Summary Notes on Fabric

- Chaincode – handles business logic (smart contract)
- Highly supports Go/Node.js – we'll use Node.js
- Fabric.js is a framework which works with HTML5 (canvas) elements interactive object model, contains an SVG to canvas parser
- Fabric-shim provides APIs for the chaincode to access its state variables, transaction context and call other chaincodes
- To install fabric node_modules:

```
npm install fabric --save


npm install fabric-shim --save
```

- Technical Fabric API and package info:
  https://www.npmjs.com/package/fabric
  http://fabricjs.com/docs/
- ChaincodeInterface classes must implement this interface to utilise chaincode methods, API here:
  https://fabric-shim.github.io/ChaincodeInterface.html
- Example Smart Contract (source: https://fabric-shim.github.io/)

```
const shim = require('fabric-shim');



const Chaincode = class {

    async Init(stub) {

        // use the instantiate input arguments to decide initial chaincode
state values



        // save the initial states

        await stub.putState(key, Buffer.from(aStringValue));



        return shim.success(Buffer.from('Initialized Successfully!'));

    }
```

```
    async Invoke(stub) {

        // use the invoke input arguments to decide intended changes



        // retrieve existing chaincode states

        let oldValue = await stub.getState(key);



        // calculate new state values and saves them

        let newValue = oldValue + delta;

        await stub.putState(key, Buffer.from(newValue));



        return shim.success(Buffer.from(newValue.toString()));

    }

};



                                    ...



// Start the chaincode process and listen for incoming endorsement
requests:

shim.start(new Chaincode());
```

- Policy policy is a function which accepts as input a set of signed data and evaluates successfully, or returns an error because some aspect of the signed data did not satisfy the policy – This might be useful in evaluating whether a block[chain] is non-reputable
- Two types of policy:
    - SignaturePolicy: allows creation of permissible rules using statements (AND, OR, NOutOf). Powerful way of defining clients/users in a network (examples in the link)
    - ImplicitMetaPolicy: Only able to write rules at configuration. This is more useful for defining general rules that are static (don't change very often)
- Example of Policy Code https://hyperledger-fabric.readthedocs.io/en/release-1.2/policies.html:

```
message Policy {
  enum PolicyType {
     UNKNOWN = 0; // Reserved to check for proper initialization
     SIGNATURE = 1;
     MSP = 2;
     IMPLICIT_META = 3;
  }
  int32 type = 1; // For outside implementors, consider the first 1000 types reserved, otherwise
one of PolicyType
  bytes policy = 2;
}
```

- Policies Encoded in common.Policy and defined in fabric/protos/common/policies.proto
- More information about technicalities in that link

- Network Components:
    - Ledger Blockchain + World State
        - World State/Current State (*W*): Key-value pairs that provide latest value of the keys in the transaction log. Beneficial for chaincode as it can access these latest values directly
        - Blockchain: The transaction log. Marks the blocks as valid/invalid transactions based on defined Policies. Hash-linked blocks of transactions
    - Smart Contract/Chaincode: external to the blockchain network. Can be instantiated on 1+ channels and installed on peer nodes
    - Peer Nodes: Network entity/member that runs chaincode containers (Docker?) and maintains a ledger. Can perform read/write operations
    - Ordering Service: Cluster of nodes which can be defined that orders transaction onto the block. FCFS protocol for ordering transactions across channels in the network.
    - Channel: A private blockchain for data confidentiality. Transacting parties/cluster must be authorised/authenticated to be able to interact with.
    - CA (Not necessary for our project unless we get time): PKI-based certificates to network member organisations. Each member of the network gets an ECert to the authorised user