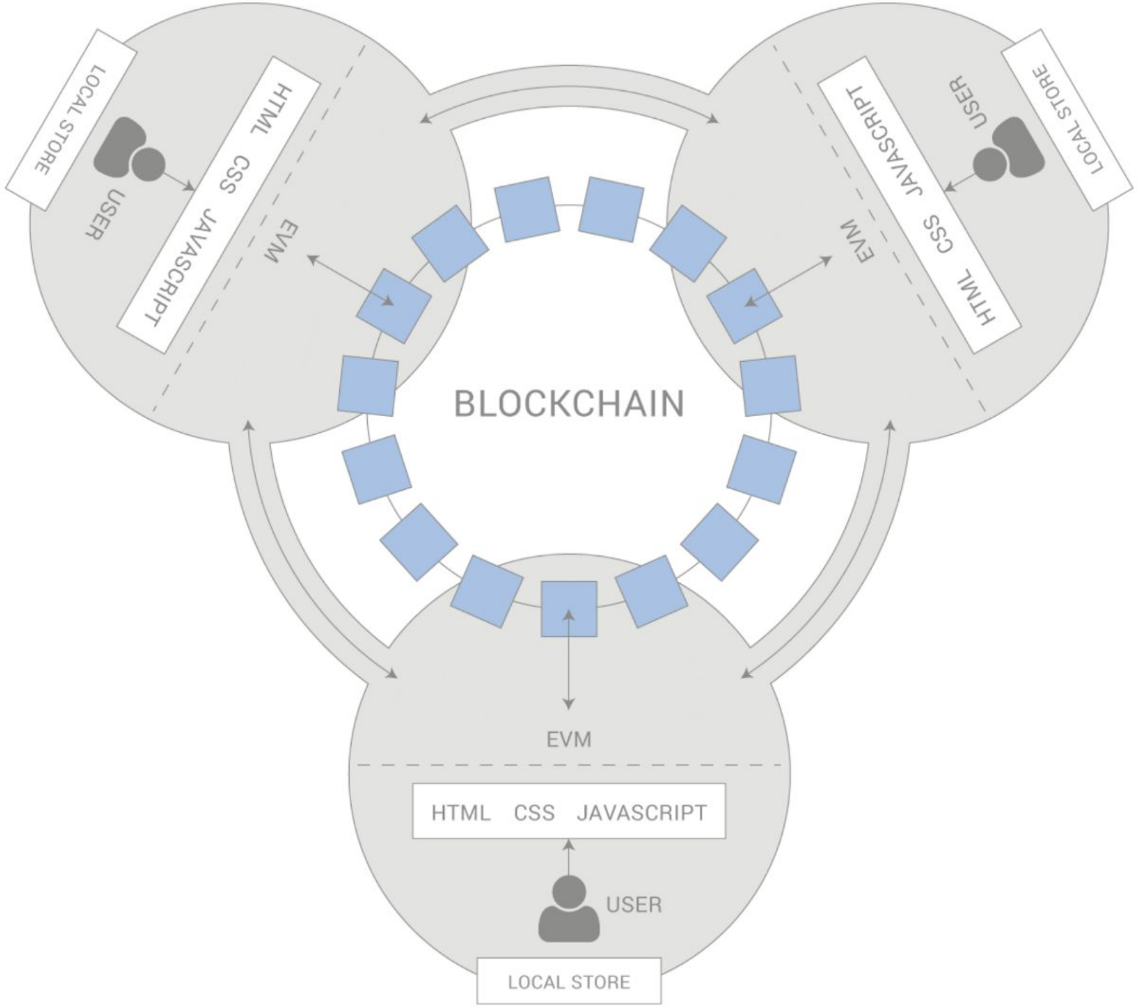
Local and Frontend Development

Setting up a local environment
Writing tests
Creating the frontend
Migrating to a test network

Decentralized Applications

Comparing Application Stacks

	Web 2.0	**Web 3.0 (dApps)**	**Status**
Scalable computation	Amazon EC2	Ethereum, Truebit	In progress
File storage	Amazon S3	IPFS/Filecoin, Storj	In progress
External data	3rd party APIs	Oracles (Augur)	In progress
Monetization	Ads, selling goods	Token model	Ready
Payments	Credit Cards, Paypal	Ethereum, Bitcoin, state channels, 0x	Ready

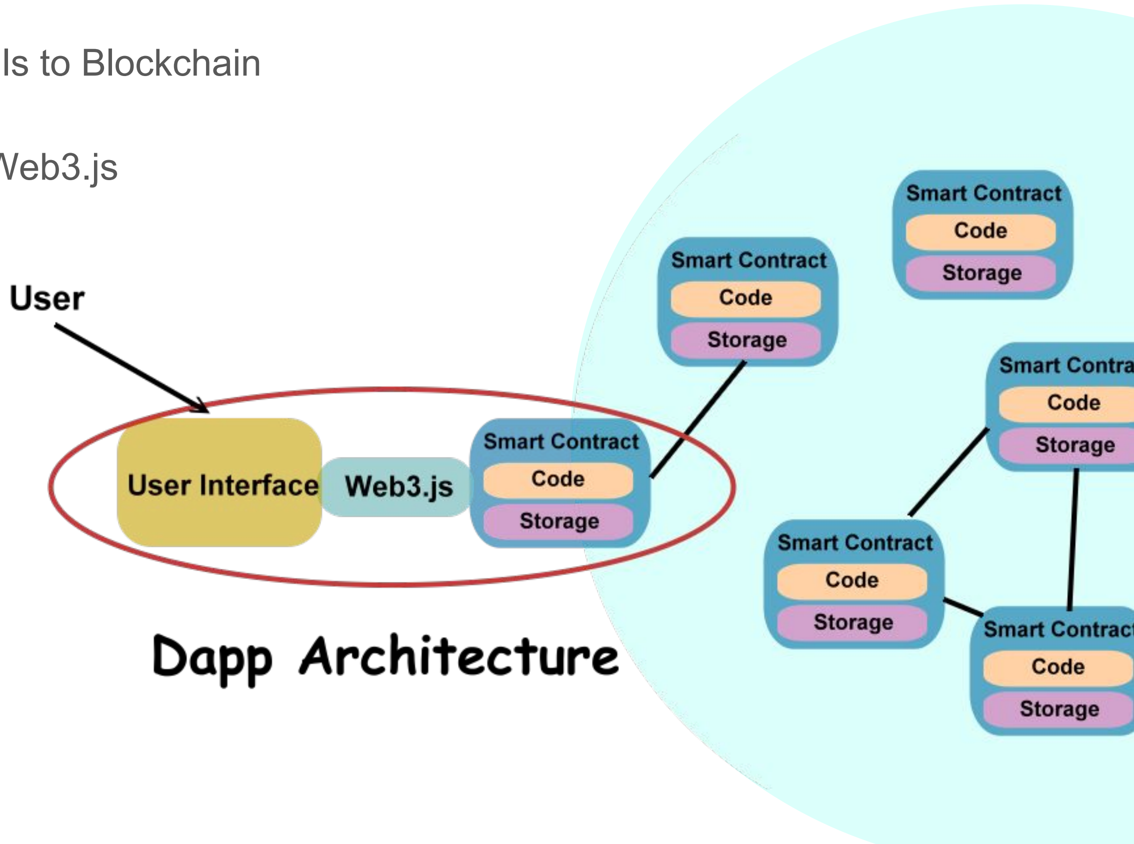

BLOCKCHAIN
LOCAL STORE
USER
HTML CSS JAVASCRIPT
EVM
LOCAL STORE
USER
HTML CSS JAVASCRIPT
EVM
EVM
HTML CSS JAVASCRIPT
USER
LOCAL STORE

Decentralized Application Architecture

Communication is done via JSON RPC calls to Blockchain
Ethereum Foundation created Web3.js
Truffle Contract (Consensys) wrapper for Web3.js

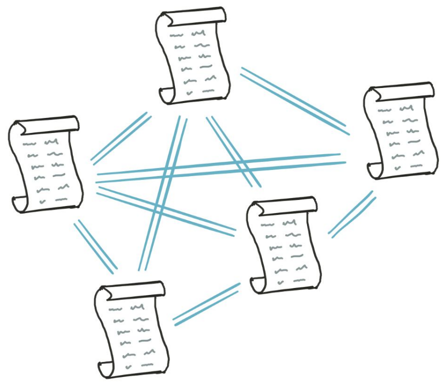
Issues:

- Waiting for blockchain
- Syncing state



Local Development

Setting Up Ganache CLI and Truffle



Recommend running at least 2 terminals

- One for ganache-cli
- One for truffle and tests
- Optional: one for git, npm, local webserver

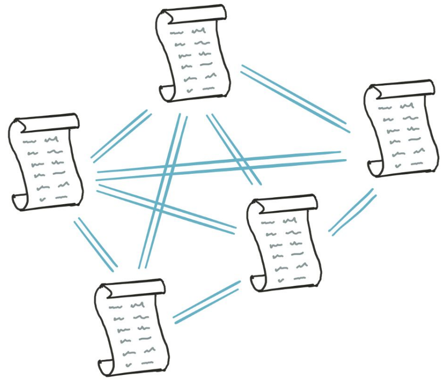
Truffle init is bare bones, no contracts and no deployments

Let’s add BikeShare.sol!

```
//terminal 1
> ganache-cli
//terminal 2
> truffle develop
> compile
> migrate
```

Creating BikeShare.sol locally

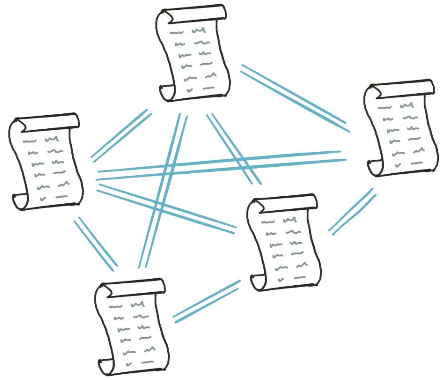
```
//contracts/BikeShare.sol


/* BikeShare.sol from Remix */


//migrations/2_deploy_contracts.js


const BikeShare = artifacts.require('./BikeShare.sol');
module.exports = (deployer) => {
  deployer.deploy(BikeShare);
};
```



Tests

Setting Up Tests

```
//test/bikeshare.js

const BikeShare = artifacts.require('./BikeShare.sol');

contract(BikeShare, function(accounts) {

  let contract;

  it('should be deployed', async () => {
    contract = await BikeShare.deployed();
    assert(contract.address !== undefined, 'Ownable was not deployed');
  });

});
```

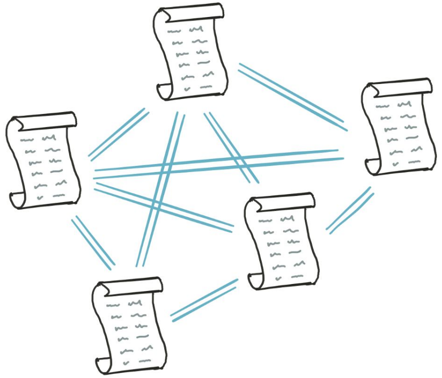


Writing Tests and Coverage

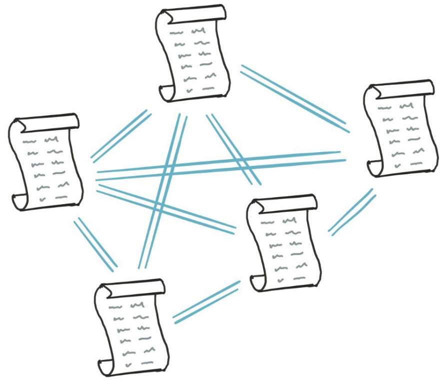
Async and Await are your best friends

Test absolutely everything, even when it fails

```
it('should NOT allow this transaction', async () => {
    //try to update state
    let tx;
    try {
      tx = await contract.iAmNotAllowed({ from: randomAcct });
    } catch (e) {
      //console.log(e);
    }
    assert(tx === undefined, 'transaction occurred when it should NOT');
  });
```



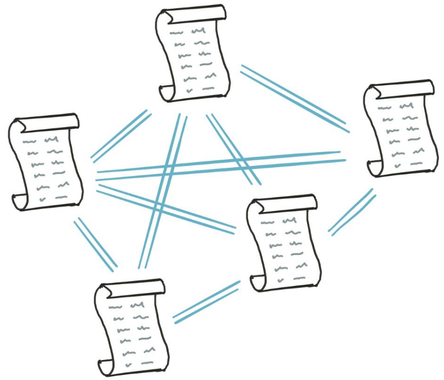
Using BigNumbers in Tests



```
//test/bikeshare.js

...

  const oneEther = web3.toBigNumber(web3.toWei(1, 'ether'));
  it('should be equal', async () => {
    const res = await contract.returnOneEther.call();
    assert(res.equals(oneEther), 'response was not equal to oneEther');
  });

...

});
```

Creating the Frontend

Vanilla JS Architecture



Focus on using TruffleContract and Web3.js

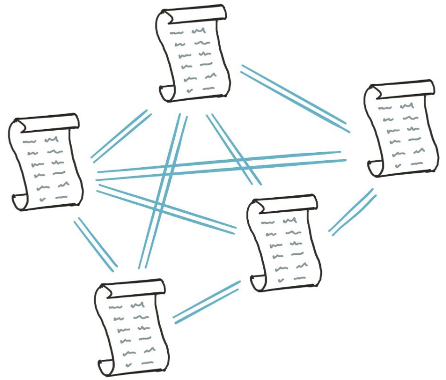
Separate concerns with 2 object literals **App** and **BikeShare**

- **App** will contain our application state, logic and update UI
- **BikeShare** will wrap our contract, currentUser and listen for events

Need the following pieces to connect to our deployed contract

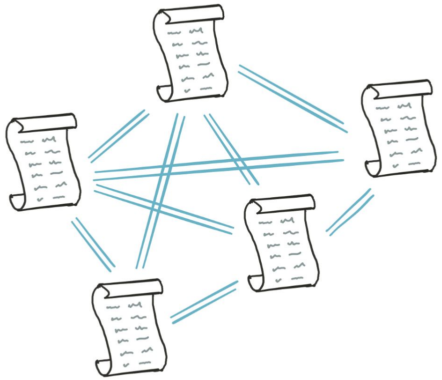
- Compiled contract abi
- Deployed address

Connecting with Truffle Contract



```
let web3;
if (web3 !== undefined) {
  web3 = new Web3(web3.currentProvider);
} else {
  web3 = new Web3(new Web3.providers.HttpProvider(fallbackURL));
}
window.web3 = web3;

const json = await fetch('../../build/contracts/BikeShare.json').then((res) =>
res.json());

const truffleContract = TruffleContract(json);
truffleContract.setProvider(web3.currentProvider);
const contract = address ? contract.at(address) : contract.deployed();
```

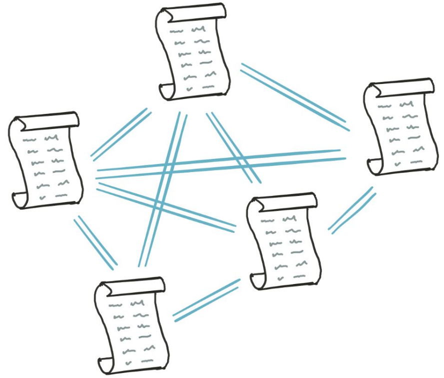
Connecting with Web3



```
let web3;
if (web3 !== undefined) {
  web3 = new Web3(web3.currentProvider);
} else {
  web3 = new Web3(new Web3.providers.HttpProvider(fallbackURL));
}
window.web3 = web3;

const abi = await fetch('../path/to/myContractABI').then((res) => res.json());

let contract;
web3.eth.contract(abi).at(ADDRESS, (err, res) => contract = res);
```

Version 1.0 and above of Web3 uses promises, so you can use async and await

What Network am I on?



For sanity and while developing your frontend, stay local

```
'http://localhost:8545' //ganache-cli (testrpc)
'http://localhost:9545' //truffle develop
```

Connecting MetaMask to Truffle

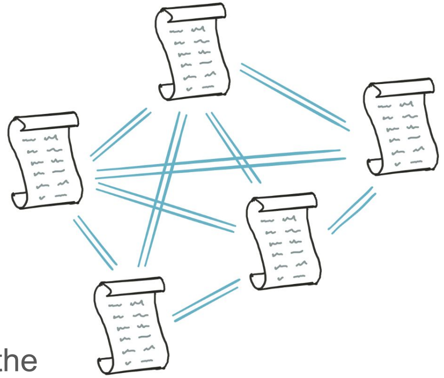
- Use the mnemonic to connect MetaMask with Truffle develop
- `candy maple cake sugar pudding cream honey rich smooth crumble sweet treat`

When connecting to a real Network with MetaMask

- web3 and the provider will be injected into the page
- *** In some JS frameworks, race conditions with web3 can occur

Challenges for BikeShare

Challenges

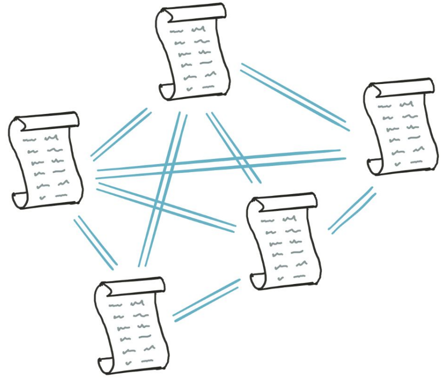


- **Challenge 1:** Refactor the code so we only use 1 mapping
- **Challenge 2:** Bikers should be ablte to transfer credifts to a friend
- **Challenge 3:** As of right now, the Ether is locked in the contract and cannot move, make the Ether transferrable to your address immediately upon receipt

- **Advanced challenge 1:** Decouple the "database" aka mapping into another contract.
- **Advanced challenge 2:** Include an overflow protection library (or inherit from a contract)
- **Advanced challenge 3:** Develop an efficient way to track and store kms per rental, per user
- **Advanced challenge 4:** Add a repair bike bounty where the work can be claimed by a user and verified complete by another user (susceptible to attack?)
- **Advanced challenge 5:** Allow all users to vote on how many credits should be given for a donated bike within a time frame (susceptible to attack?)

Integrating with Frameworks

JS Frameworks + Web3



Highly asynchronous calls to blockchain can be tricky

Make sure you understand the lifecycle of your components

- https://reactjs.org/docs/react-component.html
- https://vuejs.org/v2/guide/instance.html
- …

Personal recommendations / opinions

- Import web3 helpers and libraries in html / app root
- Connect to contract ASAP store instance at root
- Interact + sync with state via Redux / Single Store pattern

Other Frameworks For Web3

Other Libraries and Frameworks for Web3

Frameworks

- Embark
- Truffle

Wrappers

- ethers
- Ethjs
- Web3j (java / android)

Editors

- Remix

Libraries

- Open Zeppelin
- Giveth
- Dappsys… and more!