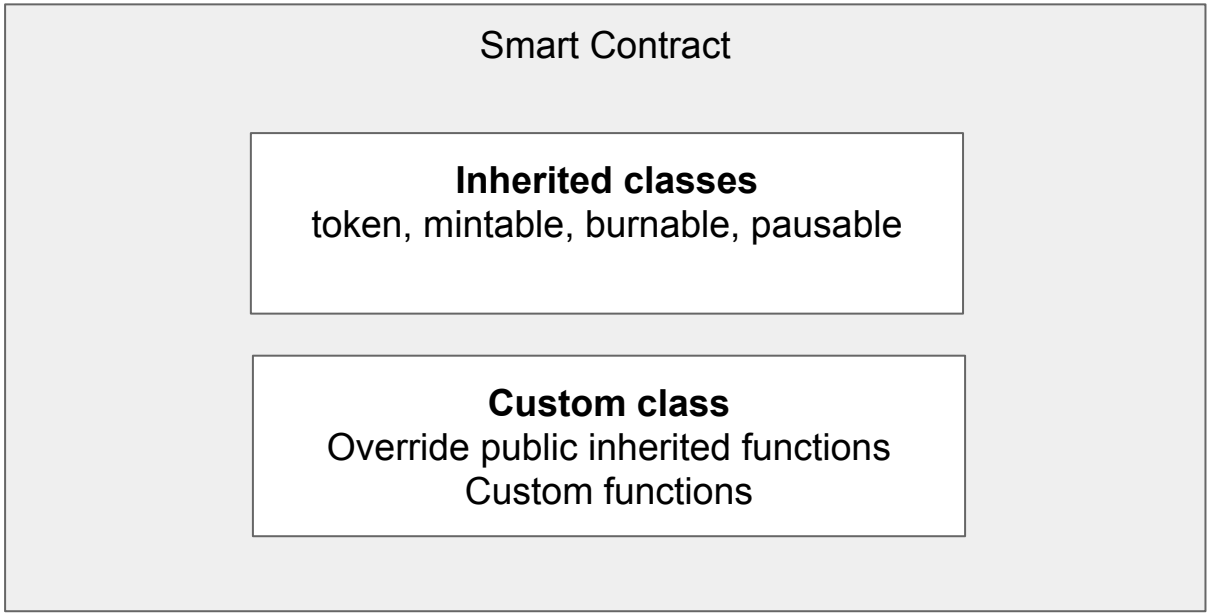
Token


Smart Contract
Inherited classes
token, mintable, burnable, pausable
Custom class
Override public inherited functions
Custom functions

Token

- Pre-existing code - tested, vetted
- Standard interface that can be called without further knowledge of your customizations - exchanges
- ERC-20
  - balanceOf , totalSupply , transfer , transferFrom , approve , and allowance
- ERC-1404 - STO
  - detectTransferRestriction, messageForTransferRestriction
- Deploy token and start transferring

Token


Smart Contract
Inherited classes
token, mintable, burnable, pausable
Custom class
Override public inherited functions
Custom functions

Token

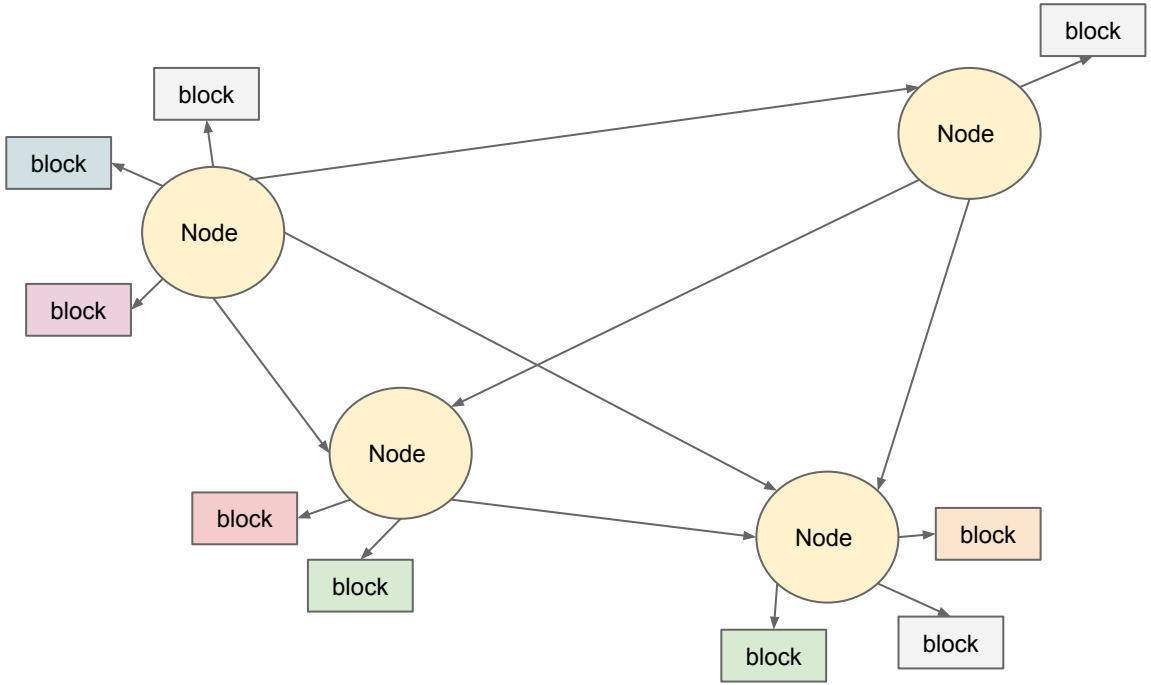
- Pre-existing code - tested, vetted
- Standard interface that can be called without further knowledge of your customizations - exchanges
- ERC-20
  - balanceOf , totalSupply , transfer , transferFrom , approve , and allowance
- ERC-1404 - STO
  - detectTransferRestriction, messageForTransferRestriction
- Good for representing fractional ownership in cases like securities

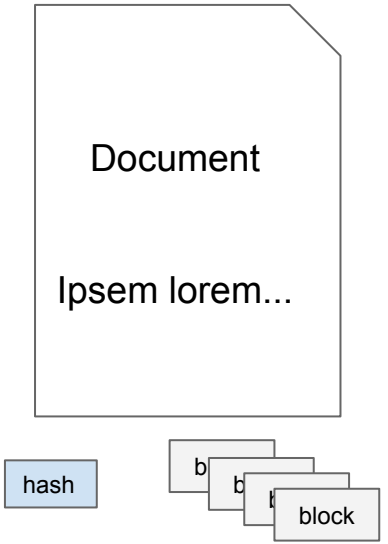
Non-fungible token with factory

Non-fungible token with factory

- ERC-721 non-fungible token with customization
- Custom function to generate new token under conditions
- Keep a list of generate new tokens
- Set the new owner
- The equivalent of minting in the non-fungible world

IPFS with blockchain

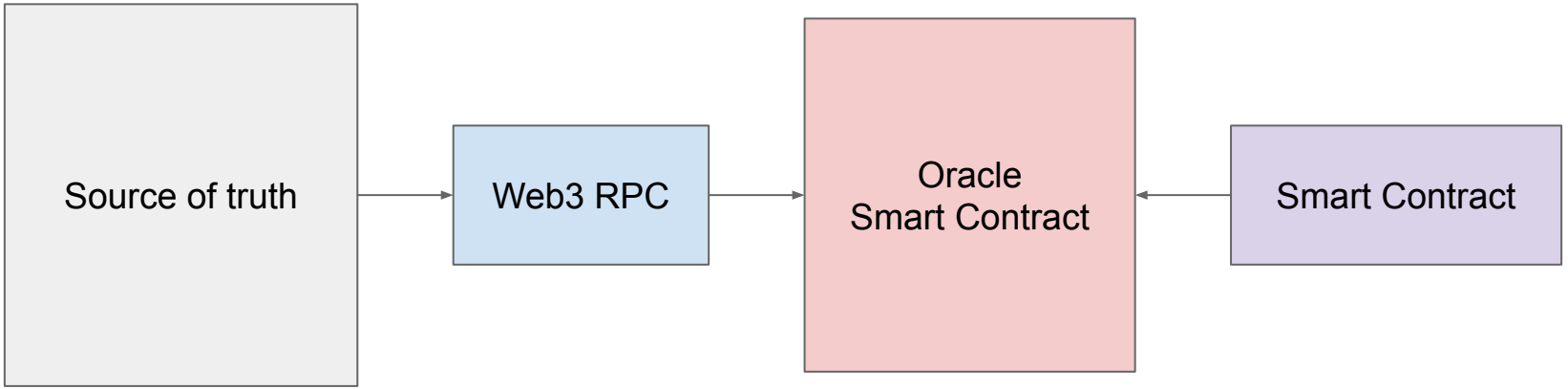




IPFS with blockchain

- Storing data on Ethereum is expensive
- Storing data in a central database is not distributed
- IPFS is distributed
- IPFS uses the cryptographic hash as the storage and lookup index
- IPFS breaks the file into blocks and the blocks are stored all over the network
- IPFS maintains an index to find the closest copies of all of the blocks to retrieve the file
- Store the hash in the blockchain

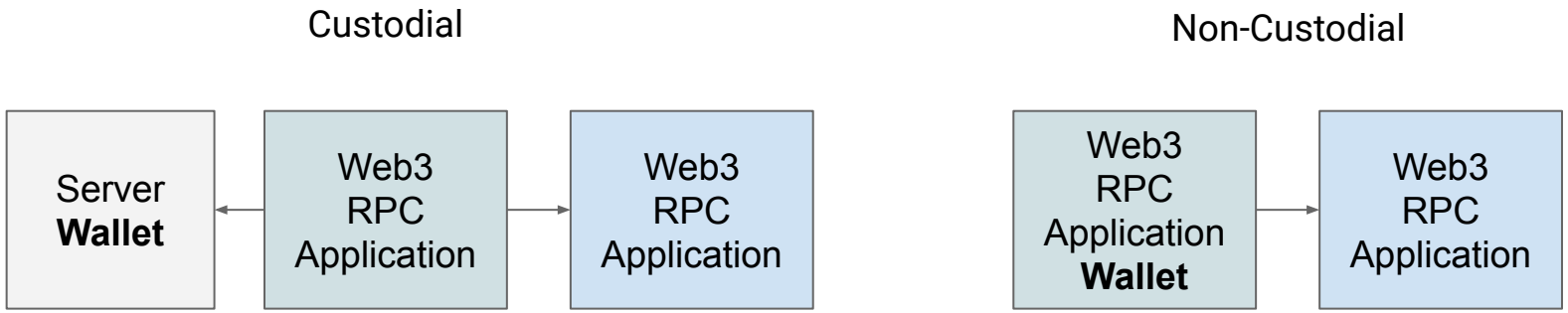
Oracle with blockchain



Oracle with blockchain

- Smart contracts can only work with data on the blockchain
- In reality you will have cases where you need to make your contract use off-chain data
- Build an application to put the data into a smart contract
- Your smart contract can access the oracle smart contract to access the off-chain data
- You can never fully trust off-chain data

Custodial vs non-custodial



Custodial vs non-custodial

Custodial

- Your application holds the private key
- You sign the transactions for the user
- Avoids user losing key
- Simplifies UI
- Not secure

Non-custodial

- User holds their key in their own wallet
- Your dApp needs to support various methods for transaction signing
- User can lose key
- Secure

Fully decentralized

Requirements to be fully decentralized

- Non-custodial
- Governance organization
- Open source smart contracts
- Open source UI code
- All data on chain
- Anyone can run UI
- Transparent processes from governance to source

BCDV1011 Design Patterns for Blockchain

Business patterns

Common patterns

- Escrow
- Voting
- Payment
- Assets

Escrow

- An Escrow is an arrangement for a third party to hold the assets of a transaction temporarily
- The smart contract holds the asset
- The rules of the smart contract may be the third party
- The use of an escrow account in a transaction adds a degree of safety for both parties

Escrow – variants and features

- Contingent beneficiary
- Decreasing term payout
- Deferred payment
- Endowment
- Escheatment
- Security Deposit
- Proof of funds
- Restricted funds
- Arbitration
- Trust Fund
- Annuity
- Trustless

Voting

- A vote is used to express a wish to follow a particular course of action
- Provides representation of the wishes of multiple parties
- Guides collective action
- Most important part of governance

Voting - variants and features

- Bidding
- Election
- RFP
- Arbitration
- First past the post
- Proportional
- Representational
- Weighted

Payments

- A payment is the action or process of paying someone or something or of being paid
- Part of the value transfer in business transactions
- Requires settlement

Payment – variants and features

- Licensing
- Consignment
- Royalties
- Incoterms
- Invoicing
- Purchase Order
- Split Tender
- Taxation

Assets

- An asset is property owned by a person or company, regarded as having value and available to meet debts, commitments, or legacies
- May increase in value or be depreciated
- Can be physical or abstract

Asset - variants and features

- Fractional
- Equity
- Alternative
- Vesting
- Restrictions

Building an escrow service

- Three party transaction
  - Buyer
  - Seller
  - Agent

Building an escrow service

- Three party transaction
  - Buyer
  - Seller
  - Agent
- Role-based access control

Building an escrow service

- Flow
  - Conditions for release are set
  - Buyer transfers funds to contract
  - If conditions met
    - Agent releases to Seller
  - If conditions fail
    - Agent releases to Buyer
  - Agent gets a % or fixed fee

Smart contract

- Use RBAC pattern
- Make parties payable
- Agent can release or revert
- Fixed fee for agent

Web interface

- Website represents the seller
- Buyer interface
  - The buyer selects the escrow agent
  - The buyer selects the conditions (some sort of oracle?)
  - The buyer sends the funds
- Agent interface
  - The agent checks condition
  - Agent releases or reverts

BCDV1011 Design Patterns for Blockchain

Designing applications with tokens

Tokens and token factories

- Token types
- Where to use tokens
- Example utility token
- Example from use case

Tokens types

- Token types
  - Utility
  - Security
  - Native coin
  - Governance
  - Asset

Utility Tokens

- Needs to be an integral part of a system - not just a stand-in for a security
- Based on ERC20
- Added extra functions to provide utility
- Needs to be a functional part of a platform

Security Tokens

- Could be modified ERC20 or another standard
- Represents securities - shares in a company
- Should be securities regulators compliant
- Howie test
- Reg D 504c
- Qualified investors

Native coin

- Built-in for blockchain
- Used for transfer fees, mining or staking
- Bitcoin, Ether, XRP, Lumens
- Utility/Currency

Governance

- To fund/control a decentralized system
- Represents voting rights
- Paid dues
- Funds projects

Token uses - precision

- Used when current financial divisions are not enough
- Micro transactions
- Fractional ownership

Token uses - disintermediation

- Remove middle men
- Smart contracts to handle trust
- Holding funds in escrow
- Transparency for voting
- Explicit rules for processes

Token uses - risk reduction

- Trade risk for security
- Derivatives for hedging
- Fungible liquidity

Token uses - complimentary services

- Behaviour rewards
- Frequent customer
- Sell your info
- Watch ads

Token uses - crowdsourcing

- Collective funding of projects
- Voting mechanisms
- Escrow
- Payouts
- Prediction markets

Token uses - marketplace

- Decentralized markets
- Confederation of competitors
- Repository of goods for curation

Token uses - fractional ownership

- Large asset purchase funding
- Group ownership
- Securtize alternative assets
- Create liquidity in new markets

Token uses - digital representation

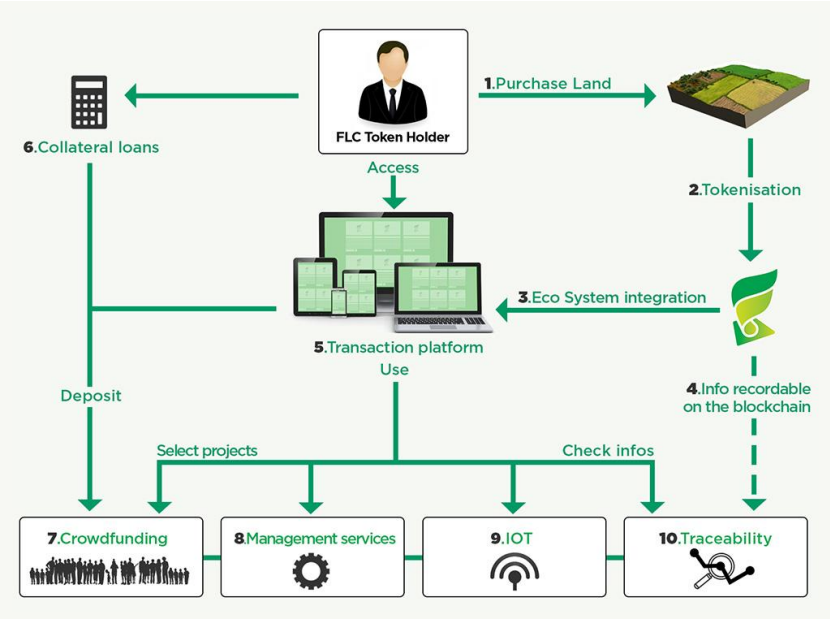
- Digital clone to real world goods
- Supply chain
- Industrial processes
- Stand in

Class examples

- Arts provenance
- Micro loans
- Real estate bidding
- Perishable goods supply chain
- Group Project Bidding
- Timeshare reservation
- Community content rewards
- Green energy tracker
- License plate registry

Fieldcoin - example

- Crowdfunding for agricultural land purchases



Fieldcoin - example

- https://github.com/Fieldcoin/Fieldcoin-ERC20
- Two tokens
  - FieldCoinSale - pre-sale token for special enticements
  - FieldCoin - token for working with the FieldCoin system
- FieldCoin is mintable, burnable
- FieldCoinSale is crowdsale, pausable
- Derived from StandardToken and ERC20

Token factory - use case

- Art provenance
- The goal
  - To show a clear line of ownership for a work of art from artist to current owner.
- People involved
  - Artist - originator of the art work
  - Owner - current owner of the artwork
- Requirements
  - Only one owner at a time
  - Artist can create new works of art

Token factory - use case

- The artist can have a collection of artworks that they have created and each artwork can have multiple limited edition prints

```
contract Artist {
    // Collection of artworks by this Artist
    mapping(uint => ArtWork) artworks;
    address artist;

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

Token factory - use case

- The artist can make new artworks and add them to their collection

```
function createArtwork(uint hashIPFS, string memory Name) public returns (ArtWork) {
   ArtWork artContract = new ArtWork(hashIPFS, Name);
   artworks[hashIPFS] = artContract;
   return artContract;
}
```

- Check to see if the artist is the originator of an artwork

```
function checkArtwork(uint hashIPFS) public view returns(bool) {
     if(artworks[hashIPFS] == ArtWork(0x0)) {
         return true;
     }
     return false;
}
```

Token factory - use case

- An artwork is stored on IPFS and we keep the hash and name of the artwork on the blockchain

```
contract ArtWork {
    // Detail of artwork
    address artist;
    string  name;
    uint  hashIPFS;
    address owner;

    constructor(uint ipfsHash, string memory artName) public {
        artist = msg.sender;
        name = artName;
        hashIPFS = ipfsHash;
        owner = artist;
    }
```

Token factory - use case

- An artwork is stored on IPFS and we keep the hash and name of the artwork on the blockchain

```
contract ArtWork {
    // Detail of artwork
    address artist;
    string  name;
    uint  hashIPFS;
    address owner;

    constructor(uint ipfsHash, string memory artName) public {
        artist = msg.sender;
        name = artName;
        hashIPFS = ipfsHash;
        owner = artist;
    }
```

Token factory - use case

- An artwork can change ownership

```
function setOwner(address newOwner) public {
    if(owner == msg.sender) {
        owner = newOwner;
    }
}
```

BCDV1011 Design Patterns for Blockchain

Common patterns

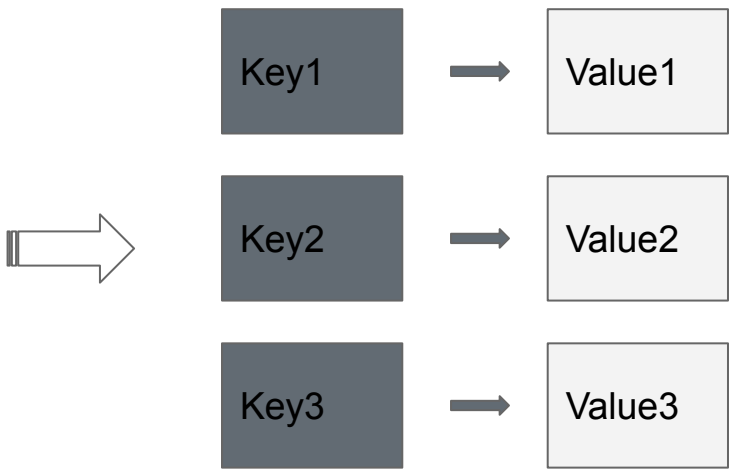
Common patterns

- Mappings
- Iterators
- Data Structures
- Ownership
- RBAC
- Factory

Mappings

- Smart contracts are based on two things **transfers** and **shared state**
- Mappings are key/value pairs
- Mappings in Solidity are assumed infinite
- The value of a mapping can be a value, another mapping or a structure
- Tokens use mappings to track the owning accounts
- Registries use mappings to track who is in the registry

Mappings



Mappings

```
mapping(key_type => value_type)
```

```
mapping(address => uint) public balances;
```

```
balances[msg.sender] = deposit;
```

Mappings

```
struct Student {
    uint age;
    string name;
}
mapping (address => Student) public class;
Student memory newStudent;
newStudent.age = 56;
newStudent.name = “Dave”;
class[msg.sender]=newStudent;
```

Mappings

```
mapping(address => mapping(uint => bool)) lotterypicks;

lotterypicks[msg.sender][2456738992] = true;
```

Mappings

- There is no order to a mapping through solidity
- You can’t iterate over a mapping
- There is no syntax like in other languages where you can loop though the contents of a collection
- You would have to go back to the ledger to see all of the transactions that resulted in entries to the mapping

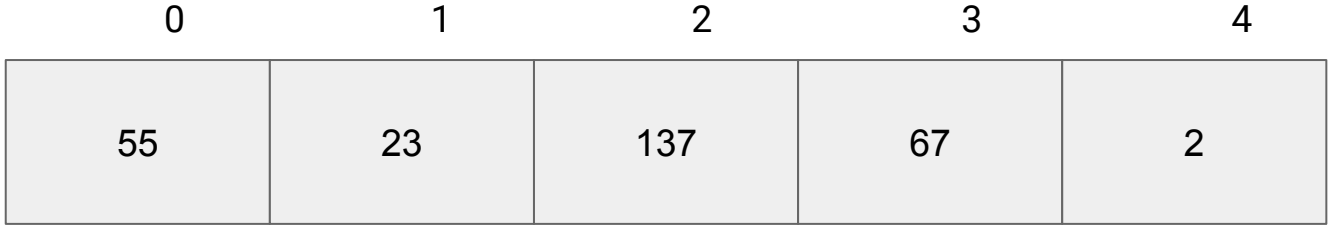
Iterators

- You still need to iterate in some cases
- Make sure that your case is bounded by a low enough amount that you do not run out of gas
- If you need to iterate over a larger number of items - don’t use Ethereum

Arrays

- Arrays respect order
- Arrays have random access
- Arrays can provide a count

Arrays




```
uint[] prices;
prices.push(55);
prices.push(23);

uint[5] prices;
prices[0] = 55;

uint[5] prices = [uint(55), 23, 137, 67, 2];
```

Arrays

```
struct Student {
    uint age;
    string name;
}
Student[ ] public class;
Student memory newStudent;
newStudent.age = 56;
newStudent.name = “Dave”;
class.push(newStudent);
```

Iterators

- Walk through each element of a collection
- Used for searching for matching a condition
- Apply a function to all or some entries of a collection

Data Structures

Simple List Using Array

Strengths

- Reliably chronological order
- Provides a count
- Random access by Row Number (not Id)

Data Structures

Simple List Using Array

Weaknesses

- No random access by Id
- No assurance of uniqueness
- No check for duplicates
- Uncontrolled growth of the list

Data Structures

```
contract simpleList {

  struct EntityStruct {
    address entityAddress;
    uint entityData;
  }

  EntityStruct[] public entityStructs;

  function newEntity(address entityAddress, uint entityData) public returns(uint rowNumber) {
    EntityStruct memory newEntity;
    newEntity.entityAddress = entityAddress;
    newEntity.entityData    = entityData;
    return entityStructs.push(newEntity)-1;
  }

  function getEntityCount() public constant returns(uint entityCount) {
    return entityStructs.length;
  }
}
```

Data Structures

Mapping with Struct

Strengths

- Random access by unique Id
- Assurance of Id Uniqueness
- Enclose arrays, mappings, structs within each "record"

Data Structures

Mapping with Struct

Weaknesses

- Unable to enumerate the keys
- Unable to count the keys
- Needs a manual check to distinguish a default from an explicitly "all 0" record

Data Structures

```
contract mappingWithStruct {
  struct EntityStruct {
    uint entityData;
    bool isEntity;
  }

  mapping (address => EntityStruct) public entityStructs;

  function isEntity(address entityAddress) public constant returns(bool isIndeed) {
    return entityStructs[entityAddress].isEntity;
  }

  function newEntity(address entityAddress, uint entityData) public returns(bool success) {
    if(isEntity(entityAddress)) throw;
    entityStructs[entityAddress].entityData = entityData;
    entityStructs[entityAddress].isEntity = true;
    return true;
  }
```

Data Structures

```
  function deleteEntity(address entityAddress) public returns(bool success) {
    if(!isEntity(entityAddress)) throw;
    entityStructs[entityAddress].isEntity = false;
    return true;
  }

  function updateEntity(address entityAddress, uint entityData) public returns(bool success) {
    if(!isEntity(entityAddress)) throw;
    entityStructs[entityAddress].entityData = entityData;
    return true;
  }
}
```

Data Structures

Array of Structs with Unique Ids

Strengths

- Random access by Row number
- Assurance of Id uniqueness
- Enclose arrays, mappings and structs with each "record"

Data Structures

Mapping with Struct

Weaknesses

- No random access by Id
- Uncontrolled growth of the list

Data Structures

```
contract arrayWithUniqueIds {

  struct EntityStruct {
    address entityAddress;
    uint entityData;
  }

  EntityStruct[] public entityStructs;
  mapping(address => bool) knownEntity;

  function isEntity(address entityAddress) public constant returns(bool isIndeed) {
    return knownEntity[entityAddress];
  }

  function getEntityCount() public constant returns(uint entityCount) {
    return entityStructs.length;
  }
```

Data Structures

```
  function newEntity(address entityAddress, uint entityData) public returns(uint rowNumber) {
    if(isEntity(entityAddress)) throw;
    EntityStruct memory newEntity;
    newEntity.entityAddress = entityAddress;
    newEntity.entityData = entityData;
    knownEntity[entityAddress] = true;
    return entityStructs.push(newEntity) - 1;
  }

  function updateEntity(uint rowNumber, address entityAddress, uint entityData) public
returns(bool success) {
    if(!isEntity(entityAddress)) throw;
    if(entityStructs[rowNumber].entityAddress != entityAddress) throw;
    entityStructs[rowNumber].entityData = entityData;
    return true;
  }
}
```

Data Structures

Mapped Structs with Index

Strengths

- Random access by unique Id or row number
- Assurance of Id uniqueness
- Enclose arrays, mappings and structs within each "record"
- List maintains order of declaration
- Count the records
- Enumerate the Ids
- "Soft" delete an item by setting a boolean

Data Structures

Mapped Structs with Index

Weaknesses

- Uncontrolled growth of the list

Data Structures

```
contract MappedStructsWithIndex {

  struct EntityStruct {
    uint entityData;
    bool isEntity;
  }

  mapping(address => EntityStruct) public entityStructs;
  address[] public entityList;

  function isEntity(address entityAddress) public constant returns(bool isIndeed) {
      return entityStructs[entityAddress].isEntity;
  }

  function getEntityCount() public constant returns(uint entityCount) {
    return entityList.length;
  }
```

Data Structures

```
  function newEntity(address entityAddress, uint entityData) public returns(uint rowNumber) {
    if(isEntity(entityAddress)) throw;
    entityStructs[entityAddress].entityData = entityData;
    entityStructs[entityAddress].isEntity = true;
    return entityList.push(entityAddress) - 1;
  }

  function updateEntity(address entityAddress, uint entityData) public returns(bool success) {
    if(!isEntity(entityAddress)) throw;
    entityStructs[entityAddress].entityData    = entityData;
    return true;
  }
}
```

Ownership

- Smart contracts act like accounts
- Ownership is required to limit access to certain account activities
- It may be desirable to transfer ownership
- You can have different types of ownership
- Ownable.sol

openzeppelin-contracts/Ownable.sol at master · OpenZeppelin/openzeppelin-contracts

Role Based Access Control

- Different classes of users with different rights to do things with your contract
- Add and remove accounts from having a role
- Verify if an account has access to do something
- Roles.sol

openzeppelin-contracts/Roles.sol at master · OpenZeppelin/openzeppelin-contracts

Role Based Access Control

- Track the different roles in your contract
- Add and remove roles
- A starting point for more advanced access control
- RBAC.sol

real-estate-standards/RBAC.sol at master · ConsenSys/real-estate-standards

Factory

- Generate a new smart contract
- Often used for generating new token contracts
- Factory.sol

openzeppelin-sdk/Factory.sol at master · OpenZeppelin/openzeppelin-sdk

Factory

- Example

```
contract CarFleet {
   address[] fleet;
   function createChildContract(string make, string model) public payable {
      address newCar = new Car(make, model);
      fleet.push(newCar);
   }
   function getDeployedChildContracts() public view returns (address[]) {
      return fleet;
   }
}
contract Car {
   string public make;
   string public model;
   function Car(string _make, string _model) public {
      make = _make;
      model = _model;
   }
}
```

Other patterns of note

- Mintable
- Burnable
- Pausable
- Withdrawl

BCDV1011 Governance

Governance

Governance

1. What is it
2. Why is it required
3. Decentralized governance
4. Types and Related
5. Trust Frameworks
6. Governance Frameworks
7. Organization Governance
8. Open Source Governance

What is governance

- Governance is the way rules, norms and actions are structured, sustained , regulated and held accountable. The degree of formality depends on the internal rules of a given organization and, externally, with its business partners.
- Governments - make laws
- Organizations - policies and procedures
- Endeavours - legal agreements

Why governance?

- Provides a common set of rules for all parties involved to follow
- Sets up mechanisms to resolve conflict
- Creates a predictable environment to operate in
- Gives a say in the process to the parties involved
- Helps to achieve long term stability

Decentralized governance

- No central authority
- All parties are treated equitably
- All parties are represented in decision process
- Even the decision process is decided upon (governance of governance)
- Transparency in process and proceedings

Types and Related

- Governance Framework
- Corporate Governance
- Governance Agreement
- Charter
- Trust Framework
- Trust Assurance Framework
- Consensus Mechanism

Governance Framework

- Roles
- Activities
- Decision making

https://wiki.polkadot.network/docs/en/learn-governance

https://bedrock-consortium.github.io/bbu-gf/

Trust Framework

- Compliance rules to ensure trust
- Can be used to build a trustmark

https://diacc.ca/trust-framework/

https://www.gov.uk/government/publications/the-uk-digital-identity-and-attributes-trust-framework

- Compliance laws - GDPR, CCP, PIPEDA, HIPPA
- Conformance rules/Standards - NIST, ISO

Organization governance

- Corporate bylaws
  - Defines board and committee governance
- Shareholder agreement
  - Defines rules on issuance, transfer, dividends and voting

Open source governance

- Repository
- Roles - owner, maintainer, writer, reader
- Types of governance
  - "Do-ocracy"
  - Founder-leader
  - Self-appointing council or board
  - Electoral
  - Corporate-backed
  - Foundation-backed

BCDV1011 Design Patterns for Blockchain

Build an Oracle and use IPFS

Build day!

- Build an Oracle
- Integrate to IPFS

Goal for Oracle

- Create a Smart Contract
- Create an Express server app
- Call a REST interface to get data
- Create a simple React app
- Call Express app from React to get data
- Write the data to the smart contract
- Read back from the smart contract

Oracle Smart Contract

- Use Remix and Metamask
- Create state data

```
/// quote structure
struct stock {
    uint price;
    uint volume;
}

/// quotes by symbol
mapping( bytes4 => stock) stockQuote;

/// Contract owner
address oracleOwner;
```

Oracle Smart Contract

- Create functions

```
/// Set the value of a stock
function setStock(bytes4 symbol, uint price, uint volume) public {...}

/// Get the value of a stock
function getStockPrice(bytes4 symbol) public view returns (uint) {...}

/// Get the value of volume traded for a stock
function getStockVolume(bytes4 symbol) public view returns (uint) {...}
```

- Test in Remix

Oracle Express.js server app

- Use the Express app generator

```
npx express-generator

Cd myapp

npm install axios

npm start
```

- Make call to your REST API to gather data

REST call

- Register for you free access to REST call
  - https://www.alphavantage.co/

```
fetch('https://www.alphavantage.co/query?function=GLOBAL_QUOTE&symbol=MSFT&apikey=KEY')
    .then(res => res.json())
    .then((data) => {
      this.setState({ quote: data["Global Quote"] })
    })
    .catch(console.log)
```

Oracle React App

- Install and run Ganache locally
- Create new app with create-react-app

```
npm install -g create-react-app

npx create-react-app new-oracle

cd new-oracle
npm start


npm install web3
```

Connect React to Smart Contract using Web3

- Get a copy of your ABI and contract address from remix
- Put them in a file that we can use lie src/quotecontract.js

```
export const STOCK_ORACLE_ADDRESS = '0x0YOURADDRESS'

export const STOCK_ORACLE_ABI = [ ... YOUR ABI ... ]
```

- Import in the stuff we need to connect

```
import Web3 from 'web3';
import { STOCK_ORACLE_ABI, STOCK_ORACLE_ADDRESS } from './quotecontract'
```

Connect React to Smart Contract using Web3

```
const web3 = new Web3("http://localhost:7545")
const accounts = await web3.eth.getAccounts()
console.log("Account 0 = ", accounts[0] )

const stockQuote = new web3.eth.Contract(STOCK_ORACLE_ABI, STOCK_ORACLE_ADDRESS)

var retval = await stockQuote.methods.getStockPrice(web3.utils.fromAscii("AAAA")).call();
console.log(retval);
```

React interface

- Create an interface in React
  - Ask for stock symbol
  - Lookup symbol using REST call
  - Write it to smart contract using setStock call
  - Read back and display results from smart contract using getStockPrice, getStockVolume
- Verify that it works by using getStockPrice, getStockVolume on remix with the symbol

Goal for IPFS

- Upload a file to IPFS from webpage
- Store hash on blockchain
- Read back hash
- Retrieve file
- Display on webpage

Local IPFS

- Get a copy of the IPFS from https://dist.ipfs.io/#go-ipfs
- Extract the tar.gz file tar -xvf and run the install.sh

```
ipfs init
```

- Check out the quick-start
- In a new terminal window run

```
ipfs daemon
```

- Check out your local interface on http://localhost:5001/ipfs/

React interface to IPFS

- Create your react app

```
npm install -g create-react-app
npx create-react-app new-ipfs
cd new-ipfs

npm install fs-extra
npm install ipfs
npm install web3
```

React interface to IPFS

- Connect to your local IPFS
- Make an ipfs.js file

```
const ipfsApi = require('ipfs-api');
const ipfs = new ipfsApi('localhost', '5001', {protocol:'http'});
export default ipfs;
```

React interface to IPFS

- Upload file
  - Open file
  - Save to buffer
  - Call ipfs add
  - Save the hash that is returned

React interface to IPFS

- Retrieve file
  - Need the hash
  - Buffer to read to
  - Call ipfs get
  - Write buffer out to a file

BCDV1011 Design Patterns for Blockchain

Requirements Analysis

Requirements gathering

- Goals
- Stakeholders
- State data
- Restrictions
- Exceptions

Goals

- What is the client trying to achieve?
- How do they measure that?
- Problem statement

Stakeholders

- Who is involved?
- What are the roles that they play?
- What are their restrictions?

State Data

- What is the system tracking?
- What needs to be captured?
- What is generated?

Restrictions

- Are there restrictions by roles/users?
- Are there date/time restrictions?
- Limitations by rules?
- Calculated restrictions?

Exceptions

- Can any of the rules be broken under certain circumstances?
- Should any new rules be added in certain circumstances
- What about edge conditions?

Example

Media Timestamp

Description

A system that creates a blockchain record representing media data that can be used to show authenticity and authorship.

Problem

We are in an age where media is hard to trust:

- Online text can be easily changed. This can be used to distort the record of online conversations.
- Photos and videos can be manipulated to change the image to suit the needs of a political agenda.
- Audio can be edited to change the order of words or remove offending excerpts.

Example

Media Timestamp - Goals

- Record a timestamp on Ethereum to show ownership
- Be able to prove ownership from a media file

Example

Media Timestamp - Stakeholders

- Owner of timestamp contract
- Anonymous accounts that file a timestamp

Example

Media Timestamp - State Data

- A cryptographic hash of the media file
- Date&time, description, type of media, owner contact
- Link to media on internet

Example

Media Timestamp - Restrictions

- Only one entry per hash
- Ownership cannot be changed

Example

Media Timestamp - Exceptions

- Ownership is contested in court of law and fails
- Previous ownership proven

BCDV1011 Design Patterns for Blockchain

Server signed transactions

Server signed transactions

- When to server sign
- Create express app
- Connect to web3
- Get contract object
- Create transaction
- Sign and send

When to server sign

- For automated calls to change state in blockchain
  - Enterprise system access to smart contracts
  - CRON job connection
  - Automated oracle
  - Generated payments
  - Exchange transactions
  - Avoid custodial model
- Requires server app to have access to private key
- May require message queue to handle large volumes

Create Express app

- Create a server application in node.js using express.js framework

```
mkdir project-folder
cd project-folder

npx express-generator

npm install

npm start
```

- Check out the app in the browser

  http://localhost:3000/

Connect to Web3

- Run ganache
- Install npm module web3

```
npm install web3
```

- Add to app.js

```
const Web3 = require('web3');
var Tx = require('ethereumjs-tx').Transaction;
const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:7545"));
web3.eth.getAccounts(console.log);
```

Get contract object

- Need the contract address and ABI - add to app.js

```
const contractAddress = 'YOUR_CONTRACT_ADD';
const ABI = require('YOUR_ABI_FILE');

var TestContract = web3.eth.contract([YOUR_ABI], contractAddress);
```

Create transaction

- Need the contract address, ABI, account, private key and nonce

```
const contractAddress = 'YOUR_CONTRACT_ADD';
const ABI = require('YOUR_ABI_FILE');
const account = '0xACCOUNT_ADDRESS';
const privateKey = Buffer.from('YOUR_PRIVATE_KEY', 'hex');
const newAddress = '0x5aB5E52245Fd4974499aa625709EE1F5A81c8157';
var TestContract = new web3.eth.Contract([YOUR_ABI], contractAddress);
const _data = TestContract.methods.setOwner(_newAddress).encodeABI();

web3.eth.getTransactionCount(account)
.then(nonce => {
   var rawTx = {
      nonce: nonce,
     gasPrice: '0x20000000000',
     gasLimit: '0x27511’,
     to: contractAddress,
     value: 0,
     data: _data
   }
```

Sign and send

- Sign the transaction

```
    var tx = new Tx(rawTx);
    tx.sign(privateKey);

    var serializedTx = tx.serialize();

    web3.eth.sendSignedTransaction('0x' + serializedTx.toString('hex'))
      .on('receipt', console.log);
});
```

Message Queue

- Asynchronous communication
- Queued messages
- Ethereum
- Transaction failures
- Options

Asynchronous communication

- Communication between two parties where responses are not required to be immediate
- Ethereum transactions take time to finalize (5secs-5mins)
- REST API calls are not guaranteed to respond fast
- Handled in Javascript
  - Callback
  - Promise
  - Await

Queued messages

- Producer/consumer mismatch
- Queue up messages and they can be handled when the system is ready
- Hide complexity - just add to queue

Ethereum

- Nonce - transaction count
- Expected that only one transaction per nonce value
- Nonce values increment
- Any mistakes in this and the transaction is rejected
- Does not work well with microservice architecture and scaling

Transaction failures

- Transactions can fail
- Need to requeue
- May need to adjust
- Message queue can avoid losing transactions

Options

- RabbitMQ - good interface
- ActiveMQ - good for scaling
- Kafka - high speed

Key Management

- Custodial/Non-custodial
- Hardware devices
- Backup process
- Key Management
- Social Key Recovery

Custodial/Non-custodial

- Custodial model means the application holds your private key for you
  - Users lose their keys
  - Honeypot
- Non-custodial
  - You control your own - no impersonation
  - You need to manage backups

Hardware keys

- Hardware device that can sign using your private key
- Hardware devices are upgradable to handle different types of signing
- Can be moved around without having to expose your key(s) to the systems you are using
- Clumsy interfaces
- Device can be lost or broken or you lose your device password

Backup process

- Pass phrase
- Paper-based
- Vault
- Biometric access to backup

Key management

- Decentralized
  - MetaMask
  - Hardware keys
  - Wallets
  - Non-custodial
- Centralized
  - Exchanges
  - dApps
  - Custodial
- Distributed
  - Fractional keys

Social key recovery

- Fractional keys shared with others
- Threshold of fractional pieces returned to recover key
- Based on social or business relationships
- Can be used for small data backup too

BCDV1011 Design Patterns for Blockchain

Smart contract security

Smart contract Security

- Smart Contract Security
- Common pitfalls
- Contract audit
- Install IPFS
- IPFS in the browser
- Connecting to the IPFS API server
- Add file to IPFS

Smart Contract Security

- Smart Contracts need extra attention to security
  - Value can be transferred through the contract
  - The contract can hold value
  - The state information in the contract needs to be trusted
  - Only certain accounts or roles can change the state information
- You can’t update your smart contract, just release a new version
- Exchanges will require security audits before including your tokens
- Insurance on smart contracts is nascent and may not be available

Common pitfalls

- The default for functions and state variables is public
  - Explicitly set things to public and private
  - Check all public functions to make sure they do not expose or change private state information by accident
  - Use modifiers and assert/require statements
- Ether transfers can have security problems
  - Consider how ether can get “stuck” in a contract
  - Use reverts and fallbacks
  - Use precise units to avoid rounding confusion

Common pitfalls

- Avoid the temptation to have versioning templates
- The call stack has a limit of 1024 - be careful of recursive calls
- Integers can be unpredictable when combining very large and very small values
- Types that do not occupy 32 bytes can be set to have the “extra” bits set that could possibly cause problems.
- Careful with iterating arrays and running out of gas
- Take compiler warnings serious
- Avoid re-entrancy if you can

Contract auditing

- You may not be able to get a token on an exchange without an audit
- You can’t audit your own contracts
- If you want to provide auditing services you will need insurance
- Stick with the Open Zeppelin code as much as possible
- Keep your contracts **simple and small**

Install IPFS

- Get a copy of the IPFS from https://dist.ipfs.io/#go-ipfs
- Extract the tar.gz file tar -xvf and run the install.sh

```
ipfs init
```

- Check out the quick-start
- In a new terminal window run

```
ipfs daemon
```

- Check out your local interface on http://localhost:5001/ipfs/

Install IPFS

- Get a copy of the IPFS from https://dist.ipfs.io/#go-ipfs
- Extract the tar.gz file tar -xvf and run the install.sh

```
ipfs init
```

- Check out the quick-start
- In a new terminal window run

```
ipfs daemon
```

- Check out your local interface on http://localhost:5001/ipfs/

Configure IPFS

- You need to set your IPFS so that other software can connect

```
ipfs config --json API.HTTPHeaders.Access-Control-Allow-Origin '["*"]'
ipfs config --json API.HTTPHeaders.Access-Control-Allow-Methods '["PUT", "GET", "POST"]
```

Try out IPFS

- Use the web interface
- Upload a file
- Display the reference in your browser by putting in your address bar

    http://localhost:8080/ipfs/your_has_code_here

Connect to IPFS API

create-react-app ipfstest

npm install --save ipfs-http-client

In your App.js

```
const ipfsClient = require('ipfs-http-client')
const ipfs = ipfsClient('http://localhost:5001')

const ver = await ipfs.version()
console.log("IPFS Version=", ver)
```

Connect to IPFS API

- To add a file use add

```
var hash = ""
for await (const result of ipfs.add(this.state.buffer)) {
  console.log(result)
  hash = result.path
}
```

Connect to IPFS API – assignment

- Write a React app that can take an image file and add it to IPFS
- Display the file in the app
- Put the code in your github!