SETU Carlow

HyperLedger Fabric

Auctioning System – Final Report

Student - Oisin Hickey – C00247185

4-15-2023

Table of Contents

Table Of Figures

Abstract

This document outlines the finished project derived from the previously written functional specification. The system implemented is a full-functioning blockchain auctioning system. Throughout this document, I will talk about the development process, the problems I faced, the things I am proud of and the things I would change. I will also demonstrate the functionality of the project.

Introduction

The finished system has been titled "Auction chain". Auction chain syncs with a root node, allowing users to transact currency, create auctions and place bids. Users can also query the system for other information, such as the history of a particular item or Wallet.

Installation Instructions

The entire project source is hosted on a GitHub repository at https://github.com/ironic833/4th-Year-Project-Blockchain-From-Scratch or accessed via the industry showcase. There are two different ways you can run the code on your computer:

Desktop

Download the suitable pre-packaged release from the GitHub releases section to run the application on your desktop. The main compiled release is for Windows.

A simple double click will run the application. Please note that you may need to ensure firewall, proxy and VPN rules do not conflict with the project's P2P system. Linux and macOS editions can be compiled using "npx electron-builder" with the appropriate flags used for platform and architecture. More details here: (*Multi Platform Build - Electron-Builder*, n.d.)

WebApp

The system can also be run as a web application on your local machine. You need NPM installed to run this.

1. Clone the repositories "ServerDemoVersion" branch.
2. Navigate to the root project directory. You should see a package.json file in this directory.
3. Run "npm run dev" to run the web app version of the software.
4. You may also run "npm run dev-peer"; however, this is deprecated after certain updates and changes and may only be used in previous commits.

Description Of Submitted Project

The finished system uses a GUI front-end and an API backend to allow it to interact with a blockchain. Users register to the system using a BIP39 phrase. If they do not have one, they can make a fresh wallet and copy its wallet recovery phrase. Upon authentication to the system, they may view the transaction pool, make an auction, place a bid, send currency, or view the history of any wallet or item ID. An address book is also available, which derives its wallets from blocks on the chain.

Upon a user authenticating onto the chain, their external IP address is broadcasted to other peers. However, the original root node is prioritised as the users' individual firewall and security settings may cause sync problems. The front is built using React and is web-friendly, but I packaged it in EXE format for my distribution version. To package for MacOS, a Mac computer is needed, which I did not have access to. A web version also runs for public usage and demonstration purposes via Heroku.

Backend

The backend of the system uses an API built on express with body-parser, history and parcel bundler installed as middleware. Body-parser handles the translation of the body of node.js requests to endpoints and other web services. History is used to keep track of session history. Parcel bundler is used for packaging the overall web application for production. Finally, request is also installed to handle sending HTTP requests to other web services.

Consensus Rules

Several consensus rules are implemented to allow the chain's validity to be verified on the broadcast. To replace the existing blockchain with a new blockchain, the following conditions must be met:

- The length of the new blockchain must be longer than the current blockchain.
- The block must only contain one reward
- The reward amount must match the mining award amount exactly
- The transactions in the block must be valid, meaning their signatures must be correct
- The new blockchain must be valid according to the isValidChain() method. This means that:
    - ❖ Each blocks difficulty must not be too high or too low
    - ❖ The lastHash property must equal the actual hash of the last block
    - ❖ The block's hash must equal its actual hash value.

If any of these conditions are not met, the existing blockchain will not be replaced with the new one, and an error will be displayed in the console. This is part of why transactions go to the pool first, as it stops entire chains from being invalidated entirely and constantly.

Authentication

I used BIP39(*Bip39 - Npm*, n.d.) for authentication. BIP39 meant the system would not have to store user authentication details on the chain and instead works by taking in a 12 or 24-word phrase and generating the same public and private key from scratch every time.

The code I wrote defines a Wallet class, which represents a valid wallet structure and its methods. In addition, the Wallet class has a constructor that takes an optional mnemonic parameter.

Figure 1: Wallet code used which implements bip39

```
class Wallet {
  constructor(mnemonic) {
      if (mnemonic) {
        const seed = bip39.mnemonicToSeedSync(mnemonic);
        this.masterNode = hdkey.fromMasterSeed(seed);
      } else {
        const seed = crypto.randomBytes(32);
        this.masterNode = hdkey.fromMasterSeed(seed);
        mnemonic = bip39.entropyToMnemonic(seed);
        console.log(`Generated mnemonic: ${mnemonic}`);
      }
      this.balance = STARTING_BALANCE;
      this.path = "m/0'/0'/0'";
      this.node = this.masterNode.derive(this.path);
      this.keyPair = ec.keyFromPrivate(this.node.privateKey);
      this.publicKey = this.keyPair.getPublic().encode('hex');
      this.mnemonic = mnemonic;
  }
```

If a mnemonic is provided, the constructor uses the bip39 package to convert the mnemonic into a seed and then creates a hdkey (*Hdkey - Npm*, n.d.) object using the fromMasterSeed method from the hdkey package. The hdkey object generates a private key and public key pair using the ec (elliptic curve) package. Finally, the publicKey property of the Wallet object is set to the hex-encoded string representation of the derived public key.

If a mnemonic is not provided, the constructor generates a random 32-byte seed using the crypto package, creates a hdkey object using the fromMasterSeed method from the hdkey package, and generates a mnemonic string representation of the seed using the bip39 package, which is outputted to the console.

The Wallet object also has a balance property, initially set to a constant value defined in the config file STARTING_BALANCE.

Finally, the Wallet object has a path property set to a specific derivation path m/0'/0'/0' and a node property derived from the hdkey object using the derived method with the path as the argument. The keyPair property is set to the key pair derived from the node. The mnemonic property is set to the mnemonic passed in as an argument or generated randomly if not provided.

The bip39 package is only focused on translating the phrase to entropy, with hdkey being what creates the Wallet. It is also worth noting that the private key is generated first, and the public key is then derived from it instead of both keys being generated simultaneously.

The derivation path used is m/0'/0'/0'. This path specifies the hierarchical deterministic (HD) wallet derivation path, and it is a specific path within the HD wallet derivation structure defined by BIP (Bitcoin Improvement Proposal) 44.

In this path, m stands for master, and the apostrophes denote hardened derivation, which means that child keys cannot be derived from the parent public key. The 0 values represent the account level, and the 0' value represents the external chain level of the Wallet. The last 0' value represents the index of the first receiving address in the external chain of the Wallet.

I chose this path as I had stumbled across it when researching improvement proposal 39; however, other paths are available. (*What Are BIP39, BIP32, and BIP44? - Vault12*, n.d.)

I originally tried and only used the bip39 package using example code from GitHub (*Bitcoinjs/Bip39: JavaScript Implementation of Bitcoin BIP39: Mnemonic Code for Generating Deterministic Keys*, n.d.). However, I kept encountering errors, and upon implementing hdkey into parts of the code, I eventually achieved a successful output.

I used the test phrase “device true list zone amateur multiply vault guitar category quick traffic call expire long enable rebel forward sport yard unique hobby sense path earth" to derive the public key:

042c5f15fda98db5ce4ab4f9c16b07e18eadcbf749d3ee490a34381e5eaeee52ab384b2434dc0e627df023d0ab4561861234ca75029af6a9695b5cd637487c3276 for testing.

Database Layout

I knew that the block itself would be stored for the database layout, so it must have a consistent, verifiable structure. In each block, there were naturally going to be different kinds of transactions due to different kinds of actions submitted to the chain. Initially, I looked at NoSql, which allows for much flexibility and uses a JSON-like layout. This inspired me to look at JSON and how I could use it to store and send data. Below are some example data:

Figure 2: Sample JSON chain data

```
{
    timestamp: 1,
    lastHash: '-----',
    hash: 'hash-one',
    data: [],
    nonce: 0,
    difficulty: 3
  },
  {
    timestamp: 1681570077951,
    lastHash: 'hash-one',
    hash: '23f0700aafa46fd9151addb5cd3ac09bac420c4858370df3949cc02ce571e7e7',
    data: [ [Object], [Object] ],
    nonce: 4,
    difficulty: 2
  },
  {
    timestamp: 1681570126081,
    lastHash: '23f0700aafa46fd9151addb5cd3ac09bac420c4858370df3949cc02ce571e7e7',
    hash: '4c551ce1610e626b4b91449231d6288de1b40ad11be34a8c08c4abd9397e75cf',
    data: [ [Object], [Object] ],
    nonce: 2,
    difficulty: 1
  },
  {
    timestamp: 1681570348330,
    lastHash: '4c551ce1610e626b4b91449231d6288de1b40ad11be34a8c08c4abd9397e75cf',
    hash: '316e2b990bf066c8e21a9d13300d9cc4291e18fd14c8362ba3cb113ecc6990d2',
    data: [ [Object], [Object] ],
    nonce: 1,
    difficulty: 0
  }
]
```

Endpoints Summary

The backend API uses numerous endpoints to give the front-end its functionality. Most of these endpoints can be broken down into the following subcategories:

- History Endpoints
- Transaction Endpoints
- Wallet Endpoints
- Mine Endpoints
- Block Endpoints

The endpoints used for retrieving item or wallet history from the chain fall into the history endpoints subcategory. These endpoints do not write in any way to the chain and, instead, only retrieve from the chain.

The place bid, create auction, reinitiate auction, end auction, close auction, and transact endpoints are all transaction endpoints. These endpoints function by taking details inputted by a user from the front-end, sending requests to the backends running API, and using the API to write to the chain. The sent transactions are then added to the transaction pool.

The wallet endpoints are the Wallet, Wallet generate mnemonic, my wallet address and wallet info endpoints. The wallet endpoint handles the submission of a previously generated phrase to derive a wallet, and the generation endpoint is for when a user has no valid wallet or phrase. My wallet address and info endpoints are used purely to retrieve a user's current wallet key for use in transactions or to allow the user to identify themselves on the chain.

The mine endpoints comprise the transaction pool and mine transactions endpoints. These are the only two endpoints interacting with the pool and are the gap between valid and invalid data on the chain. The transaction pool endpoint views what transactions are currently in the pool. The mine transaction endpoint strobes the functions which validate the pool, and upon validation, transactions will be added to the chain.

Finally, there are the block endpoints. These are used to show the blocks on the chain as readable data in JSON format. These are the blocks, block's length and block's id endpoints. These are used to display the entire chain data, return the chain's total length and retrieve slices of the overall chain, respectively. The known-addresses endpoint could fall into the history endpoint; however, it better fits here as it returns data from the current iteration of the chain.

Front-end

The front end became more complex in some ways than the back end. When I started the project, I knew I wanted to use React. This was mostly because I wanted to gain experience with it as a library; however, my understanding of React needed to be corrected in hindsight. My initial research led me to believe it was just a software library used to make front-end components, but due to the open-source nature of node and NPM, it has evolved into something more akin to an umbrella of extensions and parts. Upon this realisation, I settled on using the following React libraries:

- react
- react-bootstrap
- react-copy-to-clipboard
- react-dom
- react-router-bootstrap
- react-router-dom

In this project, React highlighted one of the major disadvantages of JavaScript. JavaScript, having started as a scripting language, is extremely basic. Node builds upon this to make it more compatible with desktop platforms. Nevertheless, it has a different level of large, standardised libraries than C++ has. The six packages outlined above were purely just for react front-end.

1. React is a popular JavaScript library for building user interfaces. It is a declarative, efficient, and flexible library that allows developers to easily build complex and dynamic web applications. React's key feature is its ability to manage an application's state, making it easier to build responsive and interactive user interfaces. In addition, it uses a virtual DOM to update the UI efficiently and minimise the number of DOM operations required. (*React*, n.d.)
2. React-Bootstrap is a popular UI library for building responsive and mobile-first web applications using React. It provides a set of pre-built UI components that are easy to

use and customise, including buttons, forms, modals, and more. React-Bootstrap also supports accessibility features like keyboard navigation and screen reader support, making it a great choice for building accessible web applications. (*React-Bootstrap · React-Bootstrap Documentation*, n.d.)

3. React-Copy-to-Clipboard is a small and simple library that provides a React component for copying text to the clipboard. It is a useful tool for creating copy-to-clipboard functionality in your web applications, and it works across different browsers and devices. With React-Copy-to-Clipboard, you can easily add a copy button to any element in your application and enable users to copy text quickly with just one click. (*React-Copy-to-Clipboard - Npm*, n.d.)
4. React-DOM is a package that provides a way to render React components to the browser's DOM (Document Object Model). It is a core part of the React library and is used to render and manage the application's state in the browser. React-DOM provides a simple API for working with the DOM and is optimised for performance and speed.
5. React-Router-Bootstrap is a package that provides integration between React-Router and React-Bootstrap. It allows you to easily create links and navigation elements in your web applications using Bootstrap styling and layout. With React-Router-Bootstrap, you can create dynamic and responsive navigation menus that work seamlessly with React-Router. (*React-Router-Bootstrap - Npm*, n.d.)
6. React-Router-DOM is a popular package for routing in React applications. It provides a declarative API for defining routes and navigating between them. React-Router-DOM supports dynamic routing, lazy loading of components, and server-side rendering, making it a powerful tool for building complex web applications. It is also easy to integrate with other React libraries and UI frameworks.

In summary, React is the base package produced by Facebook, React copy to clipboard, and React DOM extend its base functionality, but to make it more aesthetically pleasing, react-bootstrap is used. This means components can be routed together with react-router, but due to its incompatibility with bootstrap and DOM, it needs react-router-bootstrap and react-router-dom. Some of these packages, like react-router-bootstrap, aren't explicitly called or imported but are used when the web app is compiled.

Front-end Components

Several components were written to be used throughout the application. Some more important ones are used, such as the navbar, newwalletphrase, addressbook and blocks components. React-router is used to map the routes to each Component.

Navbar

The navbar component is used throughout the front-end to allow users to navigate between numerous parts of the project. The components it can access are defined in a router component, and the endpoint is rendered by simply redirecting to the correct GUI page. A no inputs version also exists to prevent users from accessing other pages before login.

newWalletPhrase

The new wallet phrase component is a great example of linking components to a single page. The no input navbar is used along with the walletMnemonic and the phraseBanner components. The walletMnemonic Component is triggered when a user asks for a new wallet. The Component calls to the backend for a phrase which is then rendered to the front end for use. The following code does this:

```
import React, { useState, useEffect } from 'react';
import { Button } from 'react-bootstrap';
import { CopyToClipboard } from 'react-copy-to-clipboard';

function WalletMnemonic() {
  const [walletPhrase, setWalletPhrase] = useState('');

  useEffect(() => {
    fetch('/api/wallet-mnemoic-generate')
    .then(response => response.json())
    .then(data => setWalletPhrase(data));
  }, []);

  const handlePassphraseSubmit = () => {
    fetch(`${document.location.origin}/api/wallet`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ phrase: walletPhrase })
    })
    .then(() => window.location.href = "/");
  }

  return (
    <div className="wallet-phrase">
      <div style={{ columnCount: 4 }}>
        {walletPhrase.split(' ').map((word, index) => (
          <div key={index}>{word}</div>
        ))}
      </div>
      <br />
      <CopyToClipboard text={walletPhrase}>
        <Button variant="danger" size="sm" style={{ margin: '10px' }}>Copy</Button>
      </CopyToClipboard>
      <Button variant="danger" size="sm" onClick={handlePassphraseSubmit}>Login with
this phrase</Button>
    </div>
  );
}

export default WalletMnemonic;
```
Figure 3:Wallet Mnemonic code

The copy to clipboard library is also used here to make it easier for users to note their new Wallet. The phrase banner component is the last Component called for the new wallet phrase code, and it simply renders a warning message to the user to copy down their Wallet, or they cannot recover their account.

addressBook
The addressBook Component queries the known addresses endpoint for any addresses seen on the chain. There is only a little going on with this endpoint other than the basic math it does to result in pagination for the known addresses. These are rendered in card components with a copy button present to make using the data on the page easier.

Blocks
Although similar to the known addresses component, slightly more occurs on the blocks component. Examining the following code:

```
import React, { Component } from 'react';
import { Button } from 'react-bootstrap';
import Block from './Block';
import NavBar from "../Usability/Navbar";
```

```

class Blocks extends Component {
  state = { blocks: [], paginatedId: 1, blocksLength: 0 };

  componentDidMount() {
    fetch(`${document.location.origin}/api/blocks/length`)
      .then(response => response.json())
      .then(json => this.setState({ blocksLength: json }));

    this.fetchPaginatedBlocks(this.state.paginatedId)();
  }

  fetchPaginatedBlocks = paginatedId => () => {
    fetch(`${document.location.origin}/api/blocks/${paginatedId}`)
      .then(response => response.json())
      .then(json => this.setState({ blocks: json }));
  }

  render() {
    console.log('this.state', this.state);

    return (
      <div>
        <NavBar />
        <br />
        <h3>Blocks</h3>
        <hr />
        <br />
        <div>
          {
            [...Array(Math.ceil(this.state.blocksLength/5)).keys()].map(key => {
              const paginatedId = key+1;

              return (
                <span key={key} onClick={this.fetchPaginatedBlocks(paginatedId)}>
                  <Button bsSize="small" variant="danger">
                    {paginatedId}
                  </Button>{' '}
                </span>
              )
            })
          }
        </div>
        {
          this.state.blocks.map(block => {
            return (
              <Block key={block.hash} block={block} />
            );
          })
        }
      </div>
    );
  }
}

export default Blocks;
```
Figure 4: Blocks Component

The code shown works by taking the entire chain length and cutting it up into sections and then querying another endpoint for each section and rendering the returned data in a block component. Each block component checks what form of transaction is present within it and then renders a suitable transaction layout to match the transaction.

Conformance To Project Brief

Looking back on the functional specifications metrics for project success, I have achieved nearly everything in the FURPS and metrics of project success section.

Functionality

Functionality in terms of FURPS defines all of the functions which the software should be able to perform. All of these aims were met in both the backend and front-end functionality.

Blockchain Functionality Within Project	
Facilitate the connection between all nodes	Success
Perform automatic node registry	Success
Store user-made transactions on the chain	Success
Store unverified transactions in a transaction pool	Success
Allow for mining a transaction pool to add its transactions to the chain.	Success
Automatically clear the valid transactions from the pool	Success
Automatically broadcast changes to the pool or chain	Success
Allow for the access or creation of valid wallets to the chain	Success
Use wallet keys to sign transactions as valid	Success

Web Application Functions within the Project	
Allows a user to log in with a previous wallet or generate a new one by interfacing with API	Success
Allow a user to create an auction item	Success
Allow a user to place a bid on an existing item	Success
Allow a user to send another user chain currency	Success
Allow a user to close, reinitiate or end an auction on the chain	Success
Allow a user to search a wallet for its history	Success
Allow a user to search an auction item for its history	Success
Allow a user to mine transactions on their machine	Success

Usability

Usability sets out the ease of use and access the software should maintain. All of these were achieved.

The software should utilise a GUI to communicate data and changes to the user.	Success
Buttons should be used to make navigation and functionality easier for a user.	Success

The software should be able to run on a desktop	Success
The software should ideally have documentation if a user wishes to modify or fork the code.	Success
The system should ideally run as a portable exe	Success
The system GUI should ideally be compatible with multiple desktop platforms or browsers.	Success
Pagination should make large amounts of data easier to navigate	Success

Reliability

Regarding reliability, the application achieved all the milestones I had set out to achieve.

The system should use consensus to verify that the incoming chain is valid	Success
Cryptography should be used to ensure data integrity	Success
The system should have low downtimes due to its P2P structure	Success
The system should not store any user authentication details for security	Success
The system should utilise error-handling methods correctly	Success
Utilise a proof of work system for mining	Success

Performance

Performance defines the speed and efficiency of the software under the hood. Measuring the software's usage on my computer showed that the following was achieved:

The system should be low in resource consumption	Success
The system should be able to run on low-spec devices	Success
The system should be able to run a valid node in a web environment for demonstration purposes.	Success
Mining a block should take around 1000 milliseconds or so on a new chain	Success
Sync chains and pools automatically	Success

Supportability

Supportability should define how broadly the system is supported by other hardware or software, e.g. a more commonly unused browser or operating system. Having written the project in JavaScript, the following milestones have been achieved.

The system should use standard technologies like JavaScript, which various browsers and web technologies should widely support.	Success
The system should be well documented to make changes or improvements easier.	Success
The system should use a clear folder structure to allow users to navigate its code base.	Success

Reflection

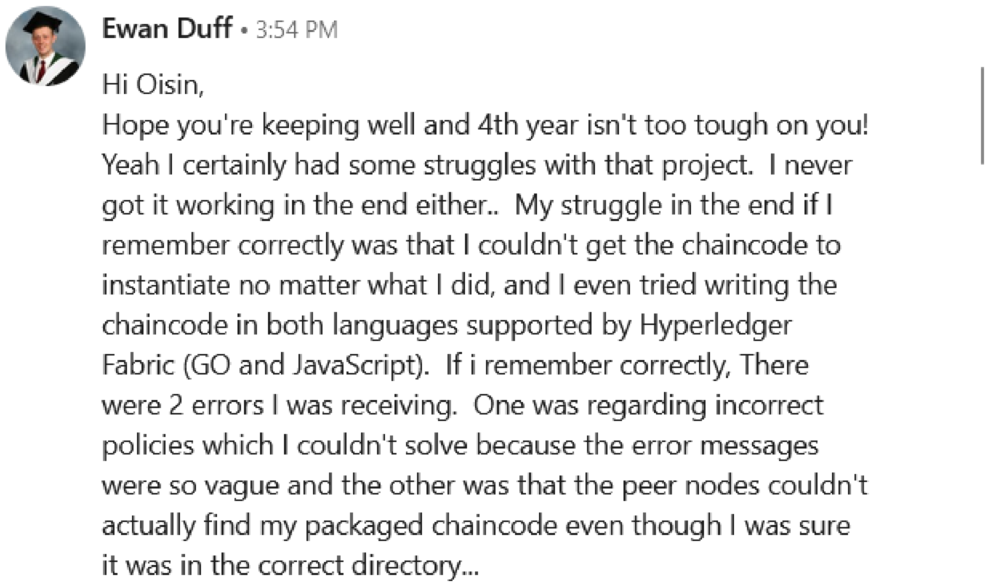
In the following section, I will go over the project as an experience overall and how it came to exist in its current state. Then, I will go over some of the things I wish I had managed to implement and some design hurdles.

Personal Experience

Going into this project, I initially intended to build a blockchain-based dApp. I originally had my mind on a chat application as there are obvious ways that a usually centralised form of software would benefit from a decentralised form of implementation. Unfortunately, the only information I had from the original project assignment sheet was that I had been assigned "A blockchain-based Dapp".

Upon meeting with my supervisor, it was explained to me that I had to make an auction system on HyperLedger Fabric. I did my due diligence to research any potential technology before deciding to use it. Since I had only begun learning Ethereum's solidity, I did not see it as much of an issue.

I tried booting the software and working with it and was plagued with issues and problems and received little to no help from the numerous tutorials online. Finally, I contacted Ewan Duff, who explained he had similar problems.



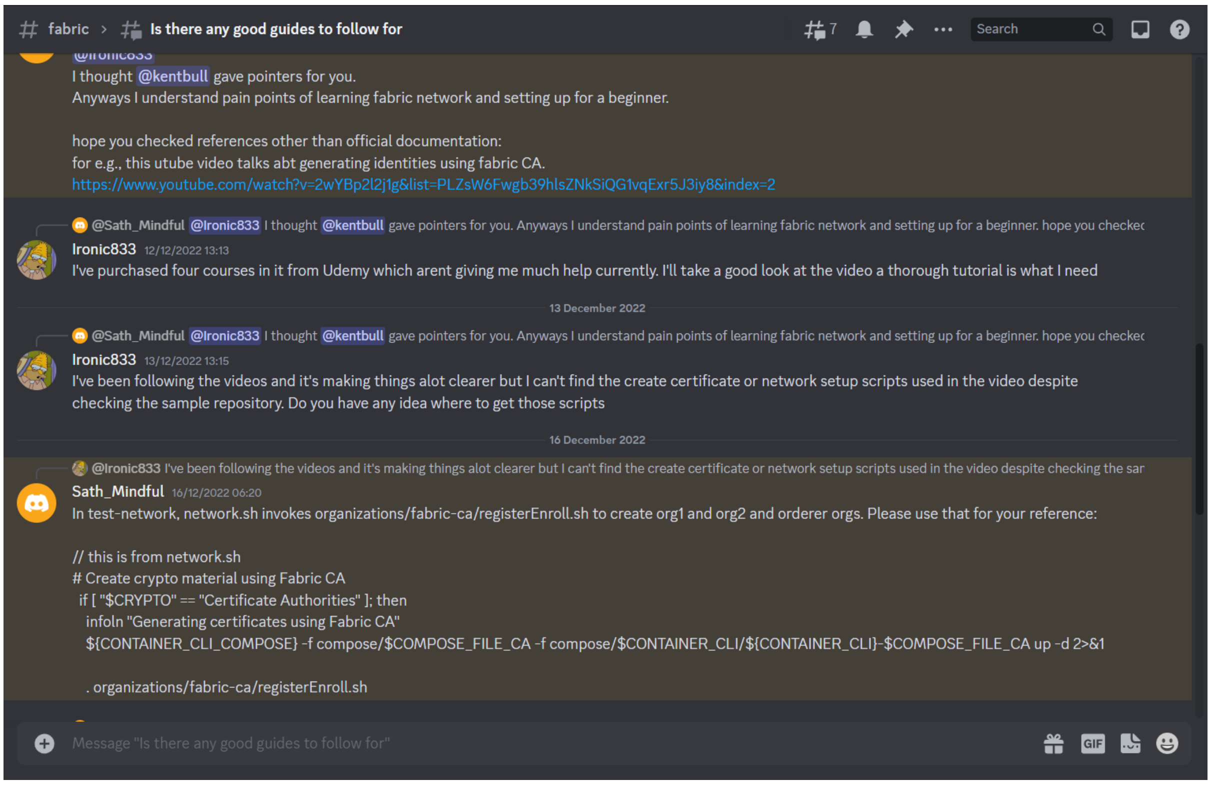
Ewan Duff • 3:54 PM

Hi Oisin,
Hope you're keeping well and 4th year isn't too tough on you!
Yeah I certainly had some struggles with that project. I never got it working in the end either.. My struggle in the end if I remember correctly was that I couldn't get the chaincode to instantiate no matter what I did, and I even tried writing the chaincode in both languages supported by Hyperledger Fabric (GO and JavaScript). If i remember correctly, There were 2 errors I was receiving. One was regarding incorrect policies which I couldn't solve because the error messages were so vague and the other was that the peer nodes couldn't actually find my packaged chaincode even though I was sure it was in the correct directory...

Figure 5: Conversation with Ewan Duff

The first set of issues I hit was with node. Node is the main software, NPM is its package manager, NVM is its version manager, and NPX is its package manager's privileged execution function. Node was complaining that parts of the code were missing in the fabric examples repository I was trying to use. Upon fixing these, I reran the software and was hit with signing issues regarding the certificates and ports.

Thankfully I managed to get into the Hyperledger Fabric Discord community. I had previously put in a post asking for help and finally began to get some replies:


fabric > Is there any good guides to follow for

I thought @kentbull gave pointers for you.
Anyways I understand pain points of learning fabric network and setting up for a beginner.

hope you checked references other than official documentation:
for e.g., this utube video talks abt generating identities using fabric CA.
https://www.youtube.com/watch?v=2wYBp2l2j1g&list=PLZsW6Fwgb39hlsZNkSiQG1vqExr5J3iy8&index=2

@Sath_Mindful @Ironic833 I thought @kentbull gave pointers for you. Anyways I understand pain points of learning fabric network and setting up for a beginner. hope you checkec

Ironic833 12/12/2022 13:13
I've purchased four courses in it from Udemy which arent giving me much help currently. I'll take a good look at the video a thorough tutorial is what I need

13 December 2022

@Sath_Mindful @Ironic833 I thought @kentbull gave pointers for you. Anyways I understand pain points of learning fabric network and setting up for a beginner. hope you checkec

Ironic833 13/12/2022 13:15
I've been following the videos and it's making things alot clearer but I can't find the create certificate or network setup scripts used in the video despite checking the sample repository. Do you have any idea where to get those scripts

16 December 2022

@Ironic833 I've been following the videos and it's making things alot clearer but I can't find the create certificate or network setup scripts used in the video despite checking the sar

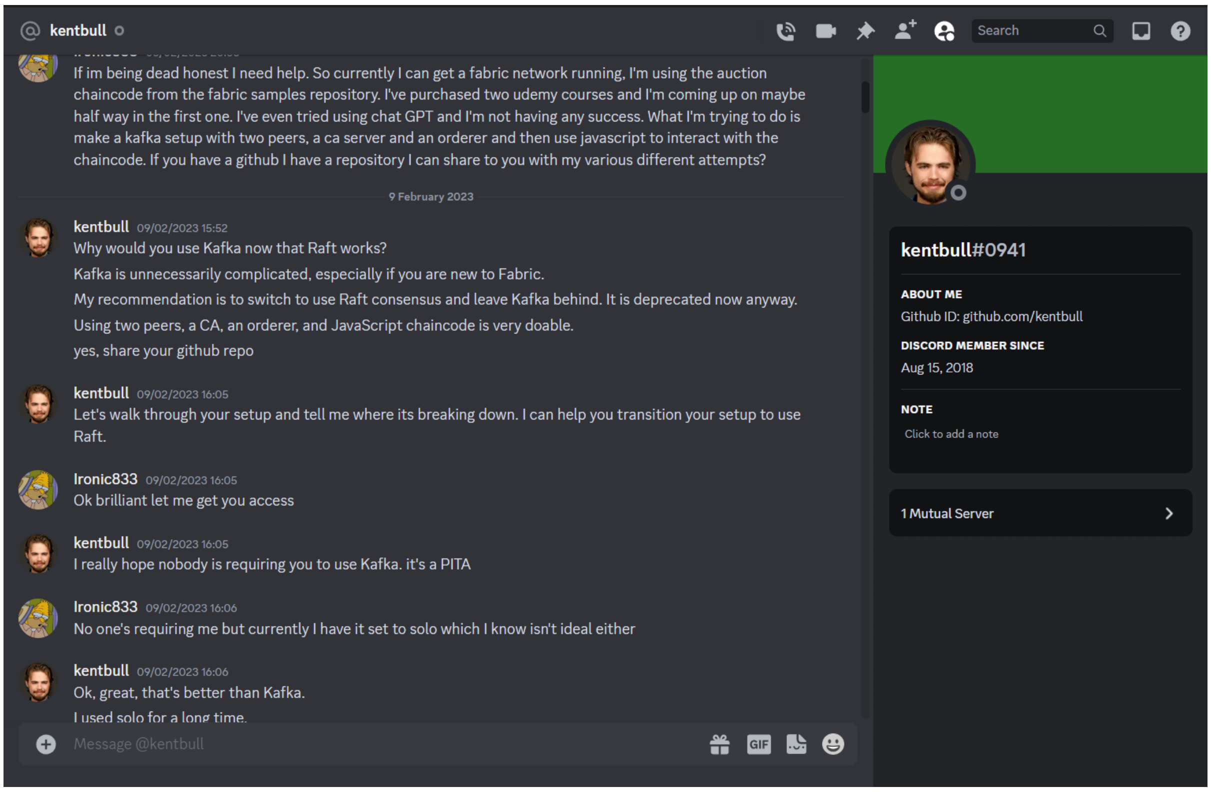
Sath_Mindful 16/12/2022 06:20
In test-network, network.sh invokes organizations/fabric-ca/registerEnroll.sh to create org1 and org2 and orderer orgs. Please use that for your reference:

```
// this is from network.sh
# Create crypto material using Fabric CA
 if [ "$CRYPTO" == "Certificate Authorities" ]; then
  infoln "Generating certificates using Fabric CA"
  ${CONTAINER_CLI_COMPOSE} -f compose/$COMPOSE_FILE_CA -f compose/$CONTAINER_CLI/${CONTAINER_CLI}-$COMPOSE_FILE_CA up -d 2>&1

  . organizations/fabric-ca/registerEnroll.sh
```

Message "Is there any good guides to follow for"

Figure 6: Hyperledger Discord


@ kentbull

If im being dead honest I need help. So currently I can get a fabric network running, I'm using the auction chaincode from the fabric samples repository. I've purchased two udemy courses and I'm coming up on maybe half way in the first one. I've even tried using chat GPT and I'm not having any success. What I'm trying to do is make a kafka setup with two peers, a ca server and an orderer and then use javascript to interact with the chaincode. If you have a github I have a repository I can share to you with my various different attempts?

9 February 2023

kentbull 09/02/2023 15:52
Why would you use Kafka now that Raft works?
Kafka is unnecessarily complicated, especially if you are new to Fabric.
My recommendation is to switch to use Raft consensus and leave Kafka behind. It is deprecated now anyway.
Using two peers, a CA, an orderer, and JavaScript chaincode is very doable.
yes, share your github repo

kentbull 09/02/2023 16:05
Let's walk through your setup and tell me where its breaking down. I can help you transition your setup to use Raft.

Ironic833 09/02/2023 16:05
Ok brilliant let me get you access

kentbull 09/02/2023 16:05
I really hope nobody is requiring you to use Kafka. it's a PITA

Ironic833 09/02/2023 16:06
No one's requiring me but currently I have it set to solo which I know isn't ideal either

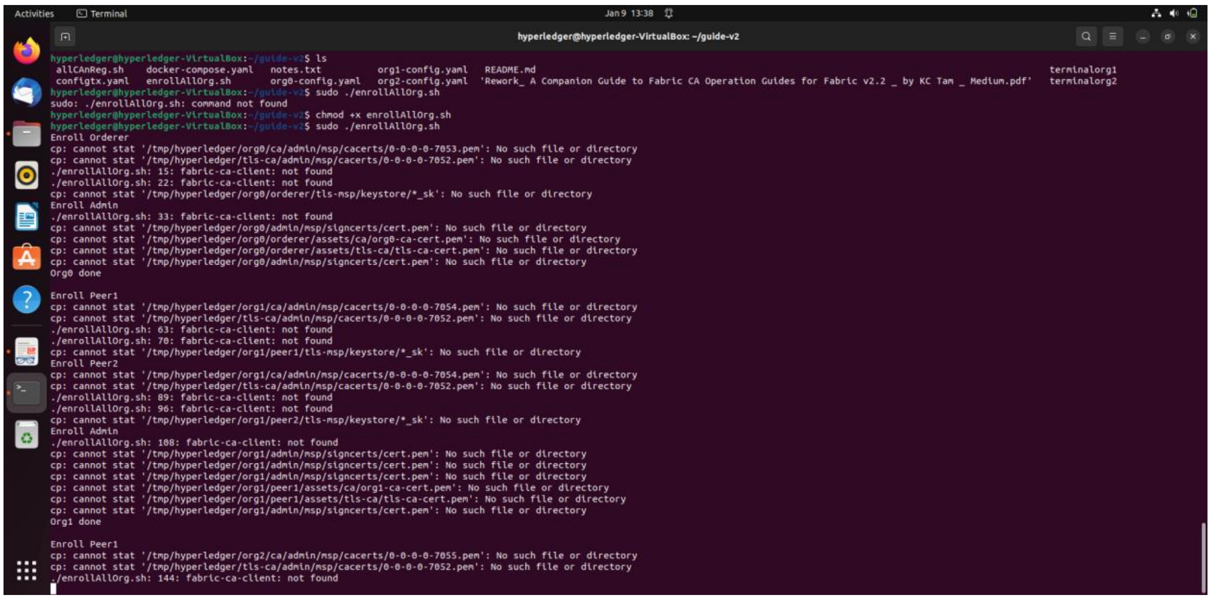
kentbull 09/02/2023 16:06
Ok, great, that's better than Kafka.
I used solo for a long time.

Message @kentbull

kentbull#0941
ABOUT ME
Github ID: github.com/kentbull
DISCORD MEMBER SINCE
Aug 15, 2018
NOTE
Click to add a note
1 Mutual Server

Figure 7: Conversation with Kentbull

I was constantly going back to this thread with new problems. Thankfully I could fix some of them myself until, eventually, the user by the name of Kent Bull was kind enough to reach out to me privately with a fantastic set of pointers. Kent's knowledge of Fabric was extensive. He introduced me to the concept of Fabric CA's, which had yet to be mentioned in brief. CA's are a way to allow the underlying cryptographic components in Fabric to be modular and for new organisations and users to register with them.



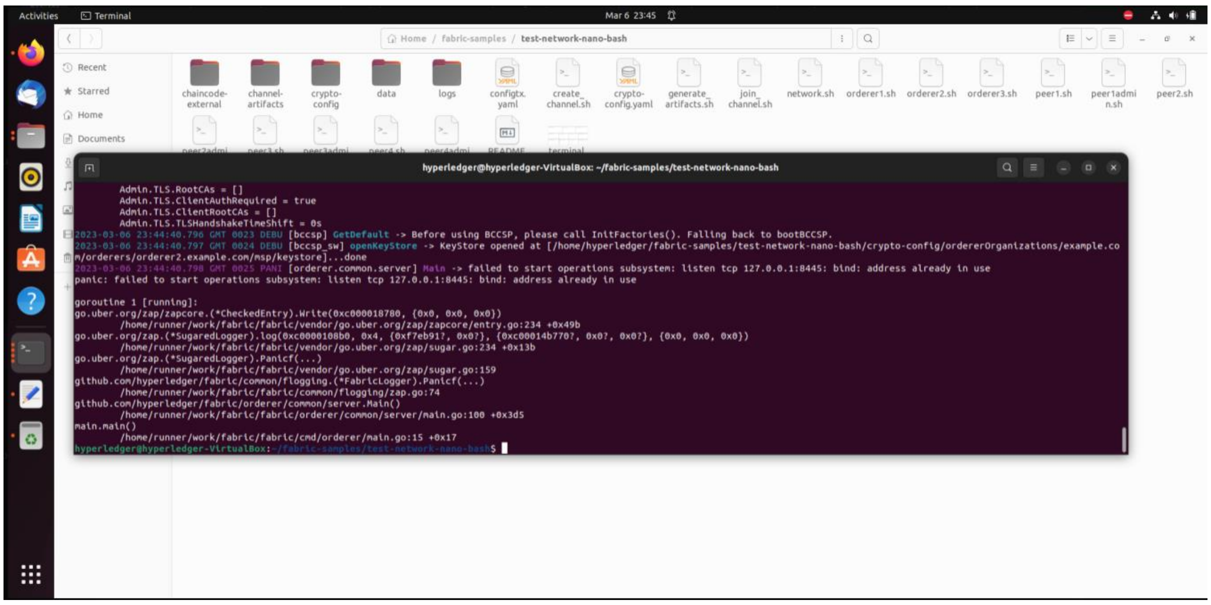
```
hyperledger@hyperledger-VirtualBox:~/guide-v2$ ls
 allCAnReg.sh    docker-compose.yaml   notes.txt         org1-config.yaml   README.md                                                                          terminalorg1
 configtx.yaml   enrollAllOrg.sh       org0-config.yaml  org2-config.yaml  'Rework_ A Companion Guide to Fabric CA Operation Guides for Fabric v2.2 _ by KC Tam _ Medium.pdf'   terminalorg2
hyperledger@hyperledger-VirtualBox:~/guide-v2$ sudo ./enrollAllOrg.sh
sudo: ./enrollAllOrg.sh: command not found
hyperledger@hyperledger-VirtualBox:~/guide-v2$ chmod +x enrollAllOrg.sh
hyperledger@hyperledger-VirtualBox:~/guide-v2$ sudo ./enrollAllOrg.sh
Enroll Orderer
cp: cannot stat '/tmp/hyperledger/org0/ca/admin/msp/cacerts/0-0-0-0-7053.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/tls-ca/admin/msp/cacerts/0-0-0-0-7052.pem': No such file or directory
./enrollAllOrg.sh: 15: fabric-ca-client: not found
./enrollAllOrg.sh: 22: fabric-ca-client: not found
cp: cannot stat '/tmp/hyperledger/org0/orderer/tls-msp/keystore/*_sk': No such file or directory
Enroll Admin
./enrollAllOrg.sh: 33: fabric-ca-client: not found
cp: cannot stat '/tmp/hyperledger/org0/admin/msp/signcerts/cert.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/org0/orderer/assets/ca/org0-ca-cert.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/org0/orderer/assets/tls-ca/tls-ca-cert.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/org0/admin/msp/signcerts/cert.pem': No such file or directory
Org0 done

Enroll Peer1
cp: cannot stat '/tmp/hyperledger/org1/ca/admin/msp/cacerts/0-0-0-0-7054.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/tls-ca/admin/msp/cacerts/0-0-0-0-7052.pem': No such file or directory
./enrollAllOrg.sh: 63: fabric-ca-client: not found
./enrollAllOrg.sh: 70: fabric-ca-client: not found
cp: cannot stat '/tmp/hyperledger/org1/peer1/tls-msp/keystore/*_sk': No such file or directory
Enroll Peer2
cp: cannot stat '/tmp/hyperledger/org1/ca/admin/msp/cacerts/0-0-0-0-7054.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/tls-ca/admin/msp/cacerts/0-0-0-0-7052.pem': No such file or directory
./enrollAllOrg.sh: 89: fabric-ca-client: not found
./enrollAllOrg.sh: 96: fabric-ca-client: not found
cp: cannot stat '/tmp/hyperledger/org1/peer2/tls-msp/keystore/*_sk': No such file or directory
Enroll Admin
./enrollAllOrg.sh: 108: fabric-ca-client: not found
cp: cannot stat '/tmp/hyperledger/org1/admin/msp/signcerts/cert.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/org1/admin/msp/signcerts/cert.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/org1/admin/msp/signcerts/cert.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/org1/peer1/assets/ca/org1-ca-cert.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/org1/peer1/assets/tls-ca/tls-ca-cert.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/org1/admin/msp/signcerts/cert.pem': No such file or directory
Org1 done

Enroll Peer1
cp: cannot stat '/tmp/hyperledger/org2/ca/admin/msp/cacerts/0-0-0-0-7055.pem': No such file or directory
cp: cannot stat '/tmp/hyperledger/tls-ca/admin/msp/cacerts/0-0-0-0-7052.pem': No such file or directory
./enrollAllOrg.sh: 144: fabric-ca-client: not found
```

Figure 8: Cert Errors

Note the time displayed at the top of the screenshot. I was nearly five months deep into the project with little to show for it. Nonetheless, I moved forward with his guidance and help and managed to get some parts of a CA test network spun up. At this point, I had gotten much further than Ewan before me, having spanned up two test networks and some aspects of a fabric CA system. Unfortunately, upon moving through Kent's instructions, I was plagued with new issues:



```
        Admin.TLS.RootCAs = []
        Admin.TLS.ClientAuthRequired = true
        Admin.TLS.ClientRootCAs = []
        Admin.TLS.TLSHandshakeTimeShift = 0s
2023-03-06 23:44:40.796 GMT 0023 DEBU [bccsp] GetDefault -> Before using BCCSP, please call InitFactories(). Falling back to bootBCCSP.
2023-03-06 23:44:40.797 GMT 0024 DEBU [bccsp_sw] openKeyStore -> KeyStore opened at [/home/hyperledger/fabric-samples/test-network-nano-bash/crypto-config/ordererOrganizations/example.co
m/orderers/orderer2.example.com/msp/keystore]...done
2023-03-06 23:44:40.798 GMT 0025 PANI [orderer.common.server] Main -> failed to start operations subsystem: listen tcp 127.0.0.1:8445: bind: address already in use
panic: failed to start operations subsystem: listen tcp 127.0.0.1:8445: bind: address already in use

goroutine 1 [running]:
go.uber.org/zap/zapcore.(*CheckedEntry).Write(0xc000018780, {0x0, 0x0, 0x0})
        /home/runner/work/fabric/fabric/vendor/go.uber.org/zap/zapcore/entry.go:234 +0x49b
go.uber.org/zap.(*SugaredLogger).log(0xc0000108b0, 0x4, {0xf7eb917, 0x0?}, {0xc00014b770?, 0x0?, 0x0?}, {0x0, 0x0, 0x0})
        /home/runner/work/fabric/fabric/vendor/go.uber.org/zap/sugar.go:234 +0x13b
go.uber.org/zap.(*SugaredLogger).Panicf(...)
        /home/runner/work/fabric/fabric/vendor/go.uber.org/zap/sugar.go:159
github.com/hyperledger/fabric/common/flogging.(*FabricLogger).Panicf(...)
        /home/runner/work/fabric/fabric/common/flogging/zap.go:74
github.com/hyperledger/fabric/orderer/common/server.Main()
        /home/runner/work/fabric/fabric/orderer/common/server/main.go:100 +0x3d5
main.main()
        /home/runner/work/fabric/fabric/cmd/orderer/main.go:15 +0x17
hyperledger@hyperledger-VirtualBox:~/fabric-samples/test-network-nano-bash$
```

Figure 9: Port Errors

I tried changing the port numbers used in the underlying script files, which failed to work, as did the other repairs I had attempted.

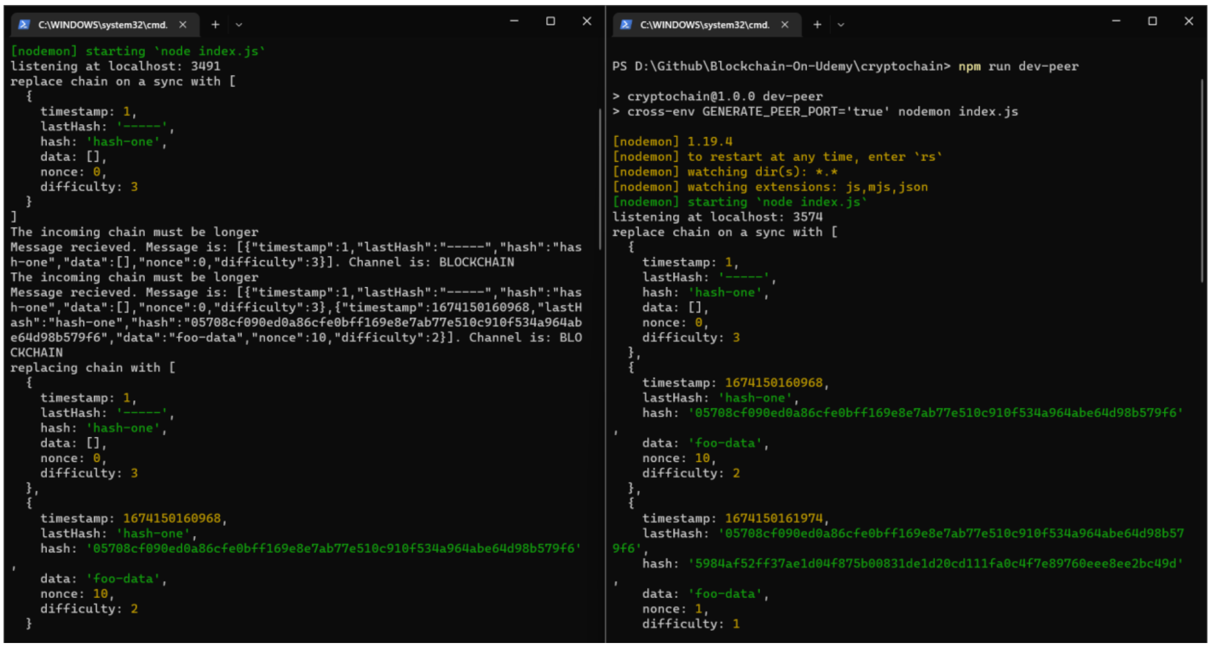
During this time, I had been trying to get my project changed with it constantly blocked behind closed doors, eventually resulting in my escalation of the issue to the department head. Finally, around December, I decided that it did not make sense for me to wait any further, and I had to find my way forward with this project.

I did consult Martin Harrigan, the college's expert in blockchain technologies, and he also put his hands up, saying he couldn't offer me support as it was a technology, he wasn't familiar with.

It's important to mention all of this because it thoroughly affected the output of my project. Even though I began work on a backup solution in December, I was still left with all my documentation to redo and needed a concrete idea of what would happen going forward with the project.

There were two main issues with my solution going forward and how I decided to build it. First, I needed something to handle internode communications effectively to improve socket programming. I chose Redis initially to do this but having installed WSL(Windows Linux Subsystem) on my computer, I could not make it work. Redis ran perfectly, but my host system could not communicate with it. I then decided to move to PubNub.

PubNub worked significantly better with plenty of examples online. I had most of the chain's basic backend built and ran some tests, such as a basic chain replacement:



```
[nodemon] starting `node index.js`
listening at localhost: 3491
replace chain on a sync with [
  {
    timestamp: 1,
    lastHash: '-----',
    hash: 'hash-one',
    data: [],
    nonce: 0,
    difficulty: 3
  }
]
The incoming chain must be longer
Message recieved. Message is: [{"timestamp":1,"lastHash":"-----","hash":"has
h-one","data":[],"nonce":0,"difficulty":3}]. Channel is: BLOCKCHAIN
The incoming chain must be longer
Message recieved. Message is: [{"timestamp":1,"lastHash":"-----","hash":"has
h-one","data":[],"nonce":0,"difficulty":3},{"timestamp":1674150160968,"lastH
ash":"hash-one","hash":"05708cf090ed0a86cfe0bff169e8e7ab77e510c910f534a964ab
e64d98b579f6","data":"foo-data","nonce":10,"difficulty":2}]. Channel is: BLO
CKCHAIN
replacing chain with [
  {
    timestamp: 1,
    lastHash: '-----',
    hash: 'hash-one',
    data: [],
    nonce: 0,
    difficulty: 3
  },
  {
    timestamp: 1674150160968,
    lastHash: 'hash-one',
    hash: '05708cf090ed0a86cfe0bff169e8e7ab77e510c910f534a964abe64d98b579f6'
,
    data: 'foo-data',
    nonce: 10,
    difficulty: 2
  }
```

```
PS D:\Github\Blockchain-On-Udemy\cryptochain> npm run dev-peer

> cryptochain@1.0.0 dev-peer
> cross-env GENERATE_PEER_PORT='true' nodemon index.js

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
listening at localhost: 3574
replace chain on a sync with [
  {
    timestamp: 1,
    lastHash: '-----',
    hash: 'hash-one',
    data: [],
    nonce: 0,
    difficulty: 3
  },
  {
    timestamp: 1674150160968,
    lastHash: 'hash-one',
    hash: '05708cf090ed0a86cfe0bff169e8e7ab77e510c910f534a964abe64d98b579f6'
,
    data: 'foo-data',
    nonce: 10,
    difficulty: 2
  },
  {
    timestamp: 1674150161974,
    lastHash: '05708cf090ed0a86cfe0bff169e8e7ab77e510c910f534a964abe64d98b57
9f6',
    hash: '5984af52ff37ae1d04f875b00831de1d20cd111fa0c4f7e89760eee8ee2bc49d'
,
    data: 'foo-data',
    nonce: 1,
    difficulty: 1
```

Figure 10: Test replacement chain

Till this point in the project, the backend I had built was more akin to a cryptocurrency, and I needed to implement auction functionality. The first place I knew this would need to occur is in the transaction Maps. So I began by implementing an auction-structured transaction Map. However, this introduced me to one of the key issues with PubNub in the following code:

```
const PubNub = require('pubnub');

// Sets the API keys for access to PubNub
const credentials = {
    publishKey: 'pub-c-f0d1aece-b8ce-424d-93c5-057ce943ca34',
    subscribeKey: 'sub-c-9586aba2-bcae-419c-a342-c7556ac00daf',
    secretKey: 'sec-c-NzNiNGUzN2UtN2U3Yi00MzIyLWE2NDUtYWVjNTlmNjA2YWIx'
};

// Sets up our default channels to be used by the system to allow for transactions
const CHANNELS = {
    TEST: 'TEST',
    BLOCKCHAIN: 'BLOCKCHAIN',
    TRANSACTION: 'TRANSACTION'
};

class PubSub {

    constructor({ blockchain, transactionPool, wallet }) {

        this.blockchain = blockchain;

        this.transactionPool = transactionPool;

        this.wallet = wallet;

        this.pubnub = new PubNub(credentials);

        this.pubnub.subscribe({ channels: Object.values(CHANNELS) });

        this.pubnub.addListener(this.listener());

    }

    handleMessage(channel, message){

        console.log(`Message recieved. Message is: ${message}. Channel is: ${channel}`);

        const parsedMessage = JSON.parse(message);

        switch(channel){
            case CHANNELS.BLOCKCHAIN:
                this.blockchain.replaceChain(parsedMessage);
            break;
            case CHANNELS.TRANSACTION:
                if(!this.transactionPool.existingTransaction({ inputAddress: this.wallet.publicKey})){
                    this.transactionPool.setTransaction(parsedMessage);
                }
            break;
            default:
                return;
        }

        if (channel === CHANNELS.BLOCKCHAIN){
            this.blockchain.replaceChain(parsedMessage);
        }
    }

    listener() {

        return {

            message: messageObject => {

                const { channel, message } = messageObject;

                //console.log(`Message received. Channel: ${channel}. Message: ${message}`);

                this.handleMessage( channel, message );
            }
        };
    }

    publish({ channel, message }) {
        this.pubnub.publish({
            channel,
            message,
            meta: {
                uuid: this.pubnub.getUUID()
            }
        });


    }

    broadcastChain(){
        this.publish({
             channel: CHANNELS.BLOCKCHAIN,
             message: JSON.stringify(this.blockchain.chain)
        });
    }

    broadcastTransaction(transaction) {
        this.publish({
            channel: CHANNELS.TRANSACTION,
            message: JSON.stringify(transaction)
        });
    }

}

module.exports = PubSub;
```

Figure 11: Original P2P class

This is the version of the code I used in a previous commit. The issue with this code is that after implementing auctions, the individual chain was too large to send over HTTP requests, with the code giving 413 and 414 errors. Initially, I had solved these using a method I encountered online called message chunking. Unfortunately, I couldn't find any example of this code, but I had a reasonable idea of how it worked. I then moved on to build everything else, the bidding, wallets with bip39 and different endpoints for various auction functions. I had even started a lot of the GUI before this resurfaced as a problem. In the end, I solved it using the following code:

```
const { json } = require('body-parser');
const PubNub = require('pubnub');

// Sets the API keys for access to PubNub
const credentials = {
    publishKey: 'pub-c-cfc32bfe-8f42-4848-8678-53d06f894bc9',
    subscribeKey: 'sub-c-61d414a4-c78e-43af-82eb-7e799ba3080b',
    secretKey: 'sec-c-YzFiYWUzYTMtY2QyOC00YzEyLTk3ZWEtNDU4YjY0MjYyMTc2'
};

// Sets up our default channels to be used by the system to allow for transactions
```

```
const CHANNELS = {
    TEST: 'TEST',
    BLOCKCHAIN: 'BLOCKCHAIN',
    TRANSACTION: 'TRANSACTION',
    PEERS: 'PEERS'
};

class PubSub {
    constructor({ blockchain, peers, transactionPool, wallet }) {
        this.blockchain = blockchain;
        this.transactionPool = transactionPool;
        this.peers = peers;
        this.wallet = wallet;
        this.heldChain = [];
        this.heldPeers = [];
        this.pubnub = new PubNub(credentials);

        this.pubnub.subscribe({ channels: Object.values(CHANNELS) });
        this.pubnub.addListener(this.listener());
    }

    listener() {
        return {
          message: messageObject => {
            const { channel, message } = messageObject;

            let clearFlag = false;

            console.log(`Message received. Channel: ${channel}. Message: 
${JSON.stringify(message)}`);

            switch(channel) {

              case CHANNELS.BLOCKCHAIN:

                if (message !== "chain end") {
                  this.heldChain.push(message);
                } else if (this.heldChain[0] !== undefined) {
                  console.log("chain end added to held chain");
                  this.heldChain.push(message);
                  clearFlag = true;
                }

                // Sort the messages by their timestamps before adding them to the 
heldChain
                this.heldChain.sort((a, b) => {
                  const aTimestamp = new Date(a.timestamp);
                  const bTimestamp = new Date(b.timestamp);
                  return aTimestamp - bTimestamp;
                });

                console.log(JSON.stringify(this.heldChain));

                const organisedChain = this.heldChain.map(message => message.payload);

                console.log("organised chain" + JSON.stringify(organisedChain));

                this.blockchain.replaceChain(organisedChain, true, () => {
                    this.transactionPool.clearBlockchainTransactions(
                      { chain: organisedChain }
                    );
                  });

                break;
              case CHANNELS.TRANSACTION:

```

```
        let parsedMessage = JSON.parse(message);

        if (parsedMessage.input.address !== this.wallet.publicKey){
         this.transactionPool.setTransaction(parsedMessage);
        } else {
         console.log('TRANSACTION broadcast received from self, ignoring..');
        }
        break;

       case CHANNELS.PEERS:

        this.heldPeers.push(message);

        console.log("Held Peers array" + JSON.stringify(this.heldPeers));

        this.heldPeers.sort((a, b) => {
         const aTimestamp = new Date(a.timestamp);
         const bTimestamp = new Date(b.timestamp);
         return aTimestamp - bTimestamp;
        });

        const organisedPeers = this.heldPeers.map(message => message.payload);

        console.log("Peer array " + JSON.stringify(organisedPeers));

        this.peers.updatePeers(organisedPeers);

        break;

       default:
        return;
      }

      if (clearFlag === true) {
       this.heldChain = [];
      }
     }
    }
   }


  peerPublish({ channel, message }){

   const getSize = message => {
    const aString = JSON.stringify(message);
    return (new TextEncoder().encode(aString)).length;
   };

   const messageSize = getSize(message);

   console.log(`\n Publishing message of size ${messageSize} bytes to channel ${channel} \n`);

   const regex = /\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}/;
   const ipAddress = message.match(regex)[0];

   console.log(ipAddress);

   const timestamp = new Date().toISOString();

   this.pubnub.publish({
    channel: channel,
    message: {
      timestamp: timestamp,
      payload: ipAddress
    }
    });
    }

    TransactionPublish({ channel, message }) {
        const getSize = message => {
            const aString = JSON.stringify(message);
            return (new TextEncoder().encode(aString)).length;
        };
        const messageSize = getSize(message);

        console.log(`\n Publishing message of size ${messageSize} bytes to channel ${channel} \n`);

        this.pubnub.publish({
            channel,
            message,
        });
    }

    blockchainPublish({ channel, message }) {
        const getSize = message => {
            const aString = JSON.stringify(message);
            return (new TextEncoder().encode(aString)).length;
        };
        const messageSize = getSize(message);

        console.log(`\n Publishing message of size ${messageSize} bytes to channel ${channel} \n`);

        const timestamp = new Date().toISOString(); // get the current timestamp in ISO format

        this.pubnub.publish({
            channel: channel,
            message: {
                timestamp: timestamp,
                payload: message
            }
        });
    }

   broadcastChain() {

        console.log("Chain Length is: " + this.blockchain.chain.length + "\n");

        for(let chainNumber = 0, chainItem; chainNumber < this.blockchain.chain.length; chainNumber++){

            chainItem = this.blockchain.chain[chainNumber];

            this.blockchainPublish({
                channel: CHANNELS.BLOCKCHAIN,
                message: chainItem
            });

            console.log("Message sent is: " + JSON.stringify(chainItem) + "\n");

        }

    }


    broadcastTransaction(transaction) {

        this.TransactionPublish({
            channel: CHANNELS.TRANSACTION,
            message: JSON.stringify(transaction)
        });
    }

    broadcastPeerMembership(peerRegistration) {

      this.peerPublish({
          channel: CHANNELS.PEERS,
          message: JSON.stringify(peerRegistration)
      });
    }
}

module.exports = PubSub;
```

Figure 12: Final P2P class

My solution to this issue was to break the chain into individual messages, each being sent with a timestamp. This meant each part of the chain left in order from one node, and when received by another node, although disorganised, it could be reassembled using timestamps. From there, the reassembled chain could be put through the consensus algorithms to verify that the chain was valid. Of course, implementing the different auction maps damaged the balancing functionality, but the project's focus was not a cryptocurrency, so this was fine for me.

If I had had more time, I would have used the first few months to learn socket programming, and it would have helped mitigate all of the issues in the P2P functionality. It also would mean the P2P API would not require a paywall.

Personal Learning

I learned much about Hyperledger fabric during this project even though I could not make it work. I'm significantly more proficient in JavaScript, having worked with web and desktop solutions. I have a much greater understanding of JavaScript's packages and restful APIs. I have learned much more about blockchain technologies, how they work under the hood, and why they work.

I have also learned much more about React, a hugely popular library, so getting experience with it was important. I learned a lot about using NPM to handle upstream conflicts, a completely new concept to me. I learned a lot about modern-day cryptographic solutions and where it is going as a field. Finally, I learned much about the full-stack development process and how to build many interesting components frequently used in modern software development.

I learned a lot about politics and how to handle many issues in that field. This was also my first time using any form of unit testing, which was a very informative experience.

Project Review

Once I got going on this project, I enjoyed it. I enjoyed working through the different parts of modern-day software development and seeing the amazing tools at my disposal to build other interesting project ideas I have. I am really happy with how the finished product turned out. I like the desktop implementation, and the BIP39 authentication is something I am especially proud of. I am very happy that I also managed to deploy the code to Heroku. My documentation turned out quite well also, and I'm happy with its format and comprehensiveness.

If I were to do this project again, I would likely use the Ethereum chain as I had planned. I would purchase a compiler server beforehand to allow me to compile for MacOS and Linux. It's annoying that the electron-builder doesn't include that functionality out of the box. Also, I'd like to redo this kind of project in Rust. A memory-safe language would be very beneficial for this form of system.

I would also escalate the problems I had to the department head sooner. Other students did not face issues changing the project as I did, and I could have completed more if I had. There are a lot of interesting ideas and approaches I wanted to explore, which I needed more time to.

Acknowledgements

I want to thank Richard Butler, Keara Barrett, Joseph Keogh and Christopher Staff for their assistance with numerous technical and documentation-based questions and aspects of this project. With them, the various aspects of this project, especially the documentation, got to where it has.

I'd especially like to thank Martin Harrigan for taking the time to answer my questions regarding numerous technical aspects of the project. His expertise was an invaluable resource for me and helped me navigate technical components like user authentication and chain indexing.

I want to thank the individual who uses the username Kent Bull in the HyperLedger discord server. He was a gold mine of information and direction regarding the early stages of this project when I was using HyperLedger and was kind enough to communicate with me across an 8-hour time gap from Utah.

I would also like to thank Ewan Duff, last year's student, who suffered nearly identical problems to me with this project. Between our conversation over LinkedIn and his documentation and repository from last year, I gained fantastic insight into routes to take with this project in its early stages.

I want to thank Ciaran Nolan, Jude Carey and Mark Rodgers in Irish Rail. I had the time to upskill myself during my internship, so I was better prepared for this project, especially in intermediate JavaScript. Their advice and guidance will last me a lifetime in this industry.

I want to thank the 2022 and 2023 software development students, IT management students and cybercrime students for making my time in SETU memorable and for their help and support throughout the project and the college experience overall. From enduring Covid-19 to the stresses of fourth year, they made it bearable.

I'd also like to thank my social circle external to college, such as Kelly Webster, Courtney Pearse, Sadhbh Kenny, Luke Burke, Yorick Van Hazendonk and Dane Hegarty. Their consistent comradery and support were invaluable.

I want to thank Judge Roy Beans and Premiere Hygiene staff, especially Vivian Caroll, Paul Dowling, Sean Keogh, Ken Herbert and Richie Curry. I cannot thank them enough for being a source of humour, employment, and mentorship.

I want to thank my family, without whom I would never have made it to third-level education. Their continuing support and help have been instrumental in getting me where I needed to be.

Finally, I would like to thank my girlfriend, who was a constant source of encouragement, support and advice during this course of this year. She was fantastic, from offering suggestions for documentation to enabling my stress eating.

References

bip39 - npm. (n.d.). Retrieved April 16, 2023, from https://www.npmjs.com/package/bip39

bitcoinjs/bip39: JavaScript implementation of Bitcoin BIP39: Mnemonic code for generating deterministic keys. (n.d.). Retrieved April 16, 2023, from https://github.com/bitcoinjs/bip39

hdkey - npm. (n.d.). Retrieved April 16, 2023, from https://www.npmjs.com/package/hdkey

Multi Platform Build - electron-builder. (n.d.). Retrieved April 15, 2023, from https://www.electron.build/multi-platform-build.html

React. (n.d.). Retrieved April 16, 2023, from https://react.dev/

React-Bootstrap · React-Bootstrap Documentation. (n.d.). Retrieved April 16, 2023, from https://react-bootstrap.github.io/getting-started/introduction

react-copy-to-clipboard - npm. (n.d.). Retrieved April 16, 2023, from https://www.npmjs.com/package/react-copy-to-clipboard

react-router-bootstrap - npm. (n.d.). Retrieved April 16, 2023, from https://www.npmjs.com/package/react-router-bootstrap

What are BIP39, BIP32, and BIP44? - Vault12. (n.d.). Retrieved April 15, 2023, from https://vault12.com/securemycrypto/crypto-security-basics/what-is-bip39/