

CONNECT FOUR

Convenience - Connectivity - Compatibility - Confidentiality



PROJECT BOOK (Using BlockChain)

**Aaron Leppin
Luis Pineda
Abdulrahman Alanazi
James Robinson
Muhammad Alam
Oshioke Oleghe
Ryan Karpinski**

Project Overview

For this project, we wanted to build a solid application that would not only be user-friendly, but also a reliable and a very convenient way to be able to access and add medical records for any party that is authorized to run this application. We completed many hours of research and discussions of opinions for the direction we wanted to go to achieve our end goal. These goals were achieved by building a (mostly complete) python version for a working concept which then was ported to Java to fulfill the requirements that we set for the final product.

Group Responsibilities

Project Manager

Team Member Aaron Leppin

Responsibilities Responsible for the entire project, the Project Manager ensures all other responsibilities, assignments, and deadlines are completed. Other duties include organizing meetings, resolving conflict, and moderating decisions.

Requirements Manager

Team Member Abdulrahman Alanazi

Responsibilities The Requirements Manager is responsible for ensuring the completion of formal requirements for the project. Also, he guides the translation of the project assignment to requirements used to design and build the application.

Planner

Team Member Luis Pineda

Responsibilities	The Planner's main responsibility is creating and managing a project plan that includes assignment of tasks to team members, estimated hours, and completion date. Secondly, the planner communicates progress of the plan to the group and works closely with the project manager to ensure timely task completion.
-------------------------	--

Designer Manager

Team Member	Abdulrahman Alanazi
--------------------	---------------------

Responsibilities	Translating the requirements into a design that can be implemented is the responsibility of the Design Manager. This team member takes the group's ideas and concepts to develop a technical plan.
-------------------------	--

Implementation Manager

Team Member	Ryan Karpinski, Abdulrahman Alanazi
--------------------	-------------------------------------

Responsibilities	The Implementation Manager is responsible for taking the design and translating it into a running product. Technology decisions, breakdown of development tasks, and decisions on which code is merged into the final product are other responsibilities of the Implementation Manager.
-------------------------	---

Collaboration Software Manager

Team Member	Jim Robinson
--------------------	--------------

Responsibilities	The Collaboration Software Manager is responsible for the tools utilized by the team to communicate, plan, and organize tasks for the project. This position works with the Planner and Project Manager to ensure all data is captured and reported on in the appropriate tool.
-------------------------	---

Webmaster

Team Member Aaron Leppin, Abdulrahman Alanazi

Responsibilities The Webmaster is responsible for the website that is the outward facing communications portal for the project. The website includes or links to all major components of the project, including the Time Log, Roles and Responsibilities, Design, Plan, and Implementation.

Documentation Manager

Team Member Oshioke Oleghe

Responsibilities The Project Book is the output for each individual team member of all work completed during the project. This role gathers, organizes, and combines all work product documentation.

Testing Manager

Team Member Jim Robinson, Abdulrahman

Responsibilities The Testing Manager works from the requirements and design documents to develop tests that show all requirements are met. He also develops documentation of these tests.

Presentation Manager

Team Member Mohammad Alam

Responsibilities Primarily responsible for the presentations given to the class, the Presentation Manager gathers documentation, condenses it to a presentable format, and lays out the structure and design for the presentation.

CODE

BlockLedger.java

```
2   This class is responsible for holding list of Block records.
3   The reason we have Block annotated with XML is to marshal and unmarshal data to the network
4   The class has getter and setter methods that will be used by ledger manager mostly
5   */
6   import javax.xml.bind.annotation.XmlElement;
7   import javax.xml.bind.annotation.XmlRootElement;
8   import java.util.ArrayList;
9   import java.util.List;
10
11   @XmlRootElement
12   public class BlockLedger {
13
14
15       List<BlockRecord> BlockRecord;
16
17       public BlockLedger(){
18           this.BlockRecord = new ArrayList<BlockRecord>();
19       }
20
21       public List<BlockRecord> getBlockRecord(){
22           return BlockRecord;
23       }
24
25       @XmlElement(name = "blockRecord")
26       public void setBlockRecord(List<BlockRecord> list){
27           this.BlockRecord = list;
```

BlockRecord.Java

```
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

/*
This class is responsible to represent a block records
it has XML annotation so Formatter class can marshal and marshal this class.
This class will be used as an elements of list that BlockLedger class has
This class has getters and setter to modify a block record
*/
@XmlRootElement
class BlockRecord {

    String Seed;
    String BlockNum;
    String SHA256String;
    String SignedSHA256;
    String TimestampVerification;
    String VerificationProcessID;
    String DataHash;
    String TimestampEntry;
    String PreviousHash;
    String SignedBlockID;
    String CreatingProcessID;
    String BlockID;
    String SSNum;
    String Fname;
    String Lname;
    String DOB;
    String Diag;
    String Treat;
    String Rx;
    String Note;
    String Provider;
```

BlocksQueue.Java

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.UUID;

/*
This class is responsible for managing unverified blocks
it suppose to keep track of what block has been not verified, so it works as queue
for entered unverified block.
This class can do:
- receive medical entry and create a block record object.
- once block record object is made, it will be added to toDo list which will by consumed by unVerified block manager.
- it follow FIFO algorithm for giving next unverified block.
- it knows it size (how many unVerified block it has)
*/
public class BlocksQueue {
    Node node;
    private Hash hash;
    private List<BlockRecord> toDo;

    public BlocksQueue(Node node) {
        this.node = node;
        hash = new Hash();
        toDo = new ArrayList<BlockRecord>();
    }
}
```

Hash.Java

```
import javax.xml.bind.DatatypeConverter;
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

/*
This class is responsible to do hash and return hashString
It can do:
- return SHA256 string for any given string.
*/
public class Hash {
    MessageDigest MD = null;
    public Hash(){
        try {
            MD = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            System.out.println("error in initializing MD");
        }
    }

    public String doHash(String s){

        byte[] bytesHash;
        String hashedString = null;
        try {
            bytesHash = MD.digest(s.getBytes("UTF-8"));
            hashedString = DatatypeConverter.printHexBinary(bytesHash);
        } catch (UnsupportedEncodingException e) {
            System.out.println("error in hashing string");
        }

        return hashedString;
    }
}
```


WebServer.Java

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;
import java.util.List;

/*
This class is responsible to run as web server to add GUI functionality.
It can be used for the following purposes:
- adding new medical entry
- displaying ledger (list of verified blocks)
- search the ledger for specific searching criteria
- display active nodes on the blockchain network

it can do:
- handle POST request from browser for adding new block
- handle GET request from browser to viewing the ledger
- manage the flow of webpages the should be displayed to the user.
*/
public class WebServer implements Runnable{
    Node node;
    public WebServer(Node node) {
        this.node = node;
    }
}
```

IMPLEMENTATION (Abdul, Ryan)

We started writing this project completely using python primarily for its capability for rapid development and extensive libraries, but we decided to head in a direction that included using a web-interface. We felt that Java would be the better language to handle the web-interface, however, we started this phase much later in development to be able to completely switch from one language to another. With these decisions and realizations, we continued writing the project in python while also using the python code as a reference to re-write the project in Java. As we arrived closer to our deadline we were confident enough in the Java code that we were able to stop development on the python code. If we were to have decided to not use Java and cut down our development speed, we could have possibly had much more features such as an application GUI or even a mobile application, however, I think that we followed the right path and pursuing the parallel project implementation because of how well the Java application felt in its final phases, especially with the rigidity of the networking system that was developed. If we had enough time we were discussing implementing interprocess communication for the networking part of the Java application to be usable for the unfinished portion of the python application so we would have two fully working and capable applications available for users to decide which one they prefer.

Networking

Peer-to-peer networking was probably the most difficult part in implementation. We wanted a true peer-to-peer system, so we had to find a way to allow nodes to join and leave the network at will while keeping their workloads and ledger synchronized with all other nodes in the network. We discussed how we wanted to lay the framework for the network for many hours and created several different designs and protocols for handling communication. The final two choices we had were to either use a dual thread system or a quad thread system. The dual thread system would create other threads if necessary to handle file transfers while still listening for communications for a block being cracked and would use a flag based system for communication (i.e. Unix-like terminal applications). As for the quad thread system, it would have static threads that are dedicated for handling file download, file upload, receiving another node's message of completion and sending out a message of completion, and handling the connection of nodes to the network. We decided to move forward with the quad thread system because we felt that it was a much more robust way of handling networking. This decision was made after both forms of networking were written and tested to see which one was better. The dual thread system would have possibly experienced some networking issues with communication since the full use of flags was not fully implemented for this test, these issues were unlikely, but we did not want to chance having problems while handling medical data.

Working with a Living Design

We spent a lot of time constantly returning back to design as this was implemented and changing aspects we originally designed because we felt that we could improve optimization to create a more efficient and less resource intensive application. The process that generally happened for each part of the application was create a design for a part we were working on, implement the design, gain insight and knowledge, and revisit the design of the part. This cycle continued for each part until we felt that the design was correct for the direction we wanted to take this project, and then we would move on to the next part and follow the same cycle. Sometimes we would have to change the previous part worked on, however, this cycle we followed helped us create a bullet-proof application that can easily be maintained and expanded on. This style was frustrating at times because of how much written code would either have to be changed or removed and written again. The benefits of this style was it always allowed us to design and implement new ideas that we came up with while working towards the final design and implementation for the part.

Research

The many hours spent research was used to gain a deeper understanding of how blockchain technology works, how to handle peer-to-peer, libraries that best suited our needs, and other utilities to make this application the best we could in the time given. The blockchain portion of research was to make sure we 100% understood how this technology worked so we could lay out the design. For peer-to-peer networking we learned how to use the socket programming libraries and be able to work with them to develop our network to allow nodes to join and leave the network as they please without any interruption to the work that is being done on the nodes in the network and to be able to transfer files, such as the ledger, to make sure everyone in the network has the same information that has been verified. Research was also part of the reason we switched to Java because of tools like TomCat that would allow us to interface a Java program with a web-interface fairly easily without have to make a special version of the application just for the web-interface. The research we put into this project lead to a multitude of different ideas we could choose from and work with, given that many of them did not make it into implementation, some of the ideas made it into our design and implementation phases and are showcased in the robust nature of our networking and the features users have access to.

Language Choices

We debated different ideas of using python, Java, C++, and C to write this application. At first we started with python since it is a powerful rapid development language that also supports multiple paradigms of programming. Then as we progressed we discussed Java a little bit for the web application, but were unsure if we wanted to head that route and would revisit the idea after we were closer to a finalized design, but at the same time, discussed also using C++ or

C for the mining algorithm since they are much more efficient languages that would be able to utilize multithreading to a much greater level while also having a stronger single thread performance. As we worked through the design we decided that we did not want to use the lower level languages at this time because of time constraints and the introduction of the parallel projects in order to be able to support a web-interface.

Early Development

With early development being with Python, this is the time that we came up with some of our first ideas. Such as our early design for mining blocks we developed a way of creating random, yet structured, data to be hashed to verify the data.

```
class Crack:
    def __init__(self, p_hash='0'):
        """Initializes the Crack data structure for randomly
        generating information to mine blocks"""
        self.index = str(random.randint(0,1000)) # index of the block that makes it unique
        self.stamp = self.date_stamp() # contains the timestamp of the block
        self.p_hash = p_hash
        self.pTransaction = self.check_phash(self.p_hash) # contains the previous transaction hash
        self.ssn = self.make_ssn() # calls make_ssn to generate a random ssn
        self.fname = ".join(random.choice(string.ascii_letters) for _ in range(random.randint(1,15))) # randomly
generates a name
        self.lname = ".join(random.choice(string.ascii_letters) for _ in range(random.randint(1,15)))
        self.dob = self.date_stamp() # call to randomly build a date of birth
        self.diag = ".join(random.choice(string.ascii_letters) for _ in range(random.randint(1,15))) # creates a
diagnosis
        self.rx = ".join(random.choice(string.ascii_letters) for _ in range(random.randint(1,40))) # creates a
prescription
        self.tx = ".join(random.choice(string.ascii_letters) for _ in range(random.randint(1,15))) # creates a
treatment
        self.transaction = self.index + self.stamp + self.fname + '\
        + self.lname + ' + self.dob + ' + self.ssn + ' \
        + self.diag + ' + self.tx + ' + self.rx + ' + self.pTransaction # full set of transaction to be added
to the ledger if cracked
        self.content = self.index + self.stamp + self.fname + '\
        + self.lname + ' + self.dob + ' + self.ssn + ' \
        + self.diag + ' + self.tx + ' + self.rx + self.pTransaction # content that will be hashed and
verified
        self.final_hash = self.package_block() # final hash to be check if it contains the correct contents
```

As mentioned, this was one of the earlier attempts at mining blocks. This piece of code creates a block that is identical to one that would be added to the chain, however, what it does is randomly generates each field for the data that would normally be added to the chain and generates the final hash that will be used for comparison in the function that calls the class to be able to check to see if we got the data correct. Other items we tested were storing the chain to disk when the program is exiting and going offline. This function was kept in the chain class and we thought would be a good idea for scalability so there would not need be a need for full file transfers, just some removals or additions on to the existing chain file.

```
def write_chain(self):
    """writes the chain to a txt file for offline storage"""
    save = open(self.default_chain, "w+") # opens the chain.txt file
    chain_string = " # accumulator variable to hold the contents of the string to be written
    for block in range(self.index, self.length()): # loops through the chain
        chain_string += self.chain[block] # adds the block in the chain to the accumulator variable
    save.write(chain_string) # writes the string to the chain.txt file for offline storage
    save.close()
```

This file that is saved could then repopulate the chain upon launching the application again using the populate_chain function in the chain class.

```
def populate_chain(self):
    """function used to populate the chain with the data from chain.txt"""
    if os.path.exists(self.default_chain): # checks to make sure that the file exists
        file = open(self.default_chain) # opens the chain.txt
        items = file.read() # reads the file
        items = items.split() # splits the string into a list
        file.close()
        for item in items:
            self.add_to(item) # adds items to the chain from the stored chain file
        return 'done' # returns done when file has been fully parsed
    else:
        return 'empty' # returns empty when the file is not found
```

The part that we worked on the most was the networking because that is the part of distributed computing that makes the computing able to be distributed.

```
def server_loop(self):
    s = self.makeserversocket(self.serverport)
    s.settimeout(2)
    self.__debug('Server started: {} ({}:{}'.format(self.id, self.host, self.port))
    while not self.shutdown:
        try:
            self.__debug('Listening for connections...')
            sock, addr = s.accept()
            sock.settimeout(None)
            t = threading.Thread(target=self.__handlepeer, args=[sock])
            t.start()
        except KeyboardInterrupt:
            print('KeyboardInterrupt: stopping mainloop')
            self.shutdown = True
            continue
    except:
        if self.debug:
```

```
        traceback.print_exc()
        continue
    self.__debug('Main loop exiting')
    s.close()
```

This being an early attempt at creating the server portion of the node class before the use of flags so the server would be able to differentiate the data that is incoming. The use of flags was never able to be implemented purely because of time constraints and by this time we had the Java networking fully functional

Requirement (Abdul)

Understanding the problem with medical records, we are solving:

1. Fax and snail mail are difficult - transfer or mobility issue
2. Access medical records from another office - accessibility or availability issue
3. Patients do not have good control - authorization issue
4. Insurance companies cannot investigate fraud. Verification or tracing back issue

The solution can be Blockchain system that meet these general requirement:

1. Find a way to transfer data from one place to another with confidentiality and integrity, which means data can be transferred from one node to another in encrypted format so no one can know view it if they are not supposed to do. Also, during the transferring process, the data cannot be modified or manipulated. For example, the int 123 need to be transferred from point A to B and during this transferring process, the int 123 cannot be viewed publicly. Also we need to make sure that the point B received the exact data that point A sent which it the int 123 for this example.
2. The data must be available all time, and there is no only one center for the medical records ledger. The ledger must be shared among all nodes.
3. Any valid record must have unique ID.
4. The ID of any valid records is generated by the time the node or transaction is created.
5. The system must be able to handle any change on the scalability.
6. There must be an Interface for the input and display of patient event data.
 1. Web interface
 2. Console interface
7. The interface need to (at least):
 1. Have searching functionality
 2. Able to view records
 3. Accept new record entry from user.
8. A way to create a record that works as correction for previously entered record.
9. If a correction record is made for another record, there must be a way that the old record can point forward to the correction record.
10. Once a record is made, it cannot be removed.

A Block should consist of:

1. initial / previous block verification ID
2. Transaction (medical records)
3. Seed
4. Verification result of hashing all three 1,2,3

Network:

Since we are dealing with peer-2-peer network, every node will be server and client at the same time.

Node:

As server:

- Add new node to network.
- Do Work
- Receive update
- Receive verified block
- Web server

As Client:

- Interact with user via CLI
- Multicast update
- Join network.
- Search record
- View records
- Add new records

Requirements Matrix (version 5)

Update 10/29/2018 Link to Testing Matrix						
Number	Revision	Requirement Description	Bad-client attack	Bad-dev attack	Approved?	Sign-off name
1	5	block components should be hashed with SHA256 standards	Non	Non	Yes	Abdul 10/16
2.1	5	Each block must have unique Block ID	Non	Non	Yes	Abdul 10/16
2.2	5	The Block ID must be generated by the time block is initially made.	Non	Non	Yes	Luis 10/25
2.3	5	Block cannot be removed from the ledger after it is been verified and added to chain	Non	Non	Yes	Abdul 10/16
2.4	5	A new block must be created instead of correcting a block already in the chain	Non	Non	Yes	Luis 10/16
2.5	5	Corrected block must reference block that has the mistaken info while not changing the block (by blockID)	Non	Non	Yes	Abdul 10/16
2.6	5	Block must be timestamped when created	Non	Non	Yes	Muhammad 10/20

2.7	5	Timestamp is stored in block when cracked to confirm which node is determined to be the cracking node (winner)	Non	Non	Yes	Luis 10/25
3.1	5	There must be updated ledger stored in server / master node side, and nodes should keep synchronized.	Non	Non	Non	Abdul 10/29
3.2	5	The system must be able to handle more than 2 nodes.	Non	Non	Yes	Jim 10/23
4.1	5	There must be a Web Interface for the input and display of patient event data	Non	Non	Yes	Abdul 10/16
4.2	5	There must be a Console Interface for the input and display of patient event data	Non	Non	Yes	Oshioke 10/25
4.3	5	Console Interface - Add block command	Non	Non	Yes	Oshioke 10/25
4.4	5	Console Interface - Display ledger command	Non	Non	Yes	Oshioke 10/25
4.5	5	Web and CLI - List all event records by date of event.	Non	Non	Yes	Aaron 10/18
4.6	5	Web and CLI - List all event records for a specific patient, by date of event.	Non	Non	Yes	Aaron 10/18

4.7	5	Web and CLI - List all event records by provider by date of event.	Non	Non	Yes	Aaron 10/18
4.8	5	Web and CLI - List all event records within a certain date window, by date of event.	Non	Non	Yes	Aaron 10/18
4.9	5	Web and CLI - List all prescriptions for a patient, with provider and by date.	Non	Non	Yes	Aaron 10/18
5	5	All data files stored outside system must be XML	Non	Non	Yes	Luis 10/16

Older versions (end)

Design (Abdul)

Finite State Machine Diagram.

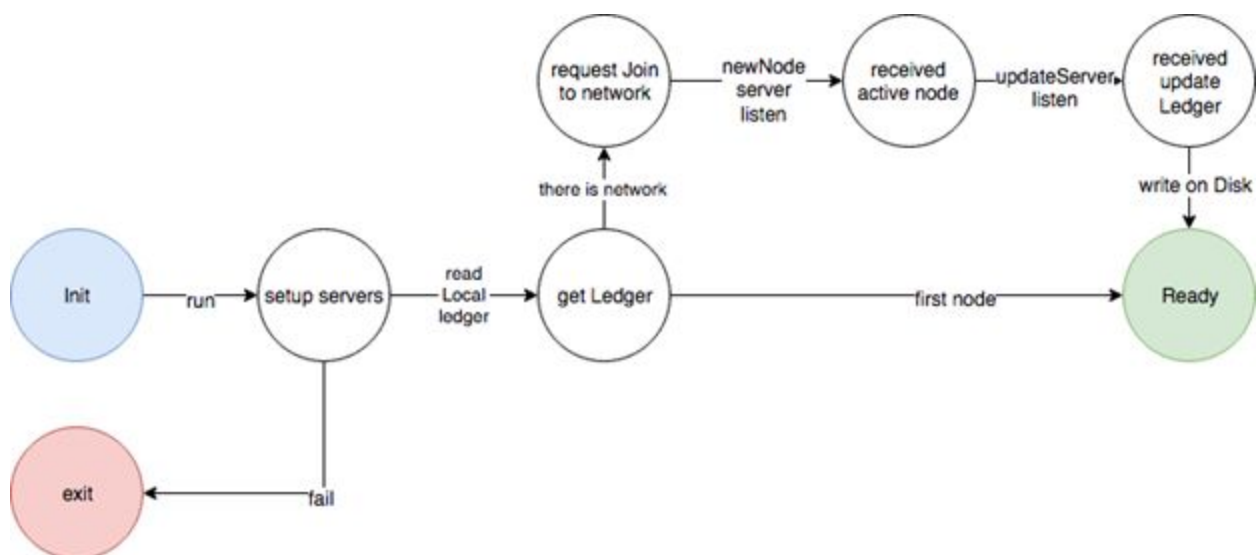
Since we are building peer to peer network, every node (peer) in the network must act as server and client. Also, a node must perform some actions to be valid node in the network.

Before the node join or establish a network, it must go through some step such as initializing server socket and setting up a ledger. Once the node succeeds these steps, it can interact with other nodes in the network.

To simplify the finite state machine (FSM) diagram which represent the different stat a node can be at, I break down the diagram into smaller diagram.

We will start with a FSM diagram for starting up node, this is what happen when we run the program.

In the following diagram, blue state represents the initial start, red state represents exiting/closing the program, and green state represent accepting state which means the node is running and able to interact with other nodes in the network.



When the program run, it will try to establish and run servers on certain ports that the node need later. Once the servers are running, the node will try to read an existing ledger; if there is no ledger on disk, it will initialize new one.

As I mentioned earlier, we are building peer to peer network, so the node will look for existing network to join. If there is network, or even one other running node, it will request joining the existed network, otherwise, the node will consider itself the only running node. If the node found another running node, it will try to join the network by sending its ID to whatever running node found. Once the node sent its ID, the other node will add the received ID to its active nodes list, and send back it ID and forward the received ID to other nodes to do the same. Meanwhile, the node that initially request joining the network, will used its New Node Server to listen to all node, so they can send their ID.

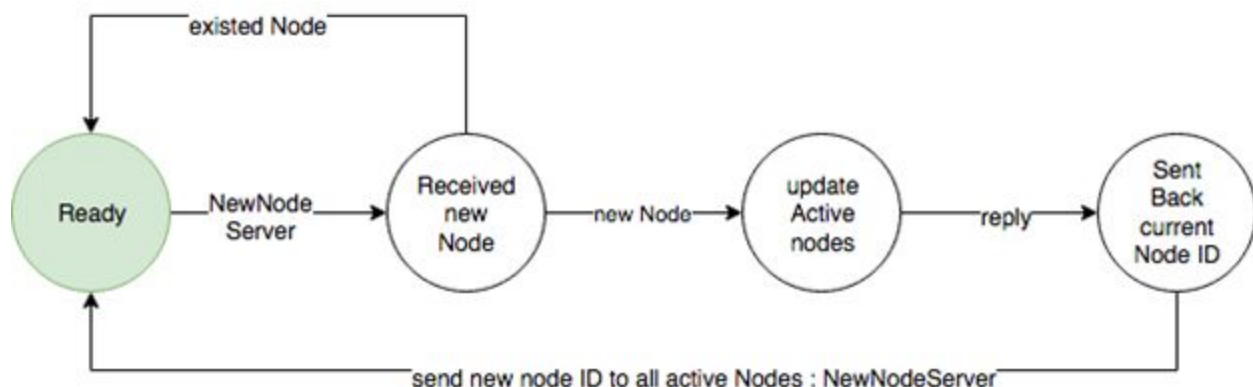
The new node will also receive copy of the ledger that other nodes have.

Now, the new node has list of all active nodes and updated ledger, and it reached the accepting state, which we call ready.

Once a node become active and on ready state, it waits for an interaction from the user or other node to change its state.

In the follow FSM diagram, I will show the cause and effect that change a node's state.

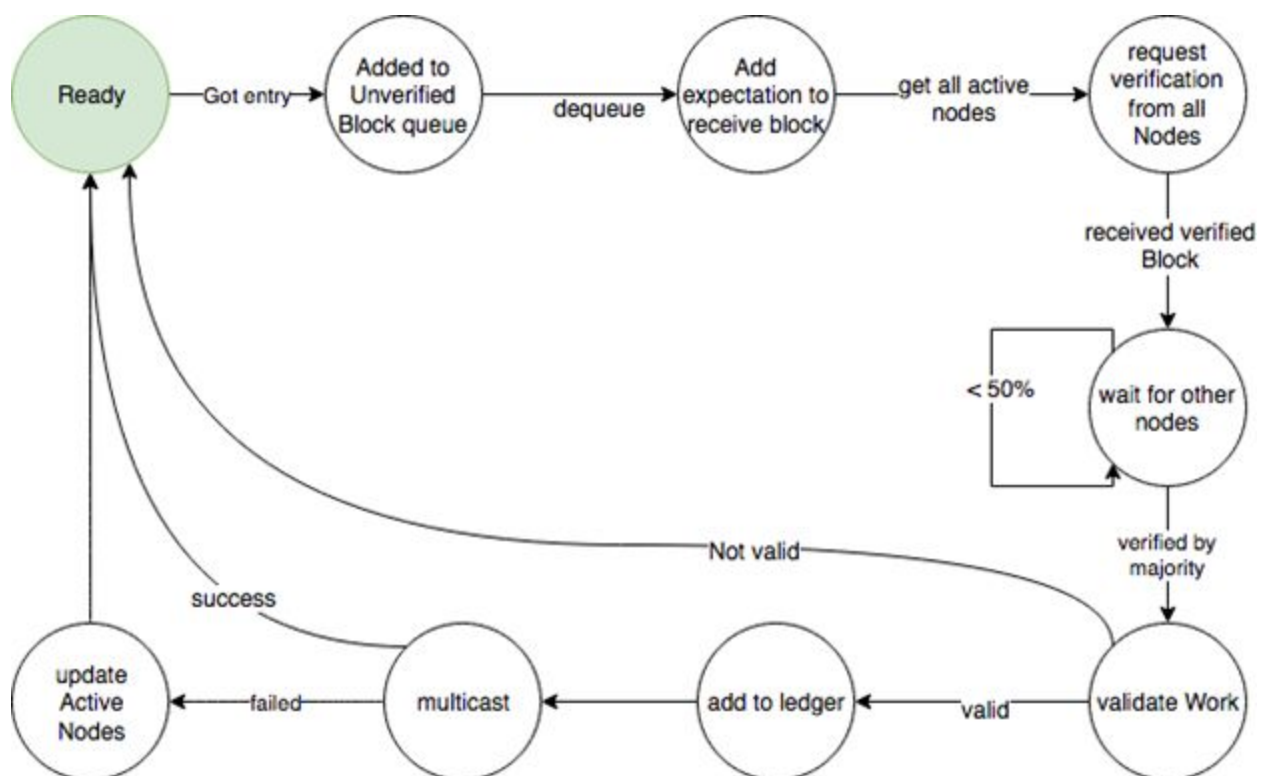
We will start with diagram that represent the different state for a node when it receives New Node request.



Every node has New Node Server that listen at specific port. New nodes will try to connect to this specific port to tell that it is trying to join the existed network. Once a node receives connection to the port, it will read the node ID that has been sent by another node.

The node will first check if it has the ID in its active nodes list. If it existed in the active node list, it will do nothing, otherwise, it will add to its active nodes list and send back its ID to the node which recently joined the active nodes list. And then, it will connect to other nodes New Node Server port and forward the Node ID that it just got. Other node will perform exactly as this node did to ensure all node received the new node ID.

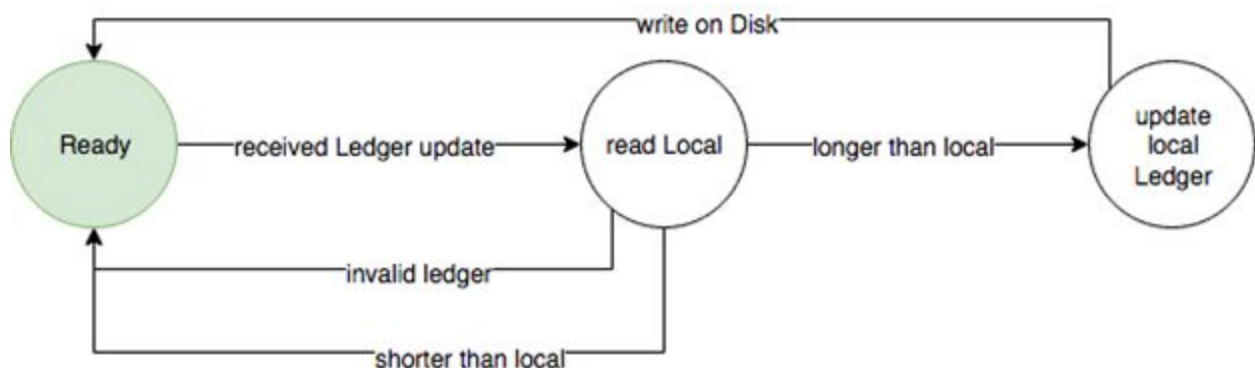
The user can interact with the program by web interface or command line interface. Either one should have similar functionalities. Once of the important functionality it to add new block record to the blockchain. In the follow FSM diagram, different state of a node is represented to show what happened when new block record has been entered.



When a user enter record, the program will create a block record object and add it to unverified blocks queue. Every node has running thread (Unverified Block Manager) that remove from the

queue and request a verification from all nodes (including current node). Before the Unverified Block Manager request a verification, it will tell the verified block receiver (BlockReceiver) to expect the block it is about to send for verification. Once the sent block ID has been added to the expected to receive, the Block Receiver will wait until it receives the block that just got sent. Since we are enforcing more security, the Block receiver need to wait until it get the block verified by half of the active node in the network. After the block got verified by half of the active nodes in the network, the block receiver will find the winner, the node that win the verification competition first, and it the block it received from the winner to the ledger. However, the ledger manager will check if the received block is actually got verified by running SHA256. If the received block is valid, the ledger manager will add it to the ledger and multicast the ledger. While it multicasting the ledger, any failure in reaching out to a node, means that node is no longer active, so it will be removed from the active nodes list.

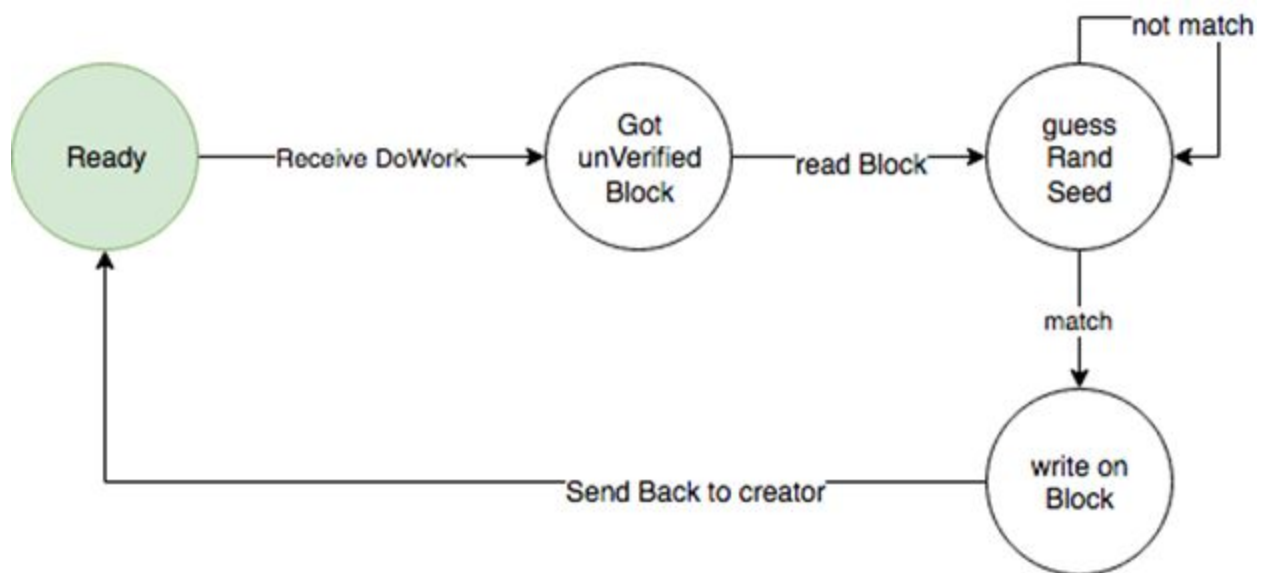
The previous FSM diagram showed the states that a node encountered for requesting a verification from nodes, receiving verified block, and eventually multicasting the updated ledger. Now all node will receive the update ledger, and this will cause the node to change it state because all node received update request at their UpdateServer that use different port number. In the following FSM diagram, state of updating ledger are shown for all node (include the node that multicast the update).



The Update server will be used for receiving ledger. This server will be used in two situations. First situation is when a node join a network. Other existing nodes will connect to the update

server of the new node to send the update ledger to ensure the new node is updated. Second situation is when one node multicast an update to its ledger such as the situation I showed in the previous FSM diagram. When a node receive ledger, it will first read the ledger it has locally, and compare it to the one it received, if it longer than the local ledger, it will update the local with the received one and then write it on disk. There is a situation where a ledger can be invalid, which happen when network newly established and no node has ledger.

A major functionality a node must be able to do is to do work and verified a block. In previous FSM diagram, I showed how the program handle entries from the user. The program will first create block record object, send it to all node for verification, and receive the block after it get verified. Thus now, we will see how all node handle the received unverified block. Every node has DoWork server that receive unverified blocks and try to figure out the random seed that should go into the block to meet the puzzle requirement.

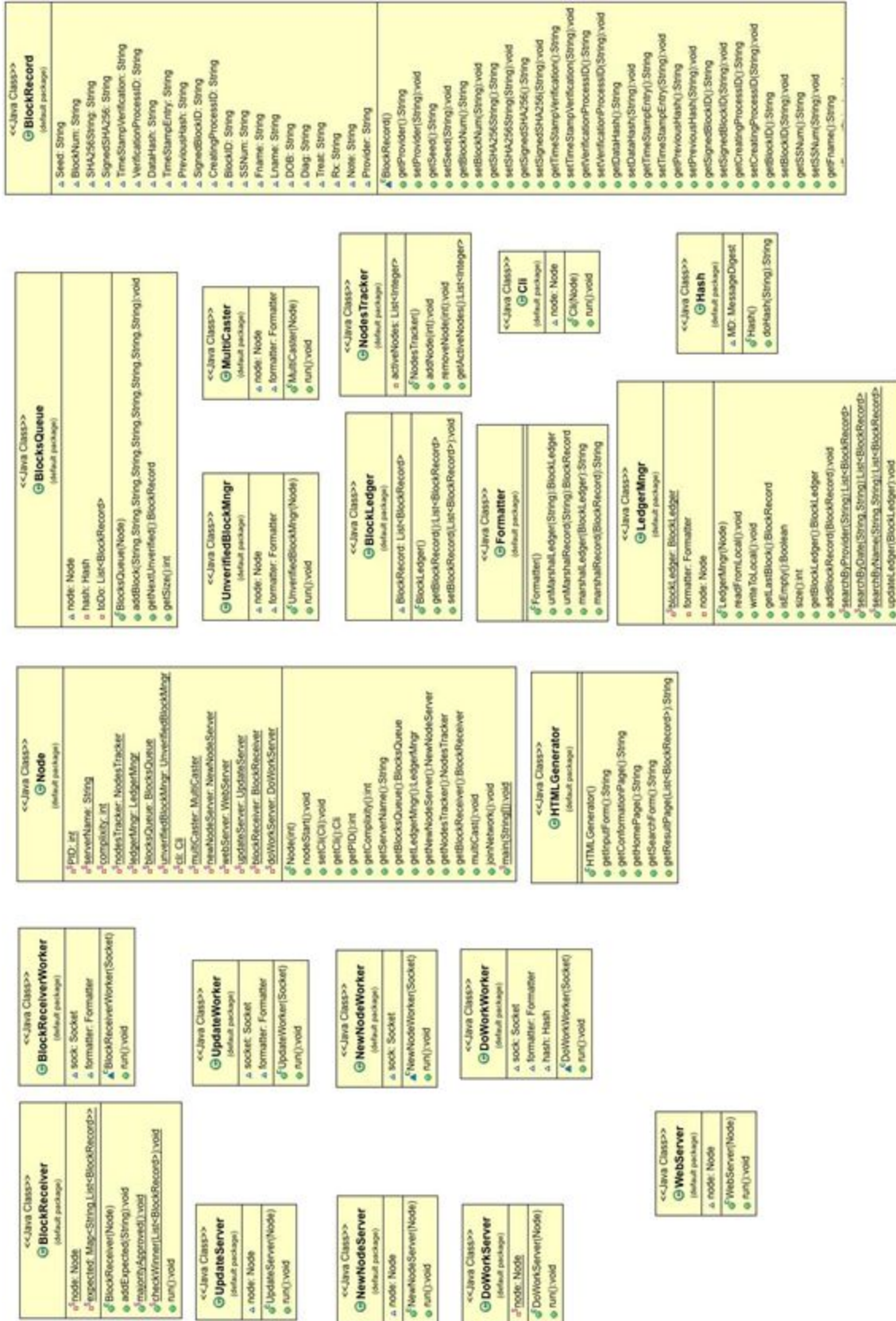


When the DoWork server received request, it will read the block and keep guessing the random seed until it get it right (or what meet the puzzle specification). Once the puzzle is solved the block will be feed with the right seed, and other data such as timestamp of when the puzzle is solved, node id, etc. Then the block will be send to creating node's Block receiver.

Technology

- Language: Java (or python), HTML
- Multi-threading
- Sockets
- XML: external data format
- IDEA: IntelliJ

Classes UML Diagram



Project Plan

Like professor Elliott said at the beginning of the course, planning is a complex task that cannot yet be accomplished by any kind of software or program. Planning can certainly be aided by software, but only in conjunction with a *planner*.

In general, our planning greatly revolved around project design and implementation. After we had finalized our project's requirements and design, we broke down our individual tasks around the pace that we expected implementation to have. For example, we expected the creation of blocks via a command line interface to be testable by mid-October, so we made sure to have sample data created and ready by then. When it came to tasks needing to be pushed back or rescheduled, we simply had to readjust its due date on our planning software. If other tasks were dependent on a specific task, as they often were, those would simply be pushed back and readjusted as well.

Trello

Our first go-to for planning was the task management service known as Trello. Having experienced Trello in past projects and jobs, we used it to plan our first week as a group and complete the first project demo. As can be seen in Figure 1, Trello allows tasks to be created and stored in one of three categories: To Do, Doing, and Done. When created, every task would initially be stored in "To Do," and it would be up to the individuals assigned to a certain task to drag and drop the tasks to their respective category throughout their duration. Although this was relatively easy to do, as tasks accumulated, the "Done" section would eventually become overpopulated.

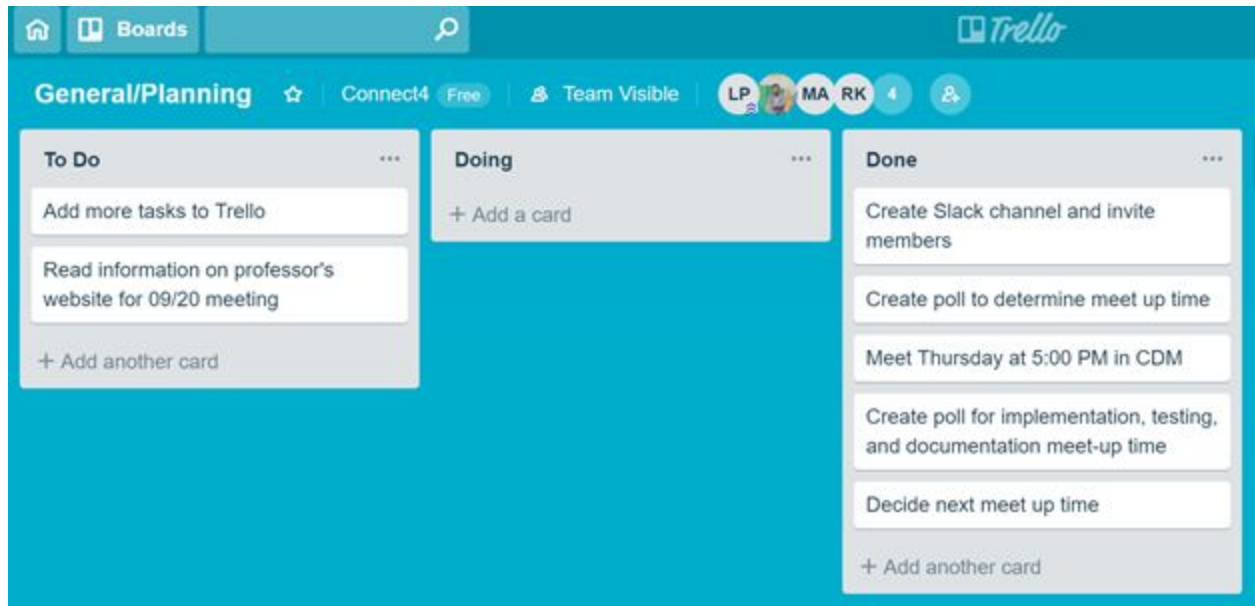


Figure 1

To help with task overpopulation, tasks would be further broken down into buckets that Trello refers to as “Boards. These boards would be named after the positions held by the groups: Documentation and Presentation, General and Planning, Implementation, and Requirements and Testing. Groupmates that fit into each respective board would then be assigned to it and would be responsible for their task upkeep.

Although it satisfied our initial needs, we would eventually move away from Trello because some of the features that the group required—such as task dependencies and calendar integration—were either absent or required a premium membership.

ClickUp

After researching and prototyping various project management software suites, we found one that met our needs: ClickUp. Unlike other services we found, ClickUp offered the same initial features as Trello plus task dependencies and calendar integration, all while being *free*.

As we approached our second project presentation, we ported all of our tasks from Trello into ClickUp—a task that was relatively painless, if a bit time consuming.

Navigation + Organization

Like Trello, tasks on ClickUp were organized in buckets, known as “Lists,” that corresponded to the individual roles of every team member: Implementation, Requirements, Testing, Design, and Presentation (See Figure 2). It should be noted that there are more Lists in our ClickUp plan than in Trello; this is because some Trello lists, like “Requirements and Testing” for example, were split in order to have clear task breakdown.

Upon creation, each List is color coded, a feature that would become extremely helpful in visualizing the workload for a specific day, week, or month. To further aid navigation of tasks, ClickUp allows one to see tasks through any member of the team. In the lower tab on Figure 2, the “Team” window, along with the names of everyone in the group, can be seen. Clicking on any of the names will show the user all task assigned to that particular member. This filter can be used in the various task windows available in ClickUp.

SPACES **Connect Four Blockchain P...**

Capstone Project	▼	+	...
Implementation	●	2	
Carry Water/General	●	3	
Requirements	●	1	
Testing	●	2	
Design	●	1	
Presentation	●	2	

TEAM +

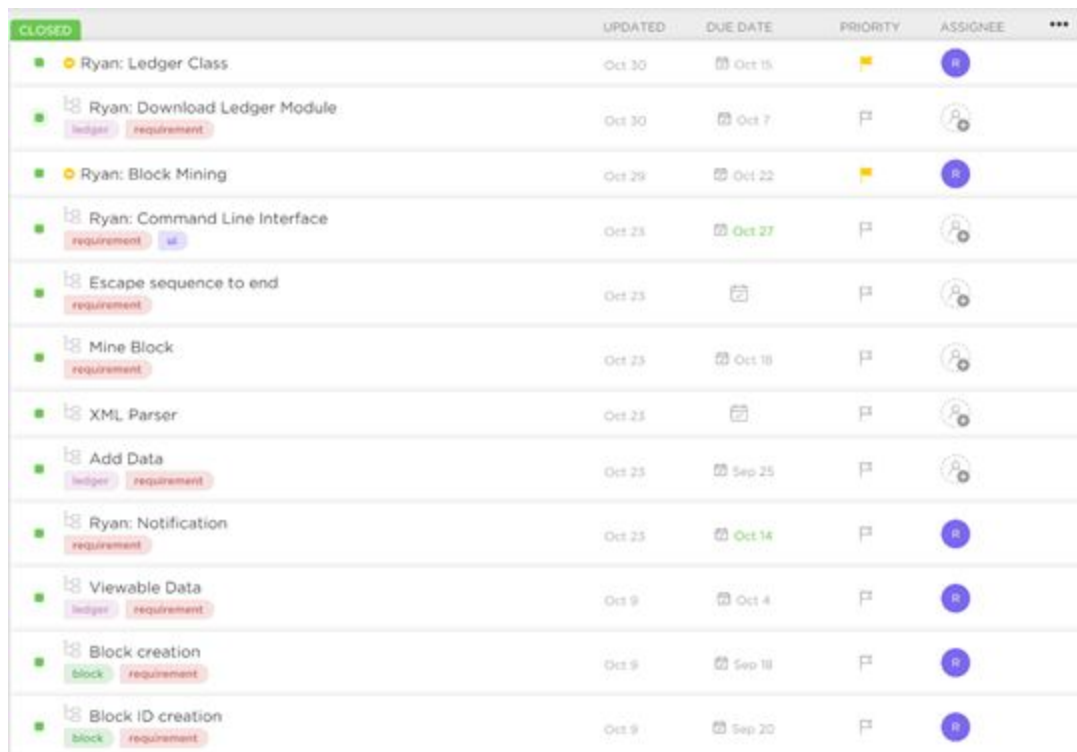
🔍 Filter...

👤	Unassigned	0
LP	Mine	3
AL	Aaron Leppin	4
AA	Abdulrahman Alanazi	5
JR	James Robinson	5
MA	Muhammad Alam	4
OO	oshioke oleghe	7
R	ryan.karpinski2@gmail.com	5

List View

Below, in Figure 3, is an example of the List View available in ClickUp. The List View shows all tasks assigned to a list. In this example, we are observing the Implementation list and the tasks that assigned to its members, mostly Ryan. From left to right is the task name, the date the task

was updated, its due date, the task's priority, and the assignee. Next to the task name are symbols that represent one of two things: whether the task is dependent on another, which appears as a yellow circle with a white dash, or if the task is subtask of a bigger task, which appears as a line with two nodes sticking out of it.



CLOSED		UPDATED	DUE DATE	PRIORITY	ASSIGNEE	...
●	● Ryan: Ledger Class	Oct 30	Oct 15	High	R	
■	■ Ryan: Download Ledger Module ledger requirement	Oct 30	Oct 7	Low	R	
●	● Ryan: Block Mining	Oct 29	Oct 22	High	R	
■	■ Ryan: Command Line Interface requirement	Oct 28	Oct 27	Low	R	
■	■ Escape sequence to end requirement	Oct 23		Low	R	
■	■ Mine Block requirement	Oct 23	Oct 18	Low	R	
■	■ XML Parser	Oct 23		Low	R	
■	■ Add Data ledger requirement	Oct 23	Sep 25	Low	R	
■	■ Ryan: Notification requirement	Oct 23	Oct 14	Low	R	
■	■ Viewable Data ledger requirement	Oct 9	Oct 4	Low	R	
■	■ Block creation block requirement	Oct 9	Sep 18	Low	R	
■	■ Block ID creation block requirement	Oct 9	Sep 20	Low	R	

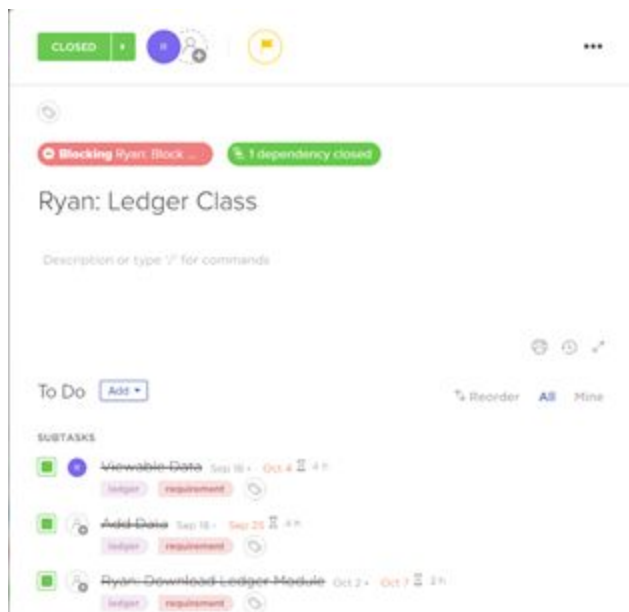
If one clicks on a task, they will be taken to the Task View.

Task View

The Task View, as seen in Figure 4 on the right, provides one with more information on a specific task. Here, one can see the current status of the task—Closed, Open, or In

Progress—any dependencies the task may have, subtasks, and a description or attachments, if available. Additionally, group members could manually add To-Do lists to a task if they needed to.

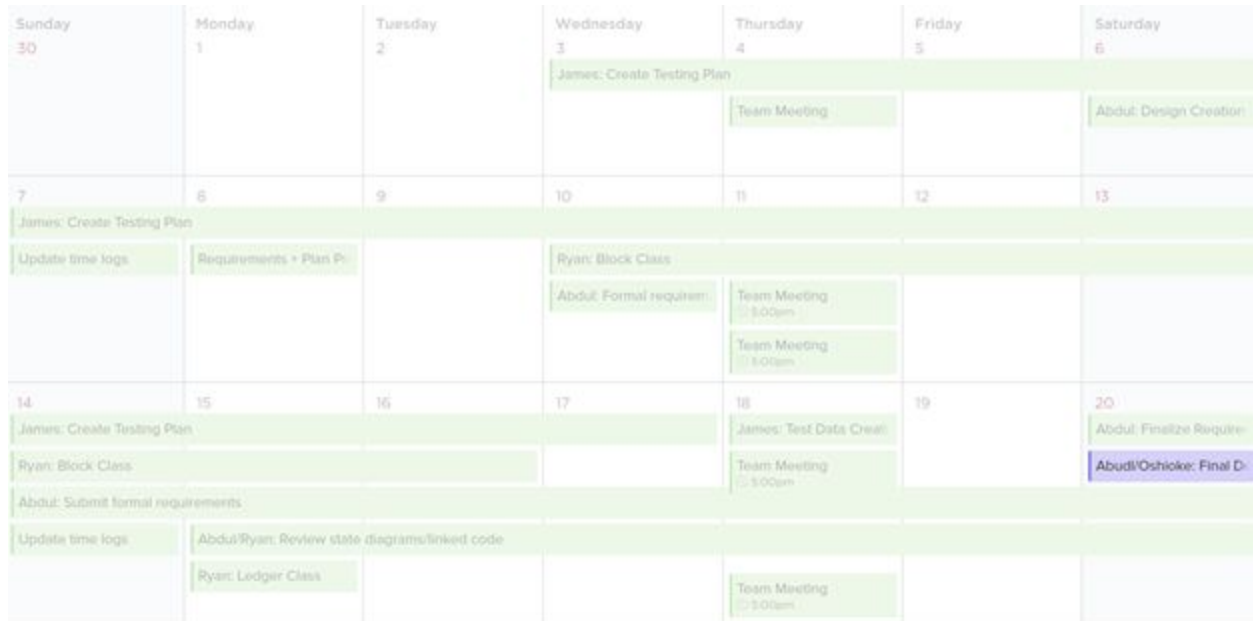
When creating tasks and subtasks, we made sure that the estimated time to complete never exceeded five hours. In the case that additional work had to be done, either because of a delay or because new unknown tasks needed to be made, we created separate tasks, so as to preserve these guidelines.



Calendar View

Finally, the Calendar View allowed us to view tasks as a collective. In this three-week window between September 30th and October 20th seen below in Figure 5, we can see the various tasks that needed to be accomplished. At the beginning of the month, we can see that James was tasked with creating Testing Plans for our implementation. At the same time, Ryan was working on completing the Block Class while Abdul was completing the design documents needed for our preview demo. It should be noted that the group met every Thursday to review tasks and overall project progress.

When viewing the Calendar View, tasks were displayed in one of three colors: green for completed tasks, purple for tasks in progress, and grey for open tasks. In the Figure below, all tasks but one appear green, since they had already been completed.

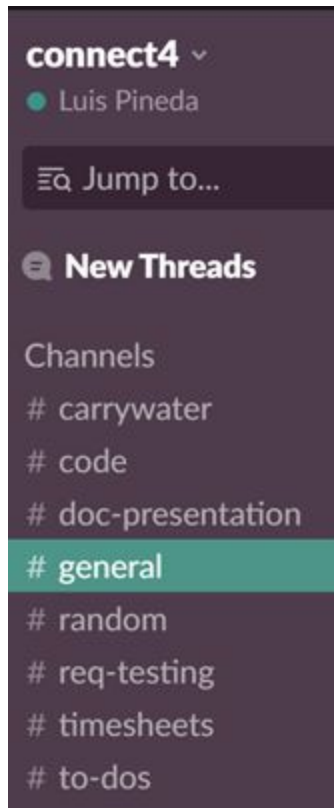


Collaboration Software

Slack

In order to establish open and persistent lines of communication within the group, we created a Slack channel. Much like other messaging services, Slack allowed us to create various channels for content-specific discussions. Like our plan, these channels corresponded to the individual responsibilities assigned to us at the beginning of the course and can be seen in Figure 1 below. Overall, this platform worked well—especially because Muhammed was unable to communicate through our group-text at the beginning of the quarter. If any group member ever had any questions about blockchain or the project, they would ask the group in the General channel, where it would be seen by everyone. This was especially helpful at the start of the course, when everyone was researching blockchain and sharing helpful resources that they had found. Often,

such questions would evolve into discussions that were beneficial to everyone. Additionally, we were able to integrate external services like Github and Google drive into our Slack group, which made it a lot easier to access group documents since they were all accessible through a central source.



Testing Plan

Testing Overview

Unit testing is an extremely critical part of the software life cycle. During test driven development, creating the test cases forces the team to think about the design from different angles. It helps to keep the team focused and create a more robust design. It also ensures that you define what that unit of code is responsible for. Identifying defects as early as possible is crucial to reducing the cost of bug fixes. The further down the development path you are the more difficult it is to detect bugs due to the layers of code changes that have been applied over the weeks.

Our team created a suite of tests to test our requirements list. The number of the test in the Testing Matrix correlates to the requirement number in the Requirements Matrix. If a test did not pass it was noted in the Pass? column and a new row was added with an incremented revision number. When a test passed the test was signed off on by the tester.

Test Plan

Test ing Matr ix								
	Number	Revision	Requireme nts Test	Results	Pass?	Date of test	Comment s	Client Sign-off name
	1	1	Verify code is using SHA256		Pass	10/30/201 8		James Robinson
	2.1	1	Verify that there are no duplicate block ids in the chain by listing all block ids	...	Pass	10/30/201 8	...	James Robinson
2.2	1	Verify that the block id is created before the block is.		Pass	10/30/201 8		James Robinson	

2.3	1	Delete verified block from ledger	Unable to delete block	Pass	11/4/2018		James Robinson
2.4	1	New block is added to the ledger when a correction is needed.					Not submitted
2.5	1	Correction block points to the block it is correcting					Not submitted
2.6	1	Block actions must be time stamped when block is created and block is cracked	Block does not contain time stamp	Fail	10/30/2018		Not submitted

2.6	2	Block actions must be time stamped when block is created and block is cracked		Pass	11/4/2018		James Robinson
3.1	1	Bring down one node and update the ledger. Bring node back up and verify that ledgers have synchronized.	The nodes that remained up were not synchronized.	Fail	10/4/2018		Not submitted
3.1	2	Bring down one node and update the ledger. Bring node back up and verify that ledgers have synchronized.			10/?/2018		Not submitted

3.2	1	Create 3 nodes and verify that the ledger is available and sync'd on all of them.	Three virtual nodes are created and the ledger is in sync on all	Pass	10/30/2018		James Robinson
4.1	1	Input patient data via web interface and validate that all fields are encrypted in block	Patient data can be entered into the ledger via web interface	Pass	11/4/2018		James Robinson
4.2	1	Input patient data via console interface and validate that all fields are encrypted in block.	All patient information can be entered via CLI	Pass	10/30/2018		James Robinson
4.3	1	Console interface - add block command					Not submitted

4.4	1	Console interface - display ledger command	Data can be displayed from CLI	Pass	10/30/2018		James Robinson
4.5	1	Query the ledger by event date and return all records for that date		Fail	10/30/2018		James Robinson
4.6	1	Query the ledger and return records for a specific patient, by date of event.	Patient data was not returned	Fail	10/30/2018		Not submitted
4.6	2	Query the ledger and return records for a specific patient, by date of event.		Pass	11/4/2018		James Robinson
4.7	1	Query the ledger and return records where provider = Cigna and		Pass	11/4/2018		James Robinson

		event date = 10/05/2018					
4.8	1	Query the ledger and return records where event date is between 09/30/2018 and 11/05/2018		Fail	10/30/2018		James Robinson
4.9	1	Query the ledger and list all prescriptions where patient ssn = 574-98-4564 and provider = Cigna and date = 10/07/2018		Pass	11/4/2018		James Robinson
5	1	The output must be in XML format	Output can be exported to XML	Pass	10/30/2018		James Robinson

Project Management

Overview

Project Management was a critical component of completely this project successfully. It allowed the team to define the requirements, design the solution, and build the software by organizing the team's communication, meetings, and work towards a common goal in a timely manner. A project is a temporary, unique endeavor that has a defined scope, beginning, and end meant to reach a single, definable goal.

Methodology

Because of the short nature of this project, the team choose to apply a hybrid project management methodology. Both traditional project management and a more agile project management approach were utilized.

Traditional Project Management

Traditional project management was used to define the overall phases of the project and to define the static timeline of this project. Each of these phases was then used to plan the tasks and due dates and are described in the project planning section.

Initiation Phase

This phase of the project began with the assignment of the team. In this phase team roles and responsibilities were defined and selected. Also, the team defined basic organizational activities, including how and when the team will meet and communicate for project meetings. We discussed the goals for the project and defined the scope that Connect Four wanted to deliver. The team also spoke about constraints around schedules, availability, and abilities.

Requirements Phase

This phase of the project was used to define the requirements for the class project. What areas were Connect Four going to focus on and which areas were we going to spend less time on because of the constraints defined during the initiation phase. One area we decided to not focus on was human computer interaction because the team had no resource from that degree program. Also decided was the testing needed to ensure the requirements were met.

Design Phase

In this phase, the requirements and testing were taken and turned into a solution. Decisions about how the nodes were structure and how data was stored and transmitted were made and documented. The team also decided on technology stacks needed to meet the requirements. Early on, Python was the development language of choice, but by the end of the course, that switched to JAVA. The desired architecture and basic data models were created to provide the implementation team detail on what to develop.

Implementation Phase

This phase was primarily focused on taking the design and developing software. Aspects of the design that did not work as expected were re-evaluated and new choices were made to meet the requirements as defined. Testing was completed as modules finished to ensure that design and changes produced the desired results.

Closing Phase

In the closing phase, documentation and the final presentation were completed. All outstanding tasks were either closed or removed if deemed unnecessary. This phase was also used to reflect on the project and determine what worked well and how the project could have ran better.

Agile Project Management

Since the class term is relatively short with quite a bit of work to complete, Connect Four also implemented several Agile project management methods. As many of the phases from traditional project management overlapped, we did not have a complete picture to plan, schedule, and organize the work. Taking the outlines and high-level detail provided by the phases from the traditional project management, each was filled in with detail as the team progressed through the project.

Weekly Team Project Meetings

The team had project meeting twice a week, on Tuesday evening after class and on Thursday at 5 PM. Tuesday, we would take the new information provided in class to refine any plans and requirements for the project that were needed. We would define, assign, and schedule tasks, updating our planning tool. We would also review work turned in during class such as presentations, taking the feedback provided to refine the project.

Thursday project meetings were a review of work completed that week, allowing team discussion and decisions to be made about the project. Again, the team would redefine tasks and modify the schedule accordingly.

Time Logs

Time logs were kept ensuring proper tracking of time and to match up with tasks required. As tasks completed, the project manager and planner could make decisions on any needed changes to schedule, scope, and delivery of the project.

Abdulrahman Alanazi			
Date	Time	Duration	Description
9/11/2018	8:15 PM	0.75h	met in group. Chose responsibilities. Concept demo
9/14/2018	1:00:00 PM	2.75h	watch more video about blockchain
9/14/2018	5:00:00 PM	2.25h	write the requirements
9/15/2018	2:00:00 PM	5h	design solutions. Started with Java Servlets and jsp.
9/16/2018	4:00:00 PM	3h	Change the design from MVC and using java servlets to p2p using python
9/16/2018	7:00:00 PM	2h	Meet with Ryan and talked about the design and we will implement the application
9/18/2018	8:00:00 AM	1h	meet with group during the class
9/28/2018	5:00:00 PM	2h	meet with group and explain the overview design
9/30/2018	5:00:00 PM	3h	design for block, transaction
10/1/2018	3:30:00 PM	2h	discover AWS
10/2/2018	8:00:00 PM	1h	meet with the team and go over the initail design for one node
10/4/2018	4:00:00 PM	2.5h	meet with Aaron and Jim, and show how the project get break down into smaller pieces. When my phone got stolen :(
10/5/2018	6:00:00 PM	1.5h	update the requirement v2
10/6/2018	4:00:00 PM	2h	complete the bad client/dev column of the requirments and make v3
10/8/2018	8:00:00 PM	3h	create electronic version of the design
10/9/2018	11:00:00 AM	3h	make dynamic web page for the design and including state diagram and sequence diagram.

Ryan Karpinski			
Date	Time	Duration	Description
9/11/2018	4:00 PM	45 minutes	Met with group for the first time. We discussed which roles each of us will be taking
9/12/2018	3:00 PM	3.5 hours	Started writing code for the block class. Researched libraries and other tools that would be best for the block class
9/13/2018	2:00 PM	1 hour	Met with group to discuss ideas and thoughts about the upcoming concept demo. Displayed some code to show that the block class is currently working
9/14/2018	3:00 PM	1.5 hours	Met with Abdul to discuss requirements and talked about some optional ideas and different directions we can go with this program
9/15/2018	6:00 PM	2.75 hours	Researched much deeper into the networking aspect for p2p and cleaned up and changed some code for the block class in the testing branch
9/16/2018	7:00 PM	2 hours	Met with Abdul about he proposed design and to cover requirements once again. We discussed blockchain technology and started working together on piecing together a new design.
9/25/2018	2:00 PM	1.5 hours	Met with group in class and continued talking about design and other aspects of the project mostly pertaining to the upcoming requirements/design/planning presentation.
			Cleaned up a lot of the code and added much more extensive comments for readability. continued to write more useful functions

Jim Robinson			
Date	Time	Duration	Description
9/11/2018	8:00 PM	45 min	Met with group, chose responsibilities
9/13/2018	5:00:00 PM	1 hour 30 min	Met with team to discuss 1st demo
9/14/2018	6:00:00 PM	1 hour	Watched video on BlockChain
9/15/2018	3:00:00 PM	1 hour	Research blockchain test cases
9/16/2018	12:00:00 PM	3 hours	Met with team to discuss demo, Created test case document
9/18/2018	8:00:00 PM	30 min	Met with group discussed requirments for the RDP demo . discussed current roles and responsibilites
9/20/2018	5:00:00 PM	1 hour 30 min	Met with team to discuss demo #2
9/23/2018	4:00:00 PM	30 min	researched smart contracts
9/25/2018	8:00:00 PM	1 hour	Met with group discussed requirments for the RDP demo, discussed who the user base is
9/27/2018	5:00:00 PM	1 hour 30 min	Met with team went over requirments matrix, carry water tasks, timesheet updates
9/30/2018	4:00:00 PM	1 hour	created test case matrix
10/02/2018	7:00:00 PM	1 hour	Met with group discussed requirments for the RDP demo . discussed the design of the code
10/04/2018	5:00 PM	1 hour	Met with team to discuss demo #2 and design uml
10/06/2018	2:00 PM	30 min	researched pros and cons of different language implementations
10/07/2018	3:00 PM	1 hour	Updated testing matrix
10/09/2018	4:00 PM	1 hour	Organized test cases
10/9/2018	7:00 PM	1 hour	Discussed technology to be used in block chain code

Oshioke Oleghe			
Date	Time	Duration	Description
9/11/2018	8:00 PM	45mins	Had group meeting to discuss individual roles.
9/11/2018	11:30 PM	30mins	Joined the group slack channel and reviewed a few documents shared in the channel.
9/12/2018	11:00 AM	2hours	Watched a few videos about blockchain to improve my knowledge on it.
9/13/2018	5:00 PM	45mins	Had a group meeting with my other group member to see what we need to do to get the demo done for tuesday's class.
9/14/2018	11:00 PM	10mins	Filled in time availability for further group meetings with the doodle poll via slack.
9/15/2018	8:00 PM	1hour	Checked documents shared by other group mates, like codes via github.
9/19/2018	11:00 AM	30mins	Read more on planning on Prof Elliots website.
9/20/2018	2:00 PM	50mins	Watched a youtube video on how bitcoin (and other cryptocurrencies) actually work which was posted by Abdul via Slack.
9/23/2018	7:00 AM	1hour 30mins	Checked the new documents shared in slack and reviewed the requirements for the RequirementsDesignPlanDemo.
9/25/2018	5:00 PM	45mins	Met with group members before class to discuss plans on our we will deliver our requirements design plan demo
9/27/2018	5:00 PM	45mins	Group meeting

Luis Pineda

Date	Time	Duration	Description
Sep/11/2018	4:00 PM	45 Mints	Meet with group, chose responsibilities
Sep/11/2018	5:00 PM	2 hours	Created Slack Channel, Doodle Poll, reached out to members via text, and watched video on BlockChain
09/12/2018	2:00 PM	20 mintues	Shared video with team members
09/13/2018	3:00 PM	1.5 hours	Met with team to discuss demo
09/14/2018	5:00 PM	30 mintues	Created doodle poll for second meeting time
09/15/2018	4:00 PM	30 mintues	Reviewed doodle poll results and scheduled meeting time for team members
09/16/2018	7:00 PM	1 hour	Spoke with Muhammad and other team members through Slack and text to coordinate meeting location.
09/15/2018	4:00 PM	1.5 hours	Created Trello for group planning and management. Assigned tasks and linked with Slack channels.
09/18/2018	8:00 PM	2.5 hours	Met with group before class to prepare for presentation. Later met with group after class to discuss roles.
09/19/2018	11:00 PM	2 hours	Researched more about Blockchain to prepare to ask Ryan questions. Also went over project requirements from website.

Muhammad Alam			
Date	Time	Duration	Description
Sep/16/18	12:00 PM	2 hours	Spoke with Aaron, & James Robinson about the ppt
Sep/16/18	8:00 PM	45 mintues	Made an outline of the presentation
Sep/17/2018	20:00	4 hours	Made Presentation
Sep/18/18`	11:00 PM	25 Mints	Reading the professor website about the project requirements
Sep/18/18	8:00 PM	1 hr	Watch a youtube video and so researching that how blockchain software works and have a clear understanding
Sep/23/18	1:00 AM	45 Mintues	Reviewed the new documents and check the requirements and design
Sep/25/18	5:00 PM	3.5 hours	Missed the class on Sep/25/18 due to illness so checked d2i throughly and watch the lecture
Sep/27/18	4:45 PM	30 Mintues	Meet with Ryan before group meeting and talk about the presentation
Sep/27/18	5:00 PM	45 Mintues	Meet with the whole group explains the pattern of our presentation and discusses about the improvements of the presentation
Sep/27/18	9:00 PM	30 mintues	look over the slack and read the project description
			Meet with group discussed the requirements, design

Aaron Leppin			
Date	Time	Duration	Description
9/11/2018	8:00:00 PM	0:45:00	Had group meeting to discuss individual roles, requirements.
9/12/2018	11:00:00 AM	0:15:00	Communications with team for meeting dates/time. Took Doodle poll.
9/12/2018	8:00:00 AM	0:35:00	Joined Slack group, Reviewed course materials
9/13/2018	4:45:00 PM	1:15:00	Group meeting, finalized requirements and design. Ryan reviewed code and concept. Finalized tasks for rest of week.
9/13/2018	8:00:00 PM	1:25:00	Watched Youtube on blockchain, review grad material provided by professor
9/13/2018	7:00:00 PM	0:45:00	Creation of basic project plan for 1st demo
9/16/2018	12:05:00 PM	1:20:00	Group meeting to review presentation for 1st demo and gather materials from team members
9/16/2018	6:00 PM	0:05:00	Entered time for week
9/17/2018	5:00:00 PM	0:20:00	Communications with team about final presentation tasks.
9/18/2018	8:00:00 PM	0:50:00	Met with group after class to discuss requirements, design, and planning presentation
9/19/2018	1:55:00 PM	0:05:00	Posted meeting agenda to Slack
9/20/2018	5:00:00 PM	1:20:00	Group meeting to review design of project
9/22/2018	1:00:00 PM	1:00:00	Evaluated several PM tools to track and present planning tasks. ClickUp decided
9/23/2018	7:00:00 PM	0:10:00	Entered time for week
9/25/2018	8:00:00 PM	1:00:00	Group meeting after class to review requirements, divide up remaining tasks, and review planning tool.
9/25/2018	9:20:00 PM	0:10:00	Wrote to slack channel review of meeting, tasks
9/26/2018	6:00:00 PM	0:20:00	Review RDP website material and reviewed outside requirements documents
9/30/2018	7:45:00 PM	0:15:00	Entered time for week

Images of each team member's time log and a detailed description of what they did individually towards the success of the project.

Business Plan

Mission

The mission of Connect Four is to offer a medical records platform to insurers, healthcare providers, and patients that is convenient, compatible, connected, and confidential.

Company summary

Connect Four will offer a distributed medical records application built on blockchain technology. By utilizing blockchain technology, the application will offer a built-in security, while allowing records to be accessible to healthcare providers, insurers, and patients. Founded by the team outlined in the project team section, Connect Four will focus on delivering a modern, cost-effective medical records platform.

Opportunity

The Electronic Medical Record Industry in 2017 generated revenue of \$22.3 billion dollars and is expected to continue to grow by 5.4% annually between 2018 and 2023. This increase is driven by the need for better information systems, better healthcare, and government initiatives (Electronic Health Record (EHR) Market). Currently, no system provides complete interoperability of medical records across public and private healthcare sectors. This compounded with increasingly fragmented technology solutions, complex data sharing, and larger silos of information, presents an opportunity for a new medical records system (Catherine Sturman).

Offering a medical records platform that allows patients to view and share their medical records with the providers of their choice give flexibility to the patient on where and how to get treatment. This flexibility increases competition, forcing the providers to provide better, affordable, healthcare solutions. With this increased competition and lower costs, insurers will be forced to standardize coverage and reimbursement to the provider to level the playing field. Patient choice will be more driven by the care given and the cost than the insurer's prescribed network of doctors and facilities.

By utilizing blockchain technology in the platform solution, security and confidentiality will be built in from the beginning, not added secondarily. By encrypting every block and the ledger, records are protected at rest. By distributing the ledger among many healthcare providers, modifying or falsifying medical records become extremely difficult.

Offering

Connect Four will build a distributed application with a software as a service (SAAS) model built on blockchain technology. Each participating healthcare provider or insurer will operate compute nodes that will hold and add to the blockchain ledger. Each node contains a command line interface (CLI) and a web interface. The application will provide functionality based on the following four pillars:

Connect Four will host a central website for patients to access their records. The patient website will allow a patient to allow records to be shared to specific healthcare providers and insurers. Patients will also be able to view their medical history, test results, prescriptions, and doctor and hospital visits all in a centralized location. Also, Connect Four will operate a minimal infrastructure to manage billing and logging of blockchain work completed by each node. The logging data collected of work completed by each node will be used to calculate discounts to the service offering. The Connect Four medical records platform will be offered in a subscription model, billed monthly and contracted for one year, three years, or five years. Work completed by the nodes hosted by a provider will be captured and used to calculate discounts for the next contract renewal. The total discounts provided to the network would be $X * 1/N$, where N is the number of total nodes in the network. X is a scaling factor to make the discounts meaningful.

Example Discounts and Pricing

In a two-node system (one node per company for two companies), where the discount was $1/N$, a total of 50% discount would be given ($1/2$). That means if a single node does 100% of the work in a given period, that company would pay \$0 for its term. If each node did 50% of the work, each would receive a 50% discount on the service.

As it scales, that discount becomes less meaningful, so a scaling factor, X , is added. If X is less than N , Connect Four will derive revenue. The amount of revenue Connect Four collects is directly related to the value of X and needs to be in balance with the discount provided to providers. Several examples below:

Description	Variable	Value
1000 participating providers in the network, each with a single node	N	1000
Monthly Subscription cost average per provider	M	\$5000
Discount scaling factor	X	See Below

Monthly Revenue for various scaling values			
Revenue for 1000 nodes * \$5,000 fee = \$5,000,000 per month			
Scaling Factor	10	100	200
Discount Formula	$10 * 1 / 1000$	$100 * 1 / 1000$	$200 * 1 / 1000$
Total Discount Value	\$50,000	\$500,000	\$1,000,000
Total Revenue	\$4,950,000	\$4,500,000	\$4,000,000
Average discount per node per month	\$50	\$500	\$1000

As seen in these examples, the amount of discount provided to providers can be scaled to balance revenue with discount. Connect Four will identify all operating costs a desired margin for profit to calculate the appropriate scaling factor for discounts.

References

Electronic Health Record (EHR) Market by Delivery Mode (Web/Cloud-Based Server, On-Premise), by Component (Practice Management, Patient Management, E-Prescription, Referral Management, Population Health Management), by End User (Inpatient Facilities [Acute Care, Long Term Care, Post-Acute Care], Ambulatory Care Centers, Physician Offices), by Geography (U.S., Canada, U.K., Germany, France, Italy, Spain, Japan, China, India, Australia, Brazil, Mexico, Colombia, Argentina, Saudi Arabia, U.A.E., South Africa) – Global Market Size, Share, Development, Growth and Demand Forecast, 2013–2023. (n.d.). Retrieved November 10, 2018, from <https://www.psmarketresearch.com/market-analysis/electronic-medical-records-market>. Catherine Sturman. (2018, August 15). The electronic medical records market is estimated to rise to \$39.7bn by 2022, report finds. Retrieved November 10, 2018, from <https://www.healthcareglobal.com/technology/electronic-medical-records-market-estimated-rise-397bn-2022-report-finds>.