HIGH SCHOOL EDITION
Journal of Student Research
Volume 11 Issue 3 (2022)

Building Better High School Clubs with Blockchains

Nevin Thomas[1] and Sam[#]

[1]American High School, USA
[#]Advisor

ABSTRACT

School clubs continue to be a popular extracurricular activity, especially in high schools. The number of new clubs and their membership has exploded in recent years. Many students participate in clubs over their four years of high school to explore interests, develop skills, engage with peers, and serve their community. The wide spectrum of member talent in a club, however, presents a challenge for the club leadership. In keeping with their mission statement, club leadership must continually train their teams to drive progress season after season. Leadership must collect, analyze and share vast amounts of member data. In this paper, we will examine a new data-sharing model a school club can use to manage its most important asset, its members. Our inspiration comes from commercial enterprises that deploy blockchain networks to manage assets through their supply chains. We will first explore the importance of member management for a club's success, before delving into the structure of a blockchain network and how its properties are well-suited for a club's data management. We will then create our own blockchain network, using open-source software from the Linux Foundation, before finally creating a prototype model that can be used by a school club.

Introduction

Running a high school club is a time-consuming process. There are events to organize, teams to train, and members to manage. As club membership and participation increase, more burden is placed on club leadership. Unfortunately, the majority of high school clubs lack the staff and resources to manage their members season after season and do not rise to their full potential. The impact of this inefficiency is particularly visible in competitive clubs, such as sports, math, and speech and debate. In tournaments, teams from resource-constrained schools are often at a disadvantage against well-funded teams from schools with dedicated staff and coaches. Technology has the potential to automate many member management tasks leading to better trained and performing teams.

Whether the club is competitive or service-oriented, tracking progress requires training and monitoring to guarantee the advancement of each team member and the team as a whole. The iterative process is similar to that followed by any successful organization. Figure 1 shows the continual improvement process in an efficient organization.

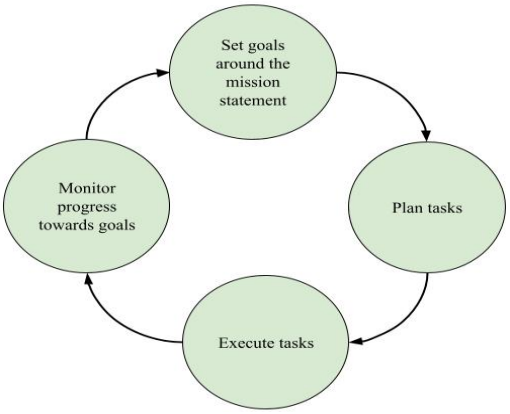

Figure 1. Continual improvement process

The continual improvement process in Figure 1 requires club leadership to collaborate on the progress of each member during the season. Officers need to collect member information, such as skill level, pending tasks, number of tournaments attended, competition results, etcetera. This produces a large volume of data that must be shared with the entire club's leadership team. Moreover, procedures that have been agreed upon by the leadership team must be in place on how to analyze this data. These policies should be smoothly handed over from one leadership team to the next at the end of the season.

The continual improvement process requires the following fundamental guarantees on the data.

- There must be transparency in the data among officers, coaches, and members.
- There must be permanent logging of all member transactions.
- There must be agreement among leadership on all club policies pertaining to its members.
- There must be secure storage and communication of all member data.

Most school clubs rely on cloud-based spreadsheets and shared folders to store club information. The spreadsheets offer ease of access, and are a "single source of truth." They operate in a client-server model that uses traditional databases. The model works well when access to the spreadsheets is limited to a few members, and the amount of data maintained is minimal, such as the names of members, their grade level, and when they paid their dues. There are minimal transactions or member state changes because the spreadsheets are updated infrequently, usually only at the start of a season when new members join.

However, as a club grows and a greater spectrum of member skill levels appears, the amount of data collected increases exponentially. The traditional database model, which consolidates data with a few members, fails to meet the needs of an expanding club organization. When spreadsheets are owned and controlled by a few members, there is reduced transparency and limited collaboration. Moreover, the data is susceptible to inadvertent or intentional tampering. If documents are copied by multiple coaches, it leaves them vulnerable to duplication, discrepancy, and corruption. Over time, a lack of trust develops in the data and its usefulness to the team leading to inefficient club operations. The client-server model fails to provide the fundamental guarantees outlined above**.**

What is needed is a platform where the data is transparent and trusted among all members of the team. This is where a blockchain network can help. Blockchain-based management can improve a student-run high school club's efficiency compared to traditional client-server management.

Blockchain network

Blockchains have become popular in recent years with the emergence of several cryptocurrencies, such as Bitcoin and Ethereum. Parties, technically known as peers, use blockchains to trade, transfer, and trace any valuable asset. It is important to distinguish between blockchains and cryptocurrency. Cryptocurrencies are

applications built on top of blockchains. The underlying blockchain is an infrastructure that facilitates crypto-currency transactions. The blockchain infrastructure has applications beyond cryptocurrencies in the emerging Web 3.0. Governments and privately-owned companies have already started investing in blockchain solutions (Khorram, 2022; *IBM Blockchain - Enterprise Blockchain Solutions & Services*, n.d.).

Databases have traditionally been used to store data. Blockchains are specialized databases, but they offer more benefits. Unlike a centralized database, a blockchain system features distributed databases or ledgers connected by a peer-to-peer network.

Our club management system relies on the key blockchain properties shown in Table 1.

Blockchain Property and Respective Data Requirement

Table 1. Blockchain property for club's data requirement

Blockchain property	Club's data requirement
Decentralization	Transparency of data among coaches and members
Immutability	Permanent logging of all transactions
Consensus	Agreement among staff on policies Consistent and automated policies
Security	Private and Secure

We will analyze each property of a blockchain and how it benefits the running of a school club.

Decentralization

A blockchain system is based on a peer-to-peer model. In this model, data is not owned by a single entity but is replicated across all participating peers. Any update to a peer's data is broadcasted to all peers in the network. Distributed ledgers make it practically impossible to alter data. There is no central authority in a peer-to-peer network, and this makes the blockchain data resilient against unintentional tampering. In the event of an inadvertent change to the data on a node, the blockchain immediately detects the discrepancy with the other nodes and corrects it. The lack of a single point of failure leads to a transparent and coherent network.

Decentralization develops a greater degree of trust among club leadership and members. Since the same data is present on all the nodes in the network, all participants know their data is consistent and coherent with their peers. This produces greater transparency of the club's data handling and processing among all staff and members.

Client-Server network

Peer-to-peer network

Figure 2. Client-Server vs. Peer-to-Peer network (*Peer-To-Peer Network*, 2018)

Immutability

Each peer node in a blockchain network holds a copy of the data in a ledger. The ledger is a transaction-based list of blocks with every operation (read or update) that has occurred in the network, starting with the first and ending with the most recent. The data blocks in a ledger are immutable and form the blockchain as shown in Figure 3. Each new block is linked to the previous entry with a hashcode. If an existing entry is altered, the hash code of the entry linked to it will have a discrepancy and indicate corruption of the data. This makes it very difficult to change an entry without having to change all the entries in the ledger. Immutability builds further trust in the blockchain.

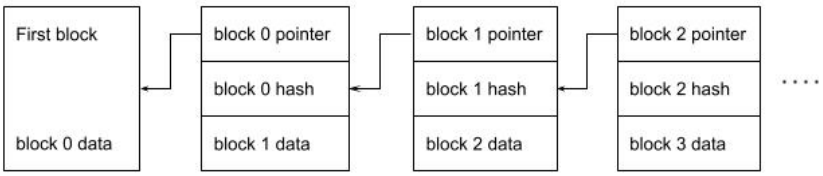

Figure 3. Link-list of blocks in a ledger

Immutability ensures a club has accurate data to use for planning goals and tracking member progress. Since blocks of data are permanently recorded on the ledger with hash codes, any intentional tampering of the data is immediately detected by the blockchain. The built-in safeguards against tampering allow members to trust the data stored on the blockchain.

Consensus

Before any data can be updated on a peer's ledger, the transaction must be approved by the other peers. Typically blockchains are configured to require at least 50% of the peers to approve the update before a new block is added to the blockchain. For someone to conceal their ledger updates, they would have to take control of the majority of nodes in the blockchain, which is practically difficult. The more peers in the network, the harder it is to manipulate the data on all peers.

Each transaction arriving at the node can be classified as a data query or data update. Peers have software code that is executed for each transaction. This code, which has been endorsed by all peers, implements the policies surrounding the handling of data in the blockchain and is known as chaincode. The same chaincode is implemented on all peers, and it governs all access to the assets' data. This establishes consensus among all the peers. Chaincode is also sometimes called a smart contract since it establishes agreement among the peers and runs autonomously.

Consensus ensures club member data is accessed with the same rules across all peers. Any query or update to member data must adhere to the chaincode policies that are present on all the peers. In our club, the policies adopted and approved by all staff are executed for all member transactions.

Security

Blockchains may be public or private. Most cryptocurrency blockchains, such as Bitcoin, are built on public blockchains. In public blockchains, any anonymous user can issue transactions on the blockchain. In contrast,

private blockchains only allow registered users to interact with the blockchain. These "permissioned" networks require the user to be authenticated before they can transact on the blockchain. The authentication is performed using public/private X509 TLS certificates, which provide a very high level of security (X.509, 2022).

Security is important to a school club as it is important to be careful with members' information. Users are registered with the Fabric and only authorized users have access to Fabric. Moreover, all data transfer is done with the Secure Shell Protocol (SSH) cryptographic keys to maintain data transfer integrity.

HyperLedger Fabric

Building a blockchain ground-up would be a herculean task. Fortunately, there are open-source blockchain platforms on which we can run our application. One of the most popular blockchain systems available is the open-source platform Hyperledger Fabric, designed by the Linux Foundation (Hyperledger Fabric, n.d.). We decided to use Hyperledger Fabric, or simply Fabric, since it is a mature and scalable technology supported by a large community of developers. IBM is a significant contributor to the open-source community and has built commercial solutions using HyperLedger Fabric and deployed them in large Fortune 500 companies (*IBM Blockchain - Enterprise Blockchain Solutions & Services*, n.d.).

The major components of HyperLedger Fabric are shown in Figure 4. It shows an example network with three peer nodes connected to form a channel for data sharing.

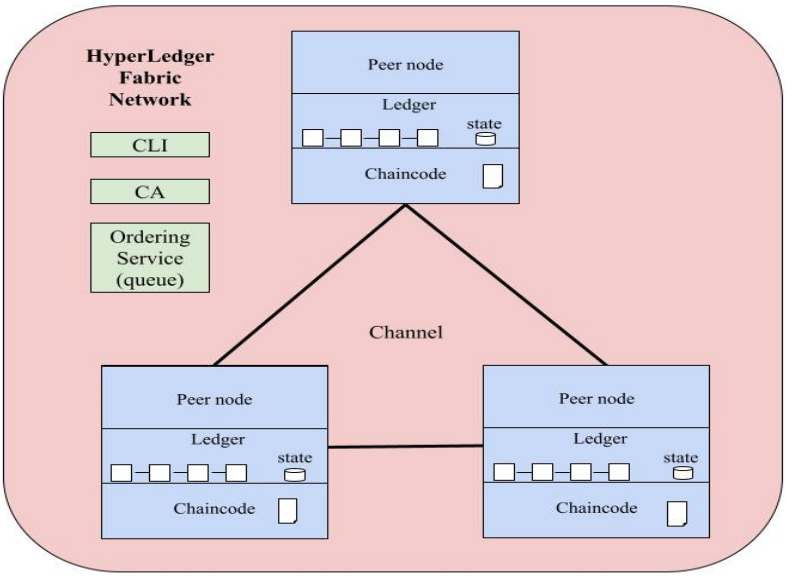

Figure 4. Three peers in a Hyperledger Fabric blockchain

We shall now describe these components to understand their role in the blockchain network.

Peer

A peer is the fundamental unit of a blockchain that stores assets. Each peer houses a ledger, a state database, and chaincode. The ledger contains all asset transactions that were sent to the peer. A transaction is sent to the peer as a block of data and the block is added to the ledger where it remains immutable.

As the ledger grows, it becomes time-consuming to search the ledger for the latest state of an asset. An asset may have been added, updated, deleted, and added again. To efficiently retrieve the latest state, each

ledger maintains a database, known as the world state, with the latest state of all assets tracked by the blockchain.

The read and update policy of any asset is defined in the chaincode. Chaincode governs how the asset's data in the ledger are accessed and updated. We can customize the chaincode to support the needs of our application.

In our prototype, there is one peer node for a coach, one peer node for an officer, and one peer node for all members. All members of the school's club are considered assets. Each asset has a unique identifier with all the relevant information for a member. Any query or update to a member is recorded on the peer's ledger, and updates are broadcast to other peers.

Channel

A channel is the private communication domain in a blockchain. All peers on a channel receive updates from one another about asset updates. All peers in a channel also share the same chaincode that governs access to the asset's data. In our prototype, we only configured a single channel, however, it is possible to configure multiple channels in the Fabric.

Certificate of Authority

As mentioned earlier blockchains may be private or public. Hyperledger Fabric is a private blockchain and requires users to register and be authenticated to access the network. Access to the Fabric is provided by the Certificate of Authority (CA). The CA issues certificates to registered users. The certificates use the X.509 protocol which uses private and public keys (X.509, 2022). The user signs their transactions with the private key that was provided to them by the CA. The user authentication is done by another component of the Fabric, the Membership Service Provider (MSP). It contains the public keys issued to all users. If the transaction is validated with the user's public key, the user is authorized to enter the Hyperledger Fabric network.

Orderer

Since blockchain keeps data in a distributed manner, updates to an asset are more complex than a traditional database using a client-server model. The assets in all peers in a channel must be updated methodically to prevent inconsistency between the ledgers. The function of the Orderer is to queue updates and broadcast the updates to all the ledgers once the transaction has been approved by all peers.

Command Line Interface

The Command Line Interface (CLI) is a process we can use to interact with the Hyperledger Fabric. From the terminal, we can directly issue commands to the ledger, such as creating a channel, downloading chaincode, initializing the ledger, querying the ledger, updating the ledger, and so on. It provides an interface through which we can test the downloaded chaincode. This is very useful as it allows us to test and verify our chaincode in isolation before calling it from an application server.

Prototype Development

To demonstrate the benefits in blockchain based clubs, we implemented a prototype of a blockchain network for a speech club using the Hyperledger Fabric. The network allows coaches to view all members, enroll new

members, and update member states. Members may view their current state and the history of state changes but they are not allowed to enroll members or view other member's data.

Coaches access the club with a browser client, which sends and receives messages from an application server connection that interfaces with the Fabric as shown in Figure 5. Similarly, members access the club with a browser client which sends and receives messages from a server connection for members.

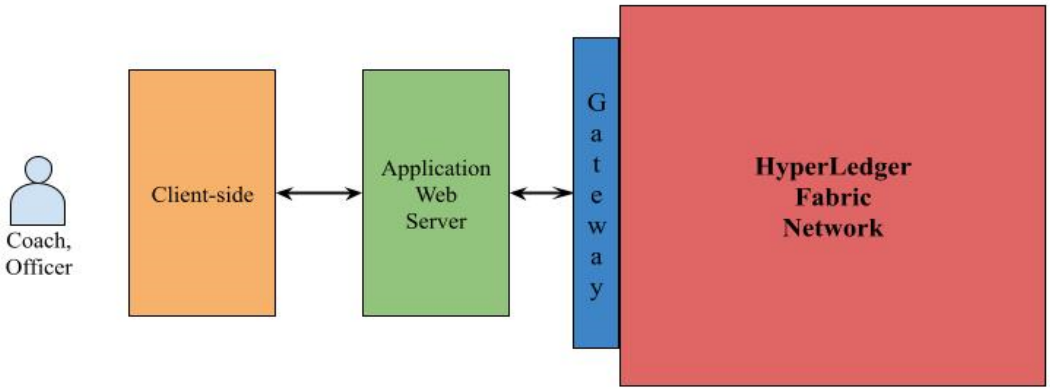

Figure 5. The pathway through which a coach connects to the blockchain network

Specifications

We want to proof-of-concept a blockchain-based speech club that coaches could use to manage their members. Coaches should be able to enroll new members, assign members tasks, and change member states as they progress through the season.

The state transitions are shown below. In our prototype, only coaches have the authority to transition members between states. Members can view their states and the history of state transitions. The criteria to determine when a member transitions states will be written into the chain code that has been approved by all coaches.

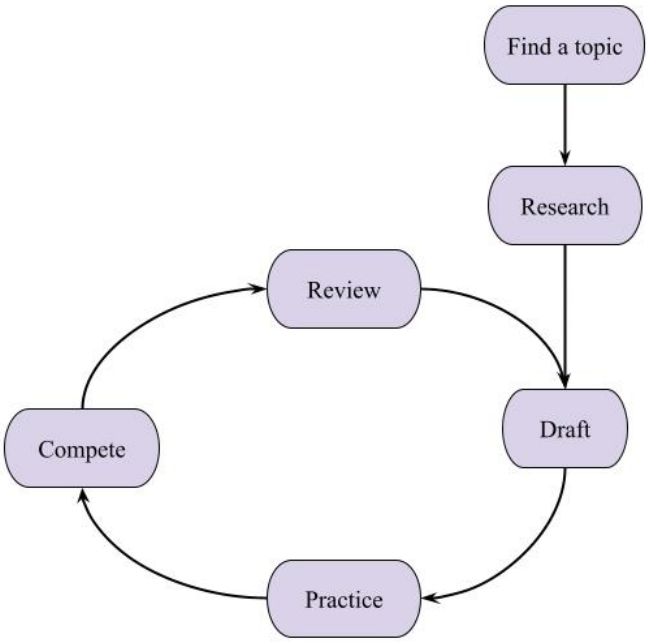

Figure 6. Different states for a speech club

The states described here may be modified for any club. For example, a club focused on tutoring might have the following states: selecting a class topic, developing lesson plans, creating handouts, and finally teaching. The state diagram is versatile and can be customized for a wide variety of clubs.

Fabric installation

Hyperledger Fabric is distributed by the Linux Foundation and is intended to run on a native Linux environment. The fabric images are conveniently packaged using Docker containers so they can be installed with most Linux distros. We installed Ubuntu Linux on a Windows machine with Windows Subsystem Linux (WSL) enabled. We then downloaded Docker Desktop and enabled it for Ubuntu Linux. From the Linux terminal, we cloned the Hyperledger Fabric git repository. We now had access to a sample network and sample applications.

The layering of Docker and Hyperledger Fabric on Linux is shown in Figure 7.

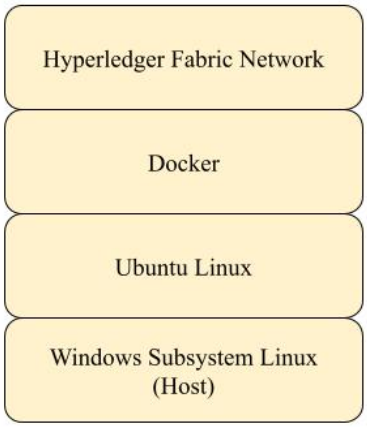

Figure 7. Application stack of HLF running on Windows

Our blockchain system will feature three peer nodes, as shown in Figure 4. One peer is for members to access the blockchain, one peer is for a coach, and the final is for a club officer. New peers may be added to the network by changing configuration files.

Chaincode

Chaincode refers to the policies we implement that govern the usage of the assets. Chaincode defines the rules of how the member data is to be accessed and updated. By defining these policies in code, there is no ambiguity about how members are managed, their progress tracked, and the state changes they undergo. For example, a member would first select a topic, then conduct research on it, and finally iterate between drafting and competing, as shown in Figure 6.

The club leadership determines a member's state and the criteria to transition between states. This would typically be done at the start of the season. Once an agreement is reached, the policies are implemented in chaincode, similar to the way a contract is written between parties. This is why chaincode are often called smart contracts, since they govern the club operations without any human intervention.

Once a Fabric channel is created, the chaincode is downloaded on all peer nodes. The Fabric is now ready to start accepting transactions. Chaincode may be upgraded as policies change, without disruption to the running fabric.

The chaincode is where we customize Fabric for our use case. The initial chaincode is built using four basic asset operations: Create, Read, Update, and Delete (CRUD). Figures 8-11 show the code snippets for the four basic chaincode operations.

```
/**
 * Creates a new asset on the ledger.
 *
 * @param ctx the transaction context
 * @param assetID the ID of the new asset
 * @param firstName the first name of the new asset
 * @param LastName the last name of the new asset
 * @param level the competition level for the new asset
 * @param topic the asset's topic for the school year
 * @param state the asset's coaching state
 * @return the created asset
 */
@Transaction(intent = Transaction.TYPE.SUBMIT)
public Asset CreateAsset(final Context ctx, final String assetID, final String firstName,
final String lastName, final String level, final String topic, final String state) {
    ChaincodeStub stub = ctx.getStub();

    if (AssetExists(ctx, assetID)) {
        String errorMessage = String.format("Asset %s already exists", assetID);
        System.out.println(errorMessage);
        throw new ChaincodeException(errorMessage, AssetTransferErrors.ASSET_ALREADY_EXISTS.toString());
    }

    Asset asset = new Asset(assetID, firstName, lastName, true, level, topic, state);
    String assetJSON = genson.serialize(asset);
    stub.putStringState(assetID, assetJSON);

    return asset;
}
```

Figure 8. Chaincode for Create Asset

```
  /**
   * Retrieves an asset with the specified ID from the ledger.
   *
   * @param ctx the transaction context
   * @param assetID the ID of the asset
   * @return the asset found on the ledger if there was one
   */
  @Transaction(intent = Transaction.TYPE.EVALUATE)
  public Asset ReadAsset(final Context ctx, final String assetID) {
      ChaincodeStub stub = ctx.getStub();
      String assetJSON = stub.getStringState(assetID);

      if (assetJSON == null ||_assetJSON.isEmpty()) {
          String errorMessage = String.format("Asset %s does not exist", assetID);
          System.out.println(errorMessage);
          throw new ChaincodeException(errorMessage, AssetTransferErrors.ASSET_NOT_FOUND.toString());
      }

      Asset asset = genson.deserialize(assetJSON, Asset.class);
      return asset;
  }
```

Figure 9. Chaincode for Read Asset

```java
/**
 * Updates the properties of an asset on the ledger.
 *
 * @param ctx the transaction context
 * @param assetID the ID of the new asset
 * @param level the competition level for the new asset
 * @param topic the asset's topic for the school year
 * @param state the asset's coaching state
 * @param preserve the asset's active state
 * @return the transferred asset
 */
@Transaction(intent = Transaction.TYPE.SUBMIT)
public Asset UpdateAsset(final Context ctx, final String assetID, final String level,
                         final String topic, final String state, final boolean preserve) {
    ChaincodeStub stub = ctx.getStub();

    if (!AssetExists(ctx, assetID)) {
        String errorMessage = String.format("Asset %s does not exist", assetID);
        System.out.println(errorMessage);
        throw new ChaincodeException(errorMessage, AssetTransferErrors.ASSET_NOT_FOUND.toString());
    }
    Asset oldAsset = ReadAsset(ctx, assetID);
    boolean active = preserve ? oldAsset.getActive() : false;

    Asset newAsset = new Asset(assetID, oldAsset.getFirstName(), oldAsset.getLastName(), active, level, topic, state);
    String newAssetJSON = genson.serialize(newAsset);
    stub.putStringState(assetID, newAssetJSON);

    return newAsset;
}
```

Figure 10. Chaincode for Update Asset

```java
    /**
     * Deletes asset on the ledger.
     *
     * @param ctx the transaction context
     * @param assetID the ID of the asset being deleted
     */
    @Transaction(intent = Transaction.TYPE.SUBMIT)
    public void DeleteAsset(final Context ctx, final String assetID) {
        ChaincodeStub stub = ctx.getStub();

        if (!AssetExists(ctx, assetID)) {
            String errorMessage = String.format("Asset %s does not exist", assetID);
            System.out.println(errorMessage);
            throw new ChaincodeException(errorMessage, AssetTransferErrors.ASSET_NOT_FOUND.toString());
        }

        stub.delState(assetID);
    }
```

Figure 11. Chaincode for Delete Asset

Application web server

The application web server in Figure 5 will be used to route client HTTP requests from the client to the Hyperledger Fabric. The web server interfaces with the Fabric using a software development kit (SDK) gateway. The SDK gateway abstracts the Fabric so we can use the REST API to query or update the Fabric. Responses from the Fabric are sent to the web server and finally to the client.

We used the Express.js framework to build the application web server. Express.js is built using the Node.js platform and allows us to quickly deploy a web server. Using Javascript code, we can quickly service a handful of client requests. We specify a host address and port number to listen on.

Client application

The client application, eg. browser, sends HTTP requests to the application web server, as shown in Figure 5. Responses from the web server are rendered by the client application.

We used React JS to build the client application. React JS provides fast and responsive libraries enabling us to quickly serve live pages on the browser. There are many online examples of React applications for Hyperledger Fabric (Peng, 2019).

Prototype Demonstration

We will now demonstrate how our simple prototype fulfills the data requirements listed in Table 1. We will use screen-shots of our application for this demonstration.

Transparency of data among coaches and members

Each coach queries their peer node and retrieves information about all the members in the club as shown in Figure 12. The identifying details of each member are listed in the first three columns. The next two columns show their level in the team and their selected topic. All the coaches and officers can see all active members in their club on this page and the progress of their speeches.

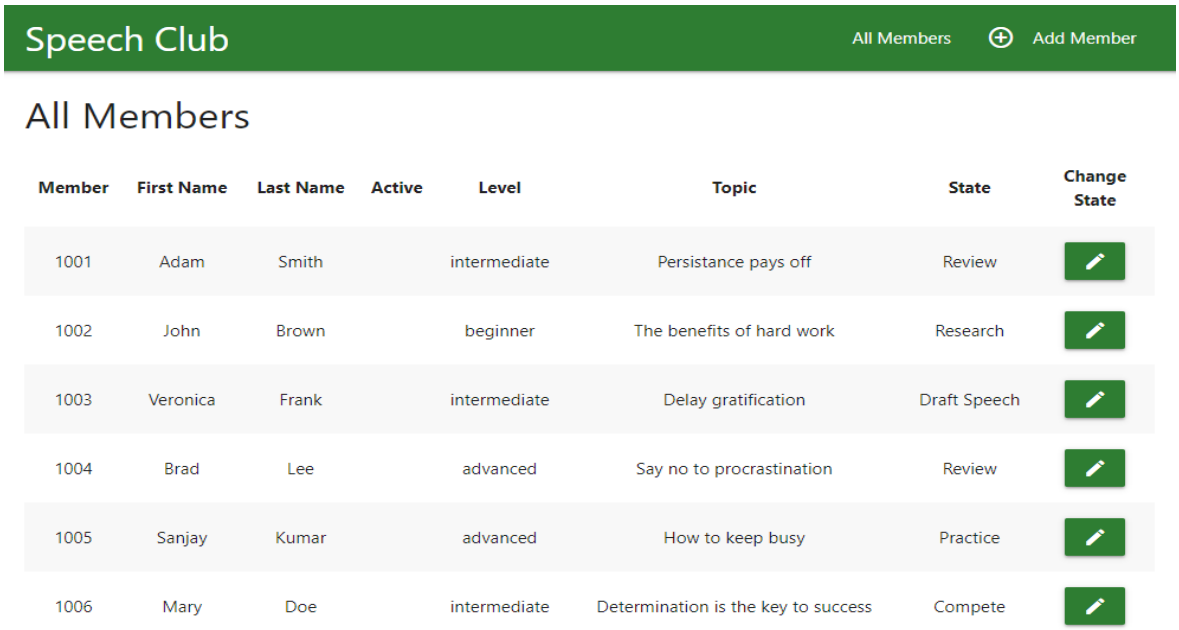

Figure 12. Coach's view of all members

Each member can also connect to their peer node and retrieve their information and current state, as shown in Figure 13. The member application only allows a member to view their details. The prototype does not allow a member to modify their state. New functionality may be added once the policy is defined and approved by the coaches.

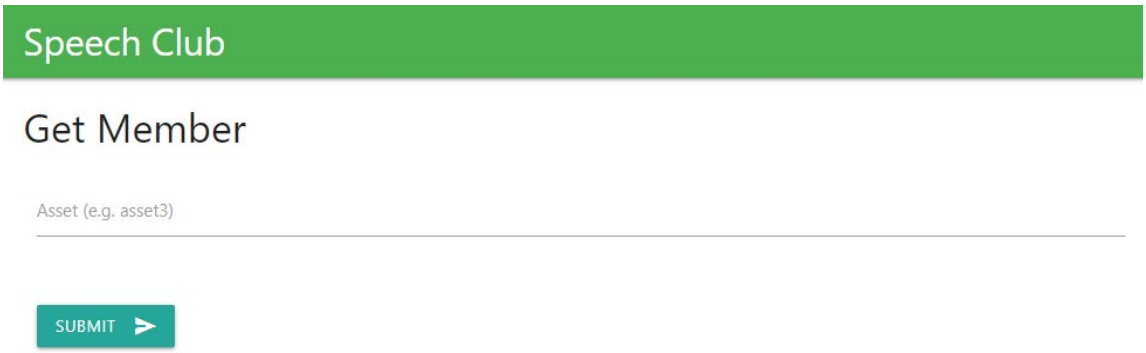

Figure 13. The member form

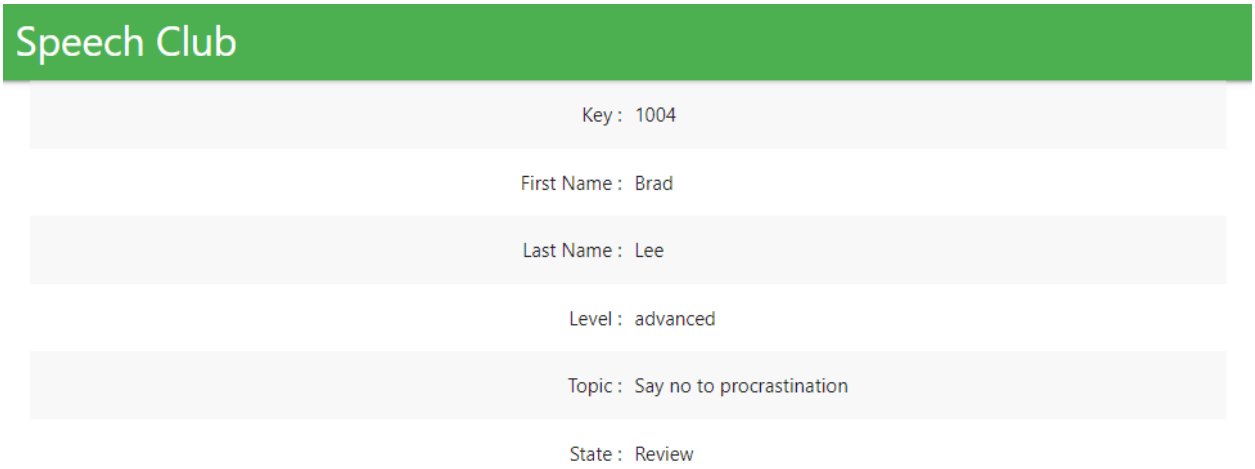

Figure 14. Results from a member query

A coach may add a new member to the blockchain using the CreateAsset chaincode in Figure 8. When a coach adds a member to its peer in Figure 15, the new member data is added to all the peer nodes, and is visible on the member peer, as seen in Figure 16. This demonstrates the decentralized nature of the blockchain.

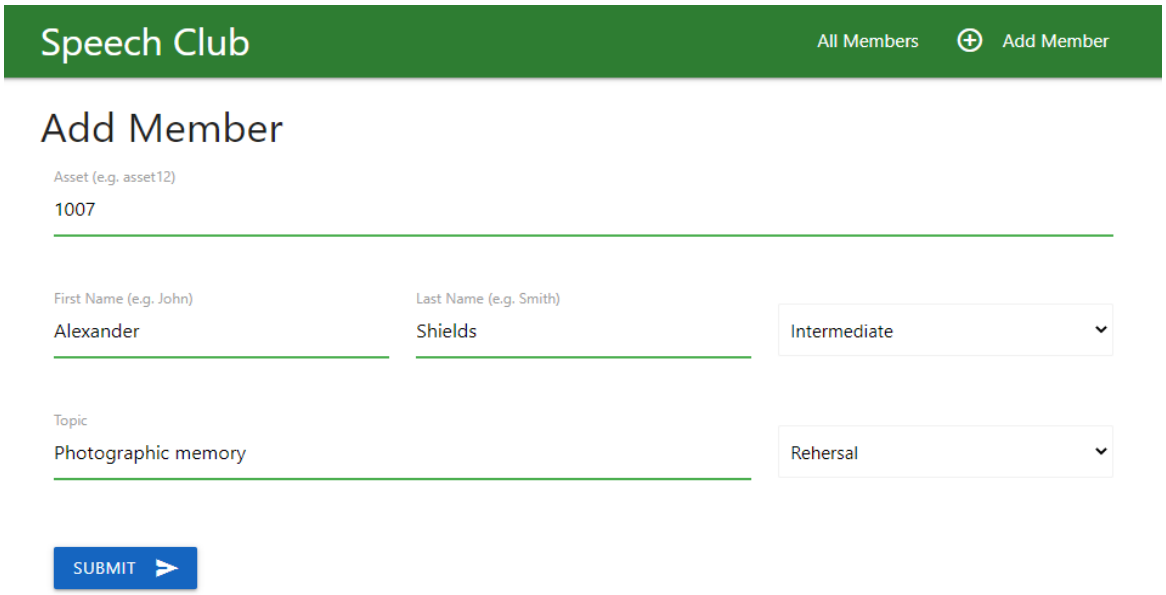

Figure 15. Coach adding a new member through the coach peer

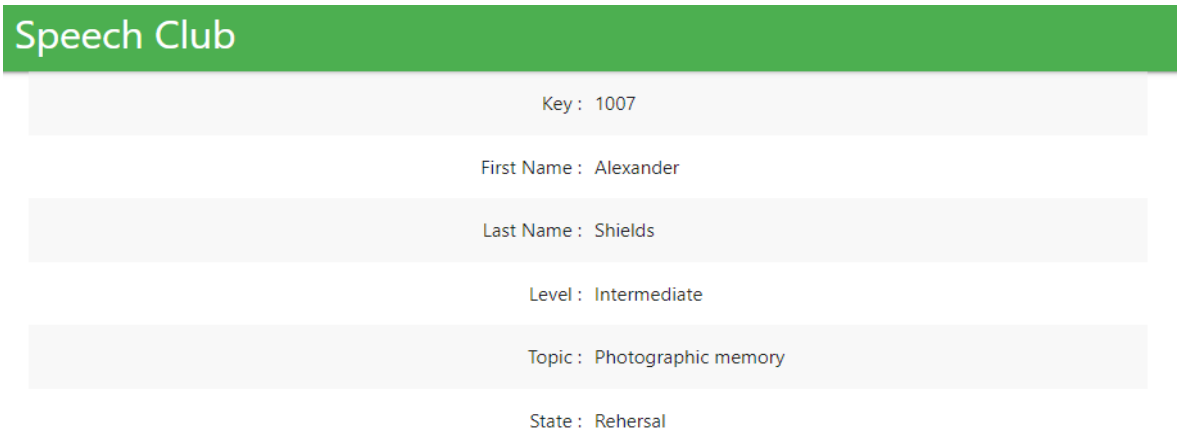

Figure 16. The new member is instantly visible on the member peer

Agreement among staff on club policies

All coaches have agreed on the member fields and possible states and implemented them in their chaincode. A coach may change the state of an existing member by calling the UpdateAsset chaincode from Figure 10 using the form shown in Figure 17. The permitted states and any legal transitions have been approved by all coaches and implemented in the chaincode. In this way, there is agreement among the coaches on permitted member states for the season.

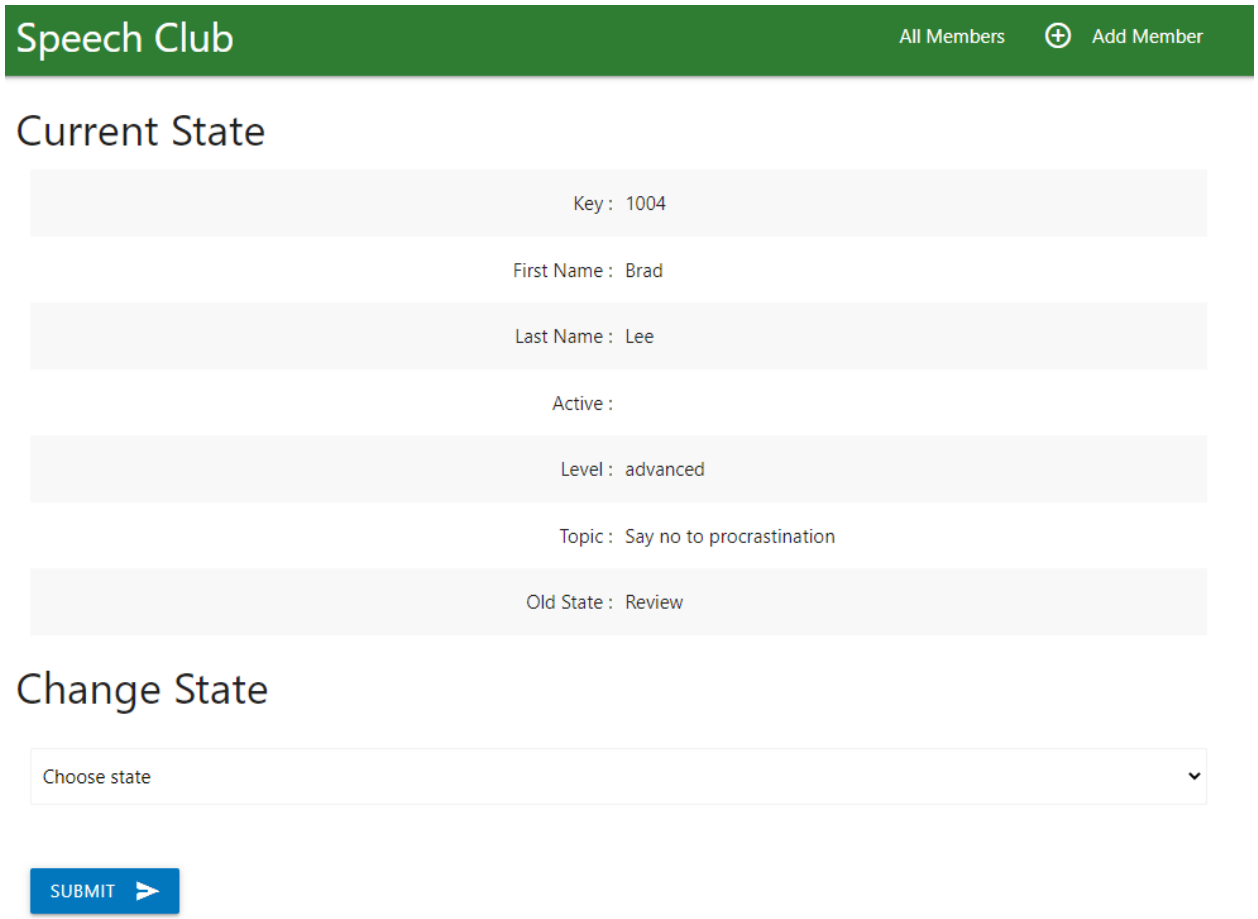

Figure 17. Changing a member state with UpdateAsset chaincode

Permanent logging of all transactions

All transactions in the peer are permanently logged in the ledger. This ensures that all member activity is recorded and can be retrieved. This immutability property of the blockchain allows us to trace any intentional or unintentional modifications to the member data.

We also included chaincode in the Fabric for our application to request the history of member state changes. Figure 18 shows an example of all the state updates for a specific member. In this example, the member has undergone six member state transactions and they are listed in reverse chronological order. Even if a member were deleted, this update would be recorded in the member's history and can be retrieved for inspection.

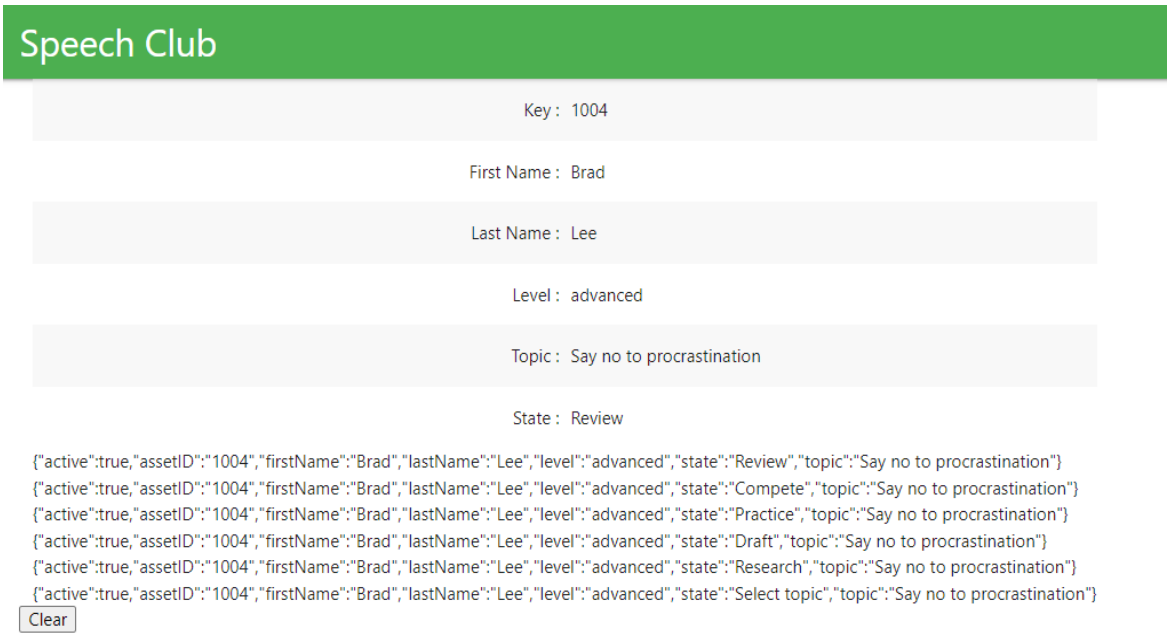

Figure 18. Record of all state changes to an asset

Private and Secure

Each peer node interfaces with an application web server as shown in Figure 5. Users with the authority to access the network are registered with the application server. Each registered user is provided a private key from the Fabric Certificate of Authority that is stored in the application server. When the user client issues a transaction, their transaction is encrypted with the private key and sent to the Fabric. The Fabric Membership Service Provider authenticates the transaction with the user's public key to verify the transaction originated from the indicated user. In this way, only registered users can issue requests to the Fabric.

Registered members can access the Fabric through the member application server that issues requests to the member peer node. Registered coaches can access the Fabric through the coach application server that issues requests to the coach peer node. The use of certificates of authority to authenticate each user before they can access the site ensures that all member information is being securely stored. Figure 19 shows the Fabric response when an unregistered user tries to send a request on the blockchain.

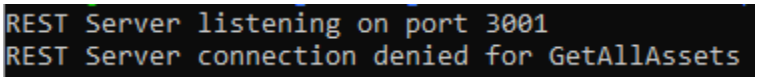

Figure 19. An unauthorized user is prevented from accessing the Fabric

Conclusion

This paper details a potential solution to the data management problem facing clubs at high schools. As club participation increases, club leadership has a responsibility to cater to the needs of their members and the community they serve in keeping with their mission statement. We have shown how member management is instrumental to the growth and success of any club and have detailed the shortcomings of traditional databases that rely on client-server networks. We have shown how a blockchain network of peers can meet the data requirements of a large club. We also demonstrated a functional prototype of a speech club based on a blockchain network. This model can be scaled up and used to improve the efficiency of any expanding club, without the

need for additional staff or resources. Finally, we hope the transparent and distributed structure of blockchains continues to be used in more applications for academic and extracurricular purposes.

Acknowledgments

I would like to thank my advisor for the valuable insight provided to me on this topic.

References

Canonical. (2004). *Ubuntu Desktop* (20.10). Retrieved from https://ubuntu.com/download/desktop

Docker. (2013). *Docker Desktop*. Retrieved January 2, 2022, from https://www.docker.com/products/docker-desktop/

Facebook Open Source. (2013). *React.JS* (18.1.0). Retrieved 18, March 2022 from https://reactjs.org/

Hyperledger Foundation. (2019). *HyperLedger Fabric*. Retrieved January 3, 2022, from https://www.hyperledger.org

IBM Blockchain - Enterprise Blockchain Solutions & Services. (n.d.). IBM. Retrieved January 27, 2022, from https://www.ibm.com/blockchain

Introduction. (n.d.). Hyperledger Fabric Docs. Retrieved January 12, 2022, from https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html#hyperledger-fabric

Khorram, Y. (2022, May 4). *California governor signs executive order on cryptocurrencies*. CNBC. Retrieved May 19, 2022, from https://www.cnbc.com/2022/05/04/california-governor-signs-executive-order-on-cryptocurrencies.html

Maurya, A. (2021, May 9). *Hyperledger Fabric on Windows*. Codementor. Retrieved February 23, 2022, from https://www.codementor.io/@arvindmaurya/hyperledger-fabric-on-windows-1hjjorw68p

OpenJS Foundation. (2009). *Node.JS* (18.2.0). Retrieved March 17, 2022 from https://nodejs.org/en/

Peer-to-peer Network. (2018, December 10). BitcoinWiki. Retrieved April 8, 2022, from https://en.bitcoinwiki.org/index.php?title=Peer-to-peer&direction=prev&oldid=381025

Peng, C. (2019, September 5). Hyperledger Fabric Fabcar Client App — Chhaileng Peng. *Chhaileng Peng*. Retrieved February 21, 2022, from https://www.chhaileng.com/blog/hyperledger-fabric-fabcar-client-app

X.509. (2022, April 18). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=X.509&direction=prev&oldid=1087373729