# ExChain: A University Record Exchange System

**Tre' R. Jeter** **Steven R. Rosenthal** **Yashasvi Mutteneni**
Department of Computer & Information Science & Engineering, University of Florida
Email: t.jeter@ufl.edu, stevenrosenthal@ufl.edu, y.mutteneni@ufl.edu

## I. Abstract

Blockchain technology has proven to have great potential in many sectors despite its challenges. It has the opportunity to change the foundation of our ever-advancing technological society. We propose a system, ExChain, that revolutionizes record keeping in the domain of higher education. Owing to the immutability and transparency of blockchain, our proposed system will supplement existing legacy database systems currently used to ensure the efficient, secure, accurate, and permanent management of student records.

## II. Introduction and Motivation

Researchers and developers alike continue to explore potential applications unlocked by blockchain technology. Cryptocurrencies like Bitcoin initiated the public's growing attention toward blockchain; however, the area has seen a considerable breadth of new developments. Decentralized applications, DApps, built upon the foundations of smart contracts, leverage the intrinsic characteristics of blockchain. These features, such as immutability and speed, draw many sectors towards incorporating blockchain technologies into their record management infrastructure.

Regardless, organizations adopting blockchain-based ledger systems may prove to be complicated. Converting existing databases to this new technology is a task many organizations may not want to pursue upfront. Blockchain databases will probably not be considered a potential alternative, despite the technical advantages, until it becomes mainstream and available talent exists to maintain such systems. However, organizations could leverage the benefits of blockchain through interoperability without internal usage.

This project, ExChain, seeks to investigate using blockchain as an intercommunication medium between collaborating organizations that maintain extensive, differently implemented, and differently marshaled databases. The higher education system is a collaboration circumstance exhibiting these attributes. Universities often need to transfer student records. The process of transferring course credits and transcripts between universities and hiring industries, respectively, costs bureaucratic overhead that can be slow and erroneous. Using a blockchain as a global student record system to transfer student records could enable higher security and speeds than current methods.

## III. Related Work

The goal of blockchain is to create a decentralized environment where transactions can take place in a trustless environment [12]. Blockchain technology can be used in the field of higher education. Immutability, provenance, and peer-executed smart contracts are some of the qualities of a blockchain that could bring a new level of security, trust, and transparency to e-learning. As a smart contract, it may also be viewed as a consensus mechanism that rewards students, instructors, and universities [11]. As a result, technology has been employed to enhance higher education. It also helps people who are less well-informed to interact with more knowledgeable peers and mentors.

In [10], the authors present a proof-of-concept blockchain-based e-learning platform, which was created to improve assessment transparency and facilitate curriculum personalization in higher education. The software could, for example, automate assessments and issue credentials. Its application transforms the concept of student-professor contact, making education more accessible and individualized. A personal plan targeted at lifelong education has become an objective need in recent years, and blockchain technology now provides the resources needed to make it a reality. At the same time, blockchain technology has the potential to create inequalities in access to online and offline schooling. It's worth noting that most colleges that have created educational blockchain technologies are still using them as a supplement to traditional modes of instruction [9].

EduCTX [1] proposed a global blockchain-based higher education credit platform. The European Credit Transfer and Accumulation System is the foundation for this platform (ECTS). It is a worldwide trusted, decentralized higher education credit and grading system that may provide students and higher education institutions (HEIs), as well as other prospective stakeholders such as enterprises, institutions, and organizations, with a globally united perspective. The authors offer a prototype implementation of the ecosystem-based on the open-source Ark Blockchain Platform as a proof of concept. EduCTX will process, manage, and govern ECTX tokens, which represent credits earned by students for finished courses such as ECTS, using a globally distributed peer-to-peer network. The blockchain network's peers are known as HEIs. A prototype has been built on the ARK blockchain platform and the authors propose to apply it on their home university, the University of Maribor. They also plan to adapt the EduCTX blockchain so that each course is assigned a unique blockchain address and a pool of tokens. The students receive tokens directly from the course address after completing a course. The course address would be a multi-signature address between the institution and a professor. [2] Experimentally implements a blockchain-based university transcript system and describes some results.

In [3] a proposal for a UniChain-based blockchain-based system for handling Electronic Academic Records (EARs). UniChain is intended to improve current management systems by allowing students, universities, and other third parties to have interoperable, secure, and effective access to EARs while maintaining the students' privacy. UniChain governs transactions and controls access to EARs using timed-based smart contracts. For added security, it employs modern encryption techniques. This paper presents a new incentive mechanism that makes use of universities' efforts to keep academic records up to date and create new blocks. Instead of evaluating the UniChain performance on different aspects, such as response time, throughput, and evaluating just creating a digital performance currency, the proposed mechanism rewards the "block's creator" with an incentive to be added to its degree, thereby decreasing its likelihood of recreating the next block, the proposed mechanism rewards the "block's creator" with an incentive to be added to its degree, thereby achieving fairness and equality among universities the UniChain different aspects, such as throughput, and The results show that the response proposal time is efficient in managing a large number of requests while also assuring the system's long-term viability. The results show that the concept is effective at managing a huge dataset with low latency. In [4] the study offers a relay-based multi-blockchain architecture that allows for cross-blockchain interaction between different blockchains. They establish four types of node roles in the multi-blockchain architecture, develop a cross-blockchain connection model, and discuss the functions of each portion. In a multi-blockchain design, the relay node primarily serves as an information adapter for heterogeneous blockchains, while the coordination node keeps track of routing information and cross-blockchain transaction identifiers. Furthermore, the cross-blockchain protocol's message format, cross-blockchain inquiry, and cross-blockchain transaction can match the privacy and consistency criteria of cross-blockchain communication.

To increase the scalability of the whole blockchain network, the study in [5] suggests a one-time cross-chain contract and gossip network. To execute the operation, each blockchain system follows a contract, and the results are tied to the system. As a result, the entire system achieves linear scalability and consistency. Interoperability in blockchain systems is the capacity to connect several blockchain networks together, increasing scalability and resolving connectivity concerns [7]. The authors evaluate and contrast the architectures of existing cross-blockchain communication solutions. They come to the conclusion that there is no complete interoperable architecture capable of meeting the needs of the industry ecosystem.

There are research gaps, such as using smart contracts to develop an interoperable protocol between homogeneous blockchains and the ability to share apps and smart contracts. [8] presented an interactive multiple blockchain architecture for information exchange across any blockchain system. An inter-blockchain connection model was employed to govern routing across various blockchains, and a three-phase commit was used to confirm the communication result. However, the authors did not include encryption or access control in the inter-blockchain communication architecture to improve security. Our model, as explained in the following sections, addresses these concerns.

## IV. Scope of Project

### A. Scope

The authors take an investigative approach on how universities currently manage their record keeping infrastructure and reveal key disparities of current university record keeping initiatives. A determined end-goal is to explain the feasibility of integrating blockchain technology into the higher education system while describing the overall benefits of an inter-university blockchain record keeping system. The paper will conclude by comparing the advantages and disadvantages of the proposed system and the current structure used by universities.

### *B. System Architecture*

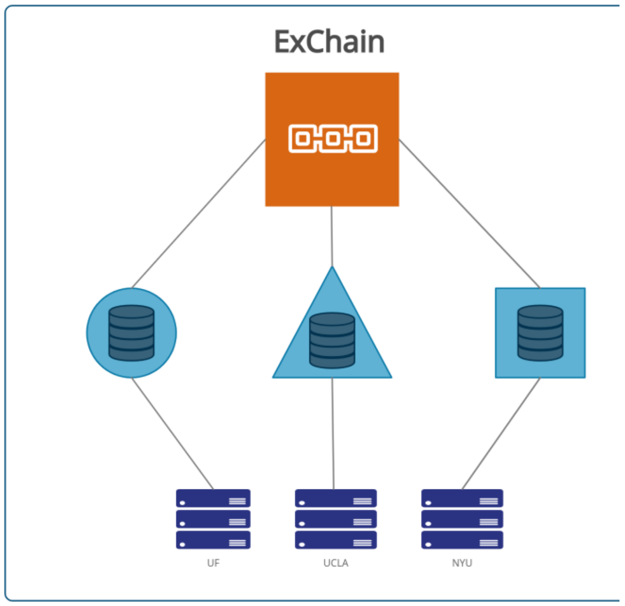


*Figure 1. Simple System Architecture of ExChain*

ExChain will act as an extension to existing university databases. The existing systems will continue to maintain their data while also pushing it to the blockchain. Each institution will then have a unified system for easier tracking and verification of student records across each member of the network. However, university representatives that collaborate through ExChain will have read-only privileges for public student records of other institutions. ExChain will stand as a private, permission-based ecosystem. When data is pushed to the blockchain, it is marshaled into a designated block representing a university. Further research is needed to determine the proper, secure mechanisms for publishing, updating, and retrieving records in ExChain.

## V. CHALLENGES OF BLOCKCHAIN

As blockchain supplies many benefits, it also has its challenges. These challenges include but are not limited to, security, interoperability, and computational costs. Integration of blockchain technology into more real-world scenarios requires further research about targeting such challenges.

### *A. Vulnerability Issues and Known Attacks*

Contrary to popular belief, the immutable nature of blockchain does not makes it inherently secure. This technology is riddled with vulnerabilities at the Application Layer, Data Layer, Consensus Layer, and Network Layer. These vulnerabilities open the blockchain up to numerous attacks at each layer as well.

Vulnerabilities at the Application Layer include but are not limited to, Reentrancy, Delegatecall Injection, Frozen Ether, Upgradable Contract, and Erroneous Visibility. The Reentrancy vulnerability appears when a smart contract's control-flow decisions rely heavily on state variables that are not updated by the smart contract before calling another smart contract. The result is allowing an attacker to forgo the validity checks used within the caller smart contract and steal Ether until it is all gone or until the smart contract runs out of gas. The Delegatecall Injection vulnerability was first seen in the Parity wallet. The Ethereum Virtual Machine provides the *delegatecall* opcode that inserts a smart contract's bytecode into the bytecode of a caller smart contract. An attacker can directly manipulate the state variables of a caller smart contract by injecting it with a malicious smart contract's bytecode. Frozen Ether is the result of a smart contract not including functions that allow spending. Users can deposit money from other smart contracts, but not spend it, deeming their money frozen. The Upgradable Contract vulnerability allows smart contracts to be upgraded even after deployment. As this was beneficial to mitigate previous smart contract problems, if the smart contract creator is malicious, then so will be smart contract upgrades. Erroneous Visibility is where the visibility of a function in smart contract is incorrectly declared. This can lead to unauthorized access to outside smart contract programmers that could be malicious.

A notable vulnerability at the Data Layer is the Indistinguishable Chains vulnerability. This occurred in the replay attack that split Ethereum into ETH and ETC. The result was that a transaction executed in one chain was able to be reused in another chain. This vulnerability has however been mitigated by including chainID's to chain fields.

At the Consensus Layer, notable vulnerabilities are the Outsourceable Puzzle and Verifier's Dilemma. Ethereum implements the Proof-of-Work (PoW) method known as Ethash that limits the use of parallel computing to solve puzzles. Miners are still able to divide puzzle searching solutions into many smaller tasks to be outsourced. This is the result of Ethash making a puzzle solution partially sequential in preimage search instead of PoW. Verifier's Dilemma is the result of high computational cost in verifying transactions by miners. Miners can spend a large amount of time verifying a computationally heavy transaction allowing attackers an advantage in finding the next block. Miners could also blindly verify transactions, but this opens the possibility of an incorrect transaction being included in the blockchain.

At the Network Layer, blockchain vulnerabilities include but are not limited to, Unlimited Nodes Creation, Fixed Peer Selection, and Sole Block Synchronization. Unlimited Nodes Creation happens in Ethereum where an attacker can create an unlimited number of nodes on a machine. With the unlimited creation of nodes, an attacker can take over incoming and outgoing connections of victim nodes, effectively isolating their victims. Fixed Peer Selection is seen in Geth. Geth clients always select the heads of randomly chosen buckets when choosing nodes from their routing tables to create outgoing connections. An attacker could always be chosen by periodically sending messages to the Geth client since nodes in buckets are chosen based on level of activity. This vulnerability is resolved in Geth version 1.9. The Sole Block Synchronization vulnerability occurs when a client reports a block with a value in its difficulty field that is greater than the total difficulty of all blocks up to that point. In Ethereum, clients can only synchronize with one other client at a time. An attacker could delay the synchronization which forces the other client to reject all incoming blocks. This could be the root of double-spending and Denial-of-Service attacks. The workaround here is to have clients synchronized with multiple clients. This however does increase the overall latency of the network.

Some attacks include but are not limited to, Key Attacks, Identity Attacks, Manipulation-based Attacks, Service Attacks, and Malware Attacks [6]. As the technology has progressed, some of these attacks have been mitigated. Consequentially, some attacks have even been deemed impossible now, but it is important to be aware of these possible attack vectors, as they could pose new security challenges in the future.

Blockchain makes use of cryptographic hash functions like SHA-256 for key generation. SHA-256, and similar algorithms, is susceptible to length extension attacks that occur when a malicious user appends data to an originally hashed message. Length Extension Attacks can be combined with the Replay Attack, a type of identity attack, that occurs when a malicious user steals a hash key from a network between two valid users. In this case, the malicious user can reuse the stolen hash key to illegally validate themselves as a legitimate user.

Blockchains are also susceptible to several manipulation-based attacks. An example of such attack is the Transaction Malleability Attack. In this attack, transactions can be altered after being created but not before being added to a block. This leads to differing transaction IDs once the block is generated. Another instance of this kind of attack is a Timejacking Attack. In this attack, a target nodes timestamp is skewed compared to the timeframe of its supporting network.

Blockchains also have a few service vulnerabilities. One being the chance of Distributed Denial of Service Attacks because the blockchain network is established with an internet connection. Others include double spending, collusion, and the vector76 attack. Malware also plays a role in blockchain vulnerabilities. Malicious mining software can operate undetected in JavaScript applications and execute in the browser. Malware takes advantage of the blockchains immutable nature because a block may contain a malware riddled transaction that will be permanently on the blockchain, and every subsequent block would be affected by this malware.

### *B. Blockchain Interoperability*

Major issues of blockchain inter-communication stems from the necessity of developing trust at the business, platform, and infrastructure levels in trustless environment. The communication between two blockchains requires the agreeance from a business perspective, platform level, and infrastructure view.

From a business perspective, each party in a blockchain-based system has to agree to a form of governance, data standardization, and an overall framework. At the platform level, each party must agree on a consensus mechanism to follow between each blockchain, smart contract setup between blockchains, and how each party is to be authenticated and/or given authorization while working between both blockchains. Lastly, the infrastructure is crucial to the function of an interoperable framework. To reduce computational costs on home machines, using a hybrid cloud approach to host the communication channels between two blockchains is theoretically a viable solution. However, a hybrid cloud approach is not an established method of deployment to handle the workload and governance requirements of large corporations. Such an approach would expose vulnerabilities work and lacks sufficient governance models that would expose vulnerabilities at the legal and network levels. At the infrastructure level, management is a challenge because there are two entities with two different ways of managing their own blockchains.

### *C. Computational Costs*

The computational costs incurred by implementing a blockchain in a system use can be enormous compared to conventional computational tasks. Blockchains typically implement cryptographic hashing as a part of their infrastructure. Generating cryptographic hashes happens very quickly, but the lookup time for a hash on a blockchain will increase proportionally to how many blocks are added. The time it takes to generate a block is also an incurred computational cost. Depending on how the blockchain is implemented can also play a significant role in incurring computational costs. The consensus algorithm used is where this computational cost will come from. Proof-of-Work (PoW), Proof-of-Stake (PoS), Proof-of-Authentication (PoAh), etc. all have different levels of computational cost because of the way they're implemented.

## VI. Challenges of ExChain

Designing ExChain has its own unique challenges compared to implementing general blockchains. These challenges include, but are not limited to, overcoming the disparity in university database implementations, the migration of data from these conventional "legacy" databases to blockchain environments, scalability, and generality.

### *A. Legacy Database Setbacks*

Legacy databases are known to be vulnerable to many malicious attacks such as SQL injections, unauthorized privilege escalation, weak authentication protocols, Denial of Service (DoS) attacks, etc. However, convincing universities to abandon their legacy record keeping infrastructure in place of a new, blockchain-based infrastructure is one of likely low initial success.

An SQL injection is a common attack on legacy databases. This occurs when specific malicious SQL statements are entered into a software applications entry field and executed. These SQL commands can be the cause of incorrectly inputted strings and characters that allow for unauthorized database control.

Unauthorized privilege escalation is the result of design flaws in operating systems or software applications such as databases. If a legacy database was incorrectly configured or its updates were not performed correctly, certain applications and/or users can gain access to normally non-accessible features.

Legacy databases are known to undergo Denial-of-Service attacks at some point. This type of attack is usually seen on networks where an attacker or malicious application floods their target with garbage traffic. Some information can also execute crashes of the system. In a database setting, a Denial-of-Service attack will slow down database operations tremendously and even crash the database. It is possible that no data is lost or destroyed, but the time and effort needed to revamp the database makes for an extremely tedious task.

ExChain could mitigate problems associated with the abovementioned vulnerabilities along with many more. ExChain could assist with record verification and inaccurate timestamps without having to make a complete shift. Inheriting from the blockchains' innate characteristics, blocks containing a copy of a university's records on ExChain cannot be altered, only updated with new blocks. ExChain would act as a global, immutable ledger for multiple universities' records across points in time. Therefore, ExChain would not secure existing non-blockchain-based databases but does not require a using party to implement a blockchain database for itself. Rather, ExChain acts as an external way to securely store snapshots of these university databases.

### *B. ExChain Interoperability*

A major challenge that ExChain faces in its development phase is adopted from blockchain technology interoperability. The challenge sits at the intersection of network communications and security. The goal is to have a system as a private, permission-based entity that is only accessible by a representative or team of representatives from each member institution in the network. As this resolves some security issues, it does not solve them all. The communication between standard databases and a blockchain environment prove to be difficult for 1) The underlying network protocols used in each 2) The restricted access of current databases compared to the internet use of a blockchain and 3) The compatibility of each set of technology.

With this said, currently universities send secure emails or paper copies of student records. Once the interoperability challenge is overcome, ExChain will also address the issue of increasing the speed of record receipt and the avoidance of human tendency to misplace documents.

### *C. Scalability*

Scalability also presents a challenge as new students enter universities each year and their records must be added to ExChain. The computational cost of repeatedly updating previous student records and adding new ones must be done with identical performance of legacy systems or better for ExChain to be adopted. Given blockchains quick verification protocols and previously observed performance, this should be feasible in theory.

### *D. Generality*

Designing ExChain for generality is important, as such a useful and powerful system could be used in other domains such as health, government, and large corporate settings. The challenge here is to prove the theoretical advantages of ExChain and to drive the fact that ExChain is beneficial in a number of environments other than the higher education system.

## VII. SYSTEM DESIGN

### A. *Architecture*

We propose an architectural design that allows each participating institution in ExChain to maintain their local database while supplementing their infrastructure by storing snapshots of that local database in ExChain. Each member institution will have an Ethereum wallet such as Metamask.

Each institutions database will store a reference to a wallet and that wallet address is how an institution will interact and be referenced in ExChain. This overcomes the challenge of technological incompatibilities between legacy databases and blockchain technology. It also mitigates the hassle of creating a universal data migration solution for legacy databases and blockchain-based environments.

ExChain acts as a container for each institutions own independent blockchain. Each institution can be referenced to by ExChain, but data from each school is only pushed to its respective blockchain within ExChain. The system itself follows a basic protocol with a simple interface and eventually an advanced interface. Figure 2 gives a more detailed illustration of ExChain and how it aims to supplement the existing infrastructure of university record keeping.

### B. *Simple Interface Description*

The basic smart contract interface allows the representatives of member institutions to read and write snapshots of other institution databases and snapshots of their own databases, respectively. Representatives of each member institution will have the privilege of reading another institutions data if and only if a block is not in the process of being added. They will also be able to write data to their own independent blockchain within the ExChain container. The sole purpose of ExChain is to supplement legacy databases. The coinRetrieval function allows institutions to request tokens. These tokens are used to submit copies of their databases at the end of each semester to their independent blockchain within ExChain.

### C. *Advanced Interface Description*

The advanced smart contract interface implements the governance aspect that manages individual institution's ability to read and write data to ExChain. The governance interface is further divided into school and organization interfaces. An important aspect of the organization interface is the presence of a "manager". One institution is randomly chosen to be the manager and this random selection happens at the end of each semester. The manager has the overseeing ability to admit and release institutions to and from ExChain while also maintaining its own data on ExChain. The school interface can be made up of several functions such as apply, withdraw, voteS, elect, voteM, and coinApply. The apply function allows an institution to apply to be a participant of ExChain and the withdraw function enables an institution to leave ExChain. However, both the apply

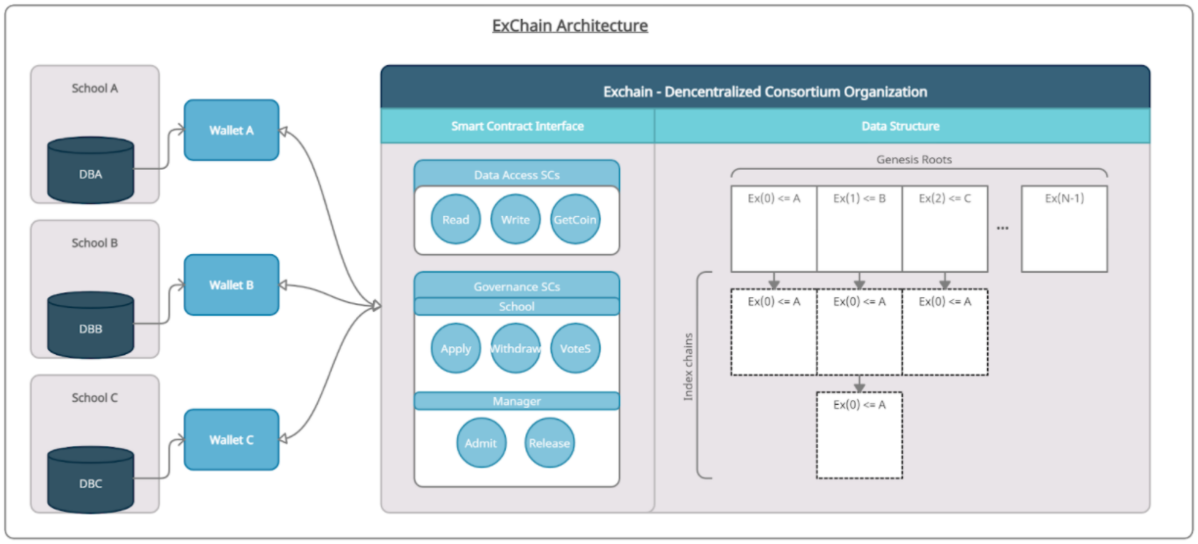


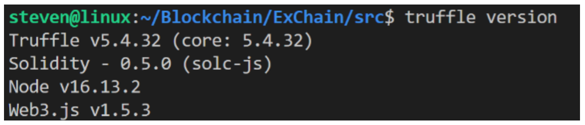
*Figure 2. Advanced ExChain System Architecture*

and withdraw functions need final approval from the elected manager to execute. In one form of the governance model, there exists an elect function where a school has the opportunity to nominate themselves or another institution to contest for the manager position in a voting process. The voteM function allows institutions to vote for one representative to be the manager of the ExChain. This election process of electing a manager may cause security issues. A better alternative to the previously stated election process would be randomly selecting a participating institution to be the manager for a specific time period (like at the end of each semester). In addition to voting for a manager, members of the ExChain can vote for whether a school should be added to ExChain or not through the voteS function. The final admittance to ExChain is done by the manager as described above in the organizational interface. One last function of the school interface is the coinApply function. Each school needs tokens to be able to submit their records to the ExChain at the end of the semester. The coinApply function allows school to request more tokens which can be used for the above-mentioned purpose.

## VIII. Implementation

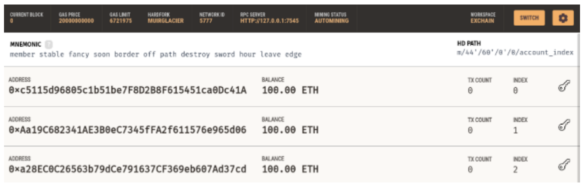
### A. Environmental Setup

To demonstrate the base functionality of ExChain, a portion of its functionality was implemented on an Ethereum-based test network hosted on an Ubuntu Virtual Machine. The Truffle Framework was used to develop and test the associated ExChain smart contract. Figure 3 lists the relevant truffle stack version information used in testing this decentralized application. Ganache was also utilized to create a personal blockchain to simulate Metamask wallets proposed in the top-level design in Figure 2. The Ganache workspace used for testing is shown in Figure 4. Please note that this workspace was deleted after testing and all associated addresses were deleted after testing for security purposes.



```
steven@linux:~/Blockchain/ExChain/src$ truffle version
Truffle v5.4.32 (core: 5.4.32)
Solidity - 0.5.0 (solc-js)
Node v16.13.2
Web3.js v1.5.3
```

*Figure 3. Truffle Stack Versions*



```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK     NETWORK ID  RPC SERVER             MINING STATUS   WORKSPACE
0              20000000000  6721975    MUIRGLACIER  5777        HTTP://127.0.0.1:7545  AUTOMINING      EXCHAIN     SWITCH

MNEMONIC                                                                        HD PATH
member stable fancy soon border off path destroy sword hour leave edge          m/44'/60'/0'/0/account_index

ADDRESS                                     BALANCE       TX COUNT  INDEX
0×c5115d96805c1b51be7F8D2B8F615451ca0Dc41A  100.00 ETH    0         0

ADDRESS                                     BALANCE       TX COUNT  INDEX
0×Aa19C682341AE3B0eC7345fFA2f611576e965d06  100.00 ETH    0         1

ADDRESS                                     BALANCE       TX COUNT  INDEX
0×a28EC0C26563b79dCe791637CF369eb607Ad37cd  100.00 ETH    0         2
```

*Figure 4. Ganache Workspace Used for Testing*

### B. ExChain Database

A database's structure, as seen by ExChain, is outlined in Figure 5. A database holds a payload, or a reference to a payload, which is represented by the encapsulated string *data* within the *DB* structure. Each database must be uniquely identified and is done by the following three elements: *uid*, *name*, and *owner*. A unique ID, *uid*, for each database is created via the hashing of the proposed human friendly string *name*. Each database name can only be added to ExChain's collection of databases once (see Figure 7 for more detail). An *owner* is also assigned as a field to the database.

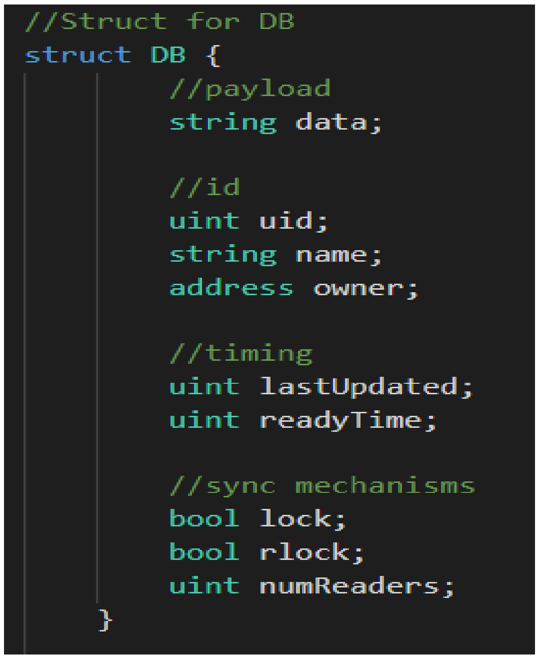
To ensure a level of security, a cooldown period has been added via the additions of *lastUpdated* and *readyTime* fields. These fields hold Unix time information related to how soon the next write execution may occur to a database. The current cooldown is set to 10 seconds for testing purposes. However, this should be scaled to mirror the known amount of writes that generally occur on the target record system. Synchronization mechanisms are also an important consideration as blockchains, such as Ethereum, expand to non-sequential forms of computing like sharding. As such, this sample ExChain system implements a reader-writer synchronization that is bootstrapped by the *lock*, *rlock*, and *numReaders* fields in the *DB* structure.

### C. Interface

The implemented ExChain decentralized application, in ExChain.sol, exposes four interface functions to demonstrate the potential functionality of a more complex, fully implemented, blockchain-enabled higher education record system. These interface functions are core components to the operation of ExChain.

*createDB()* allows for the head entity, or deployer/owner of the ExChain smart contract, to create database entries within the *dbs* array in Figure 7. Only the head entity may create databases within this array because the *onlyOwner* modifier from the Ownable smart contract requires it for the functions execution. *createDB()* accepts the proposed *_name* of the database and the wallet address the database's owner as *_setOwner*. The function also requires four other conditions to execute. The owner being set by the head entity may not own any existing databases to maintain a 1-to-1 relationship between ExChain participants and their storage. The head entity may not own a database and address 0, a commonly used "null" address in most smart contracts, are the second and third constraints. Finally, the name being proposed for the database to be created may not be already used for another existing database.

If these conditions pass, the database is created with help of the support functions *_generateUniqueID()* and *_createDB()*. The database will either be appended to the end of *dbs* or will fill an existing empty index in *dbs* that has opened up due to the deletion of a database. The *size* variable is incremented after this addition to *dbs*.

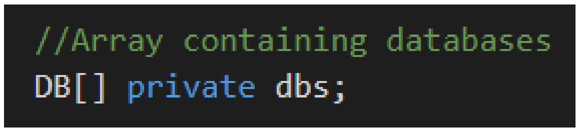


```
//Struct for DB
struct DB {
        //payload
        string data;

        //id
        uint uid;
        string name;
        address owner;

        //timing
        uint lastUpdated;
        uint readyTime;

        //sync mechanisms
        bool lock;
        bool rlock;
        uint numReaders;
    }
```

*Figure 5. Database Struct in ExChain*

*deleteDB()* allows for the head entity, or deployer/owner of the ExChain smart contract, to delete database entries within the *dbs* array. Simliar to *createDB()*, only the head entity may delete databases within this array because of the *onlyOwner* modifier. *deleteDB()* accepts the generated *uid* of the database it intends to delete. The function requires that the head entity may not delete its own database in the case the ownership of the smart contract is transferred throughout its operation and the creation of a database belonging to a past head entity occurs.

In order for the current head entity to delete the database, it must acquire the main database lock to ensure that the current owner is not writing to the database and current readers are not reading the database to prevent race conditions on a potentially sharded Ethereum network. With all conditions met, the entry can then be deleted, the *size* of *dbs* is decremented, and the current owner of the database is set to not own any database.



```
//Array containing databases
DB[] private dbs;
```

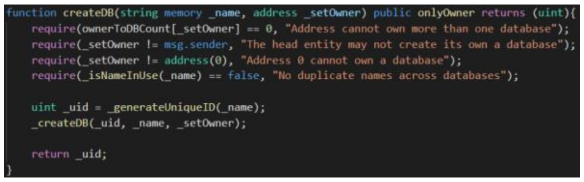
*Figure 6. Array of Databases*

*writeDB()* allows for an owner of a database to write to its own database. Only the caller that owns a database on ExChain may edit its owned database by use of the *ownsDB* modifier on the function. *writeDB()* accepts the string data to be written to the *data* field of its owned database. Before the writing occurs, a check is done to ensure that the cooldown period has ended since the last write to the database or its initial creation. If the database is ready to be written to, the main lock is acquired, the data is written, a new cooldown period is set, and the main lock is released.

*readDB()* allows for an owner of a database or the head entity to read another database within ExChain other than its own database. Only the caller that owns a database or is the head entity may read a database by use of the *participant* modifier on the function. *readDB()* accepts the *uid* of the database to be read. A cooldown period is not implemented for readers as this is a non-destructive operation. However, this should be considered to be implemented in the future given it could lead to a prolonged denial-of-service to the *deleteDB()* function. Synchronization is used to mirror a traditional reader-writer problem on the reader side. Usage of the synchronization design enables a future non-sequential, sharded blockchain.

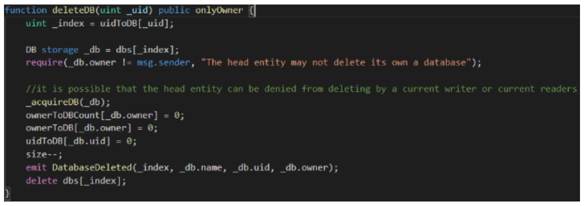
## *D. ExChain Demo*

ExChain.sol was compiled and migrated to the truffle framework. The total migration fee costed approximately 0.053 ETH. Further details can be found in the migrate.txt file in the appendix. Testing was done by interacting with the truffle console and reading the output on stdout. Javascript code found in the exchain.txt file and Table 1 of the appendix tests core aspects of the sample ExChain application.



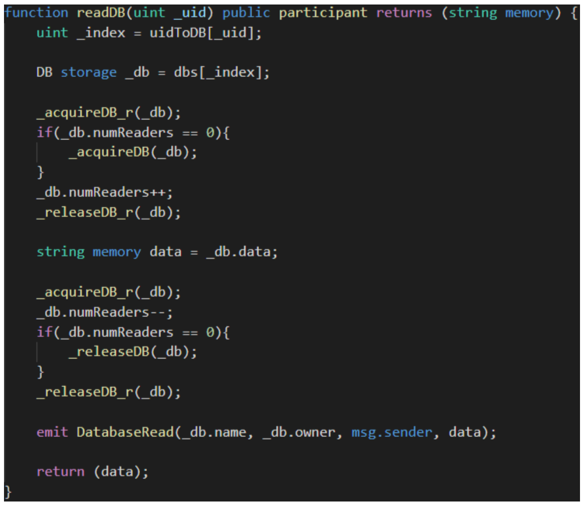
```
function createDB(string memory _name, address _setOwner) public onlyOwner returns (uint){
    require(ownerToDBCount[_setOwner] == 0, "Address cannot own more than one database");
    require(_setOwner != msg.sender, "The head entity may not create its own a database");
    require(_setOwner != address(0), "Address 0 cannot own a database");
    require(_isNameInUse(_name) == false, "No duplicate names across databases");

    uint _uid = _generateUniqueID(_name);
    _createDB(_uid, _name, _setOwner);

    return _uid;
}
```

*Figure 7. createDB Function*



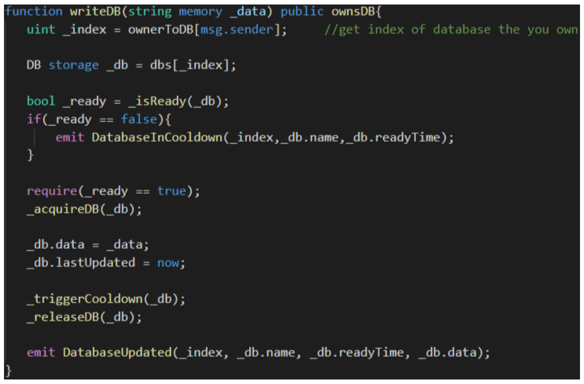
```
function deleteDB(uint _uid) public onlyOwner {
    uint _index = uidToDB[_uid];

    DB storage _db = dbs[_index];
    require(_db.owner != msg.sender, "The head entity may not delete its own a database");

    //it is possible that the head entity can be denied from deleting by a current writer or current readers
    _acquireDB(_db);
    ownerToDBCount[_db.owner] = 0;
    ownerToDB[_db.owner] = 0;
    uidToDB[_db.uid] = 0;
    size--;
    emit DatabaseDeleted(_index, _db.name, _db.uid, _db.owner);
    delete dbs[_index];
}
```

*Figure 8. deleteDB Function*



```
function readDB(uint _uid) public participant returns (string memory) {
    uint _index = uidToDB[_uid];

    DB storage _db = dbs[_index];

    _acquireDB_r(_db);
    if(_db.numReaders == 0){
        _acquireDB(_db);
    }
    _db.numReaders++;
    _releaseDB_r(_db);

    string memory data = _db.data;

    _acquireDB_r(_db);
    _db.numReaders--;
    if(_db.numReaders == 0){
        _releaseDB(_db);
    }
    _releaseDB_r(_db);

    emit DatabaseRead(_db.name, _db.owner, msg.sender, data);

    return (data);
}
```

*Figure 9. readDB Function*



```
function writeDB(string memory _data) public ownsDB{
   uint _index = ownerToDB[msg.sender];     //get index of database the you own

   DB storage _db = dbs[_index];

   bool _ready = _isReady(_db);
   if(_ready == false){
       emit DatabaseInCooldown(_index,_db.name,_db.readyTime);
   }

   require(_ready == true);
   _acquireDB(_db);

   _db.data = _data;
   _db.lastUpdated = now;

   _triggerCooldown(_db);
   _releaseDB(_db);

   emit DatabaseUpdated(_index, _db.name, _db.readyTime, _db.data);
}
```

*Figure 10. writeDB Function*

## IX. Conclusion

The possibilities of ExChain are endless at this point in time. Theoretically, it has the ability to revolutionize record keeping and the exchange of student information through an interconnected blockchain-enabled system. Universities will see a spike in information receipt and verification of data while also being able to trust that the data being transferred is safe and secure.

As ExChain is built first for the higher education system, its abilities span across many different sectors and domains. This system can be applied to the domains of health, government, transportation, and most industrial endeavors. The concept alone can transform society’s way of record keeping, timestamping, and verification of information into a much easier, accessible, efficient, and secure form of storage.

## References

[1] M. Turkanović et al., “EduCTX: A blockchain-based higher education credit platform,” IEEE Access, vol. 6, pp. 5112-5127, 2018.

[2] Arndt, Timothy & Guercio, Angela. (2020). Blockchain-Based Transcripts for Mobile Higher-Education. International Journal of Information and Education Technology. 10. 84-89. 10.18178/ijiet.2020.10.2.1344.

[3] Daraghmi, Eman & Daraghmi, Yousef-Awwad & Yuan, Shyan-Ming. (2019). UniChain: A Design of Blockchain-Based System for Electronic Academic Records Access and Permissions Management. Applied Sciences. 9.10.3390/app9224966.

[4] Xiao, Xingtang & Yu, Zhuo & Xie, Ke & Guo, Shaoyong & Xiong, Ao & Yan, Yong. (2020). A Multi-blockchain Architecture Supporting Cross-Blockchain Communication. 10.1007/978-981-15-8086-4_56.

[5] Liu, K., & Ohsawa, Y. (2020). Improving Blockchain scalability based on one-time cross-chain contract and gossip network. *arXiv preprint arXiv:2008.04601*.

[6] Dasgupta, D., Shrein, J. M., & Gupta, K. D. (2019). A survey of blockchain from security perspective. *Journal of Banking and Financial Technology*, 3(1), 1-17.

[7] Qasse, Ilham & Abu Talib, Manar & Nasir, Q.. (2019). Inter Blockchain Communication: A Survey. 1-6. 10.1145/3333165.3333167.

[8] L. Kan, Y. Wei, A. Hafiz Muhammad, W. Siyuan, L. C. Gao and H. Kai, "A Multiple Blockchains Architecture on Inter-Blockchain Communication," 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2018, pp. 139-145, doi: 10.1109/QRS-C.2018.00037.

[9] Fedorova, Elena & Skobleva, Ella. (2020). Application of Blockchain Technology in Higher Education. European Journal of Contemporary Education. 9. 10.13187/ejced.2020.3.552.

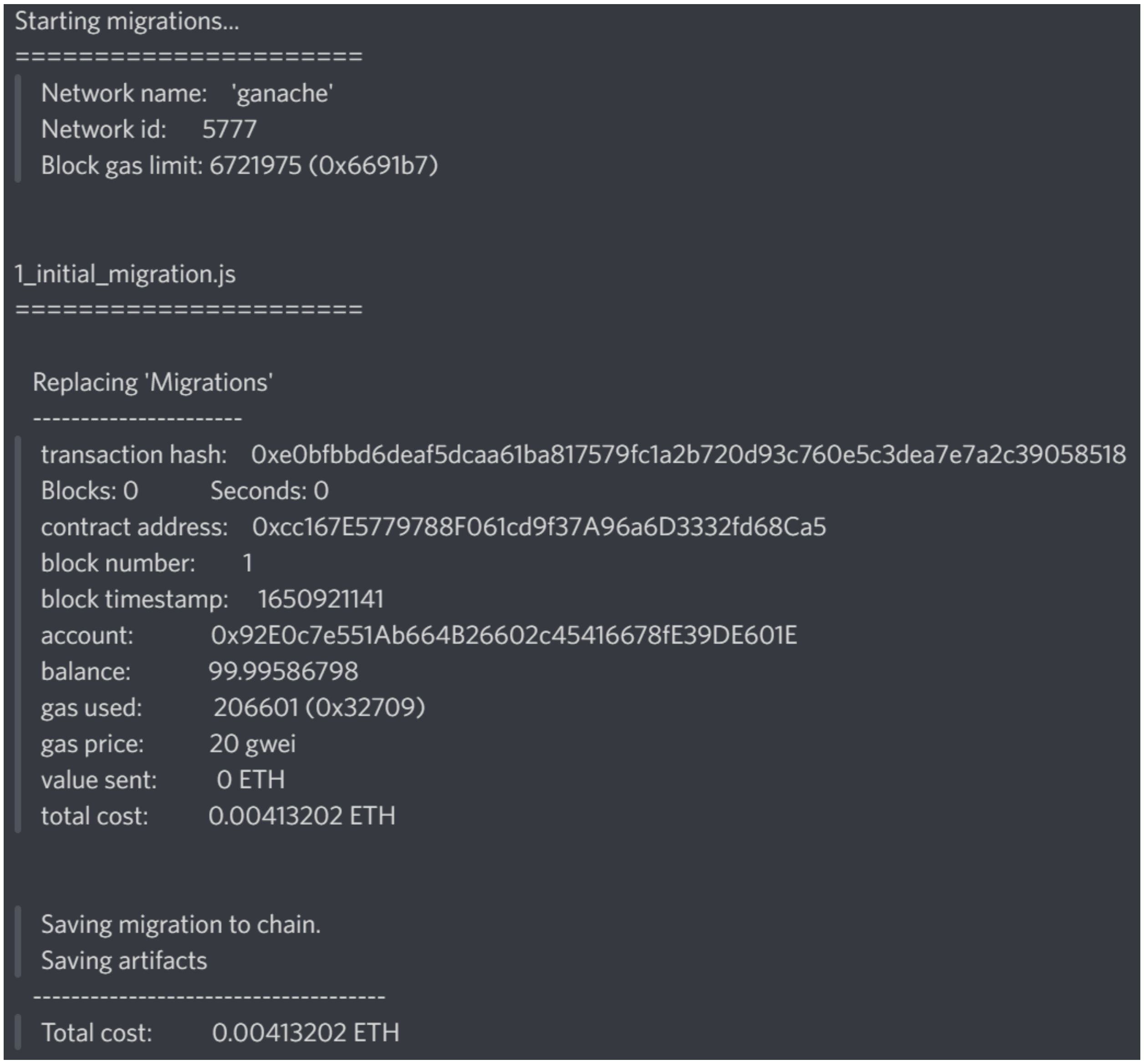
[10] Lam, Tsz & Dongol, Brijesh. (2020). A blockchain-enabled e-learning platform. Interactive Learning Environments. 1-23. 10.1080/10494820.2020.1716022.

[11] Bucea-Manea-Tonis, Rocsana & Martins, Oliva & Bucea-Manea-Tonis, Radu & Oniş, & Gheorghit, Cătălin & Kuleto, Valentin & Ilić, Milena & Simion, Violeta. (2021). Blockchain Technology Enhances Sustainable Higher Education. Sustainability. 13. 10.3390/su132212347.

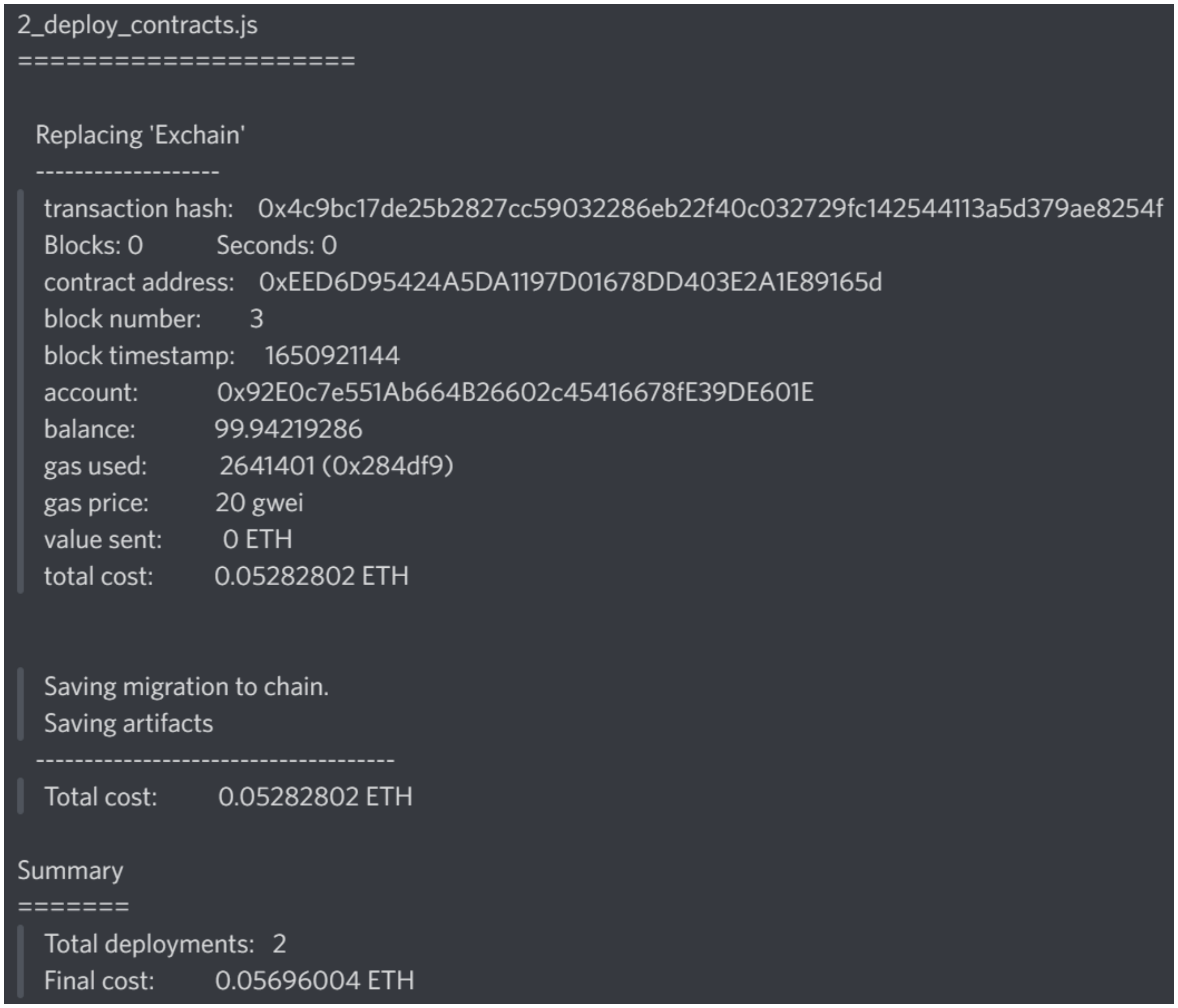
[12] Nakamoto, Satoshi. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System. Cryptography Mailing list at https://metzdowd.com.

# APPENDIX

Screenshots of the *migrate.txt* file



```
Starting migrations...
======================
> Network name:    'ganache'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Replacing 'Migrations'
   ----------------------
   > transaction hash:    0xe0bfbbd6deaf5dcaa61ba817579fc1a2b720d93c760e5c3dea7e7a2c39058518
   > Blocks: 0            Seconds: 0
   > contract address:    0xcc167E5779788F061cd9f37A96a6D3332fd68Ca5
   > block number:        1
   > block timestamp:     1650921141
   > account:             0x92E0c7e551Ab664B26602c45416678fE39DE601E
   > balance:             99.99586798
   > gas used:            206601 (0x32709)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00413202 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00413202 ETH
```



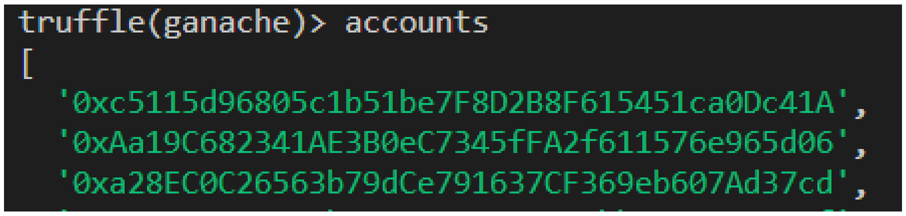
```
2_deploy_contracts.js
=====================

   Replacing 'Exchain'
   -------------------
   > transaction hash:    0x4c9bc17de25b2827cc59032286eb22f40c032729fc142544113a5d379ae8254f
   > Blocks: 0            Seconds: 0
   > contract address:    0xEED6D95424A5DA1197D01678DD403E2A1E89165d
   > block number:        3
   > block timestamp:     1650921144
   > account:             0x92E0c7e551Ab664B26602c45416678fE39DE601E
   > balance:             99.94219286
   > gas used:            2641401 (0x284df9)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.05282802 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.05282802 ETH

Summary
=======
> Total deployments:   2
> Final cost:          0.05696004 ETH
```

Table 1. Sample ExChain Test

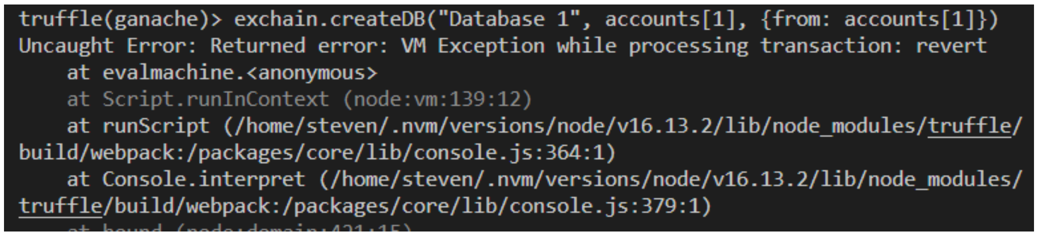
| **Line** | **Code** | **Explanation** |
|---|---|---|
| **1.** | truffle console >\| console.txt | Start the console and redirect output to console.txt |
| **2.** | let exchain = await Exchain.deployed() | Let the instance of the contract be assigned to "exchain" |
| **3.** | let accounts = await web3.eth.getAccounts() | Let the accounts available by ganache be assigned to "accounts" |
| **4.** | exchain.createDB("Database 1", accounts[1], {from: accounts[1]}) | Will not execute because cerateDB can only be called by head entity, accounts[0] |
| **5.** | exchain.createDB("Database 1", accounts[1], {from: accounts[0]})<br><br>exchain.getPastEvents("NewDatabaseCreated")<br><br>const uid_1 = web3.utils.toBN('191622698137759822581840134640635579761930185278402190781308876202126379079 1'); | "Database 1" is added to dbs<br><br>Event displays uid assigned to the database. The uid is saved as a big number constant |
| **6.** | exchain.readDB(uid_1, {from: accounts[2]}) | Accounts[2] cannot read Database 1 because it is not a participant in Exchain |
| **7.** | exchain.readDB(uid_1, {from: accounts[1]})<br><br>exchain.getPastEvents("DatabaseRead") | Accounts[1] reads its own database, which data field says is "EMPTY" |
| **8.** | let exampleDBData = "Grades-\n\nAlice: 93\nBob: 87\nCharlie: 79"<br><br>exchain.writeDB(exampleDBData, {from: accounts[1]})<br><br>exchain.getPastEvents("DatabaseUpdated") | Example data is written to database 1 by accounts[1] |

| | | |
|---|---|---|
| **9.** | exchain.createDB("Database 2", accounts[2], {from: accounts[0]})<br><br>exchain.getPastEvents("NewDatabaseCreated")<br><br>const uid_2 = web3.utils.toBN('49844171400339760096023001691183441541866923126243543143574375593897598742208'); | "Database 2" is added to dbs<br><br>Event displays uid assigned to the database. The uid is saved as a big number constant |
| **10.** | exchain.readDB(uid_1, {from: accounts[2]})<br><br>exchain.getPastEvents("DatabaseRead") | Accounts[2] can now read "Database 1" using uid_1 because it is a participant in the Exchain network |
| **11.** | exchain.deleteDB(uid_2, {from: accounts[0]})<br><br>exchain.getPastEvents("DatabaseDeleted") | The head entity deletes "Database 2" |
| **12.** | exchain.readDB(uid_1, {from: accounts[2]}) | Accounts[2] cannot read from "Database 1" because it is no longer a participant in Exchain |

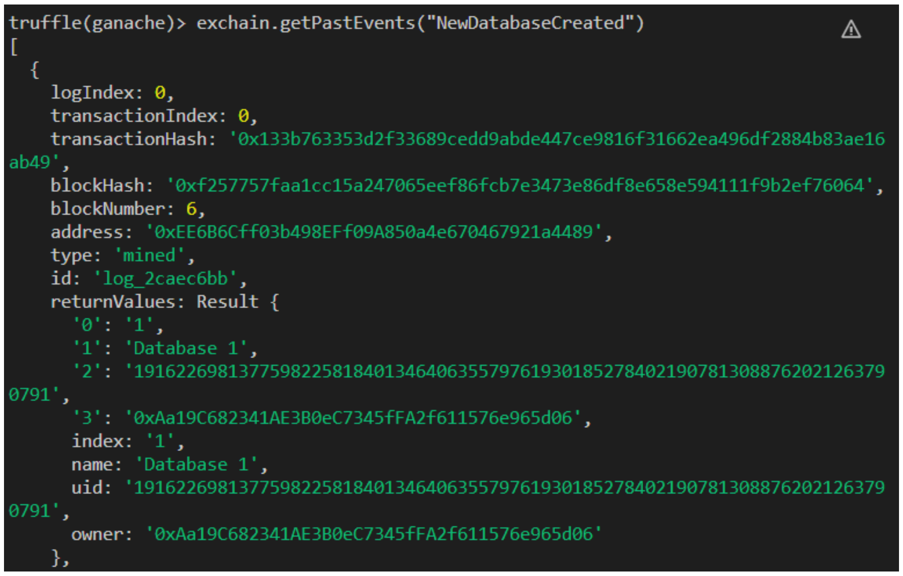
Sample Outputs in the Truffle Console in order of Table 1. :



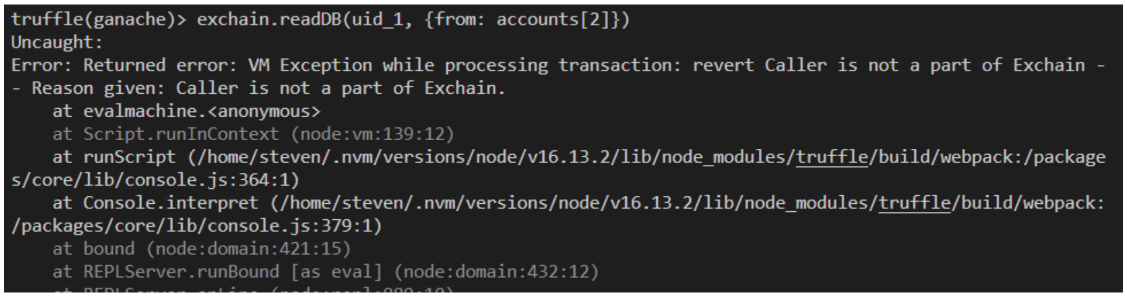
```
truffle(ganache)> accounts
[
  '0xc5115d96805c1b51be7F8D2B8F615451ca0Dc41A',
  '0xAa19C682341AE3B0eC7345fFA2f611576e965d06',
  '0xa28EC0C26563b79dCe791637CF369eb607Ad37cd',
```



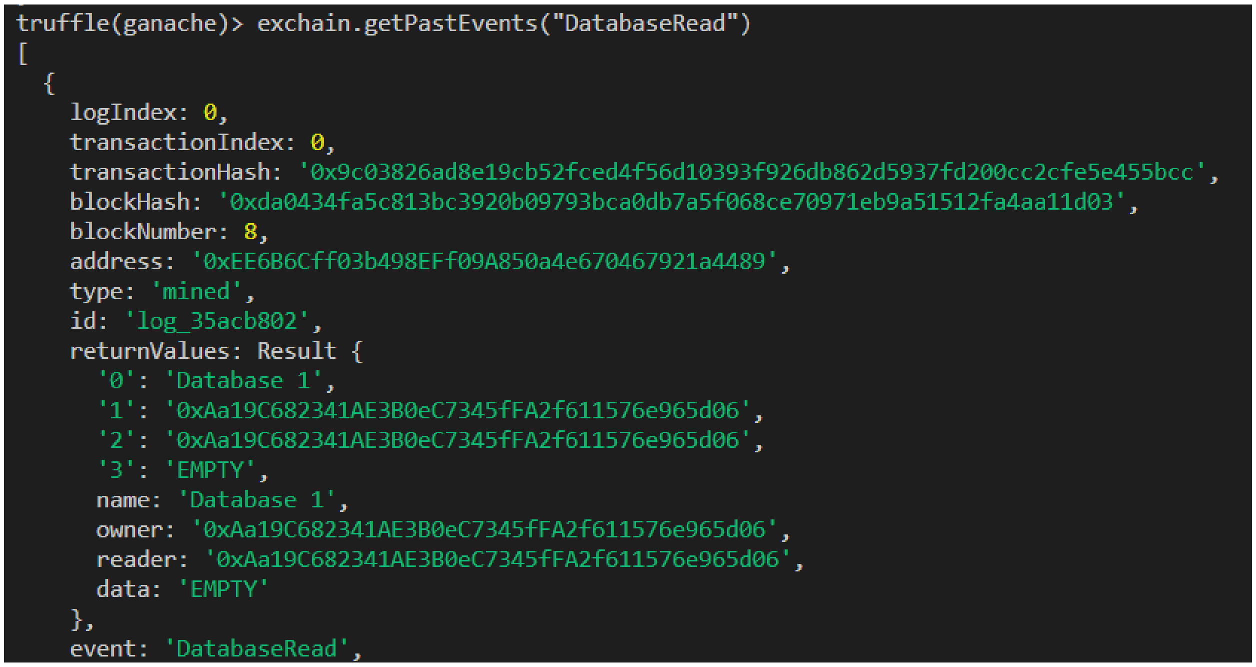
```
truffle(ganache)> exchain.createDB("Database 1", accounts[1], {from: accounts[1]})
Uncaught Error: Returned error: VM Exception while processing transaction: revert
    at evalmachine.<anonymous>
    at Script.runInContext (node:vm:139:12)
    at runScript (/home/steven/.nvm/versions/node/v16.13.2/lib/node_modules/truffle/
build/webpack:/packages/core/lib/console.js:364:1)
    at Console.interpret (/home/steven/.nvm/versions/node/v16.13.2/lib/node_modules/
truffle/build/webpack:/packages/core/lib/console.js:379:1)
```



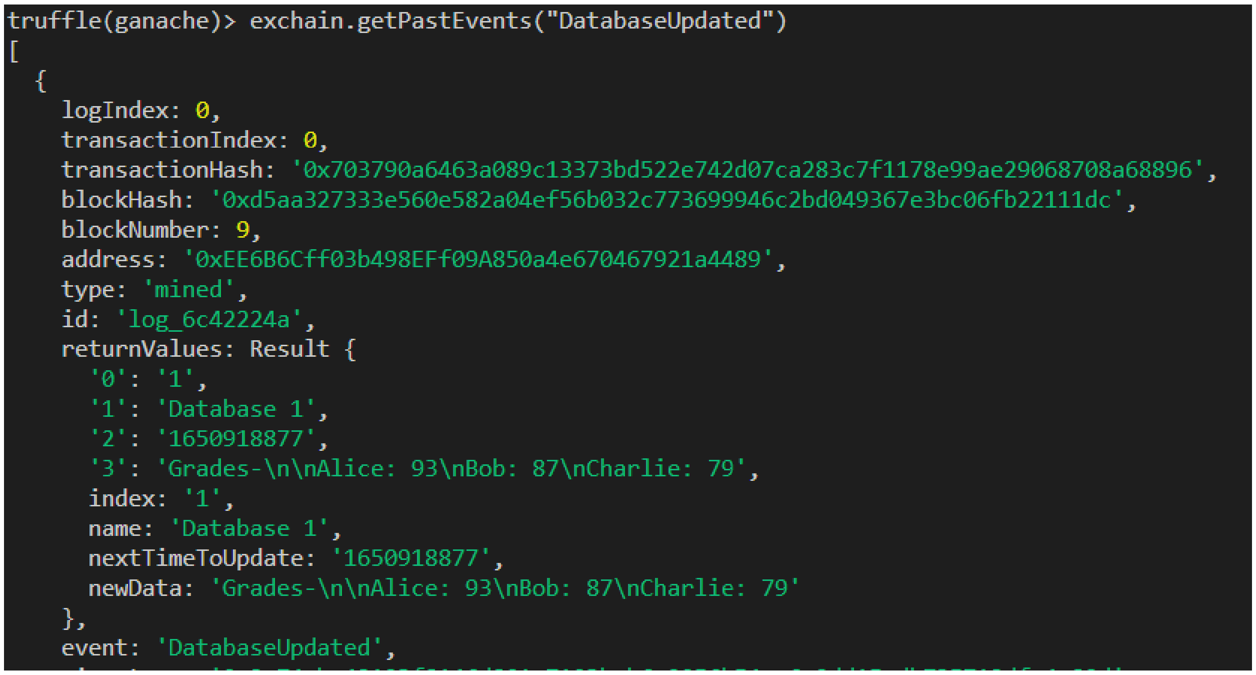
```
truffle(ganache)> exchain.getPastEvents("NewDatabaseCreated")
[
  {
    logIndex: 0,
    transactionIndex: 0,
    transactionHash: '0x133b763353d2f33689cedd9abde447ce9816f31662ea496df2884b83ae16
ab49',
    blockHash: '0xf257757faa1cc15a247065eef86fcb7e3473e86df8e658e594111f9b2ef76064',
    blockNumber: 6,
    address: '0xEE6B6Cff03b498EFf09A850a4e670467921a4489',
    type: 'mined',
    id: 'log_2caec6bb',
    returnValues: Result {
      '0': '1',
      '1': 'Database 1',
      '2': '191622698137759822581840134640635579761930185278402190781308876202126379
0791',
      '3': '0xAa19C682341AE3B0eC7345fFA2f611576e965d06',
      index: '1',
      name: 'Database 1',
      uid: '191622698137759822581840134640635579761930185278402190781308876202126379
0791',
      owner: '0xAa19C682341AE3B0eC7345fFA2f611576e965d06'
    },
```



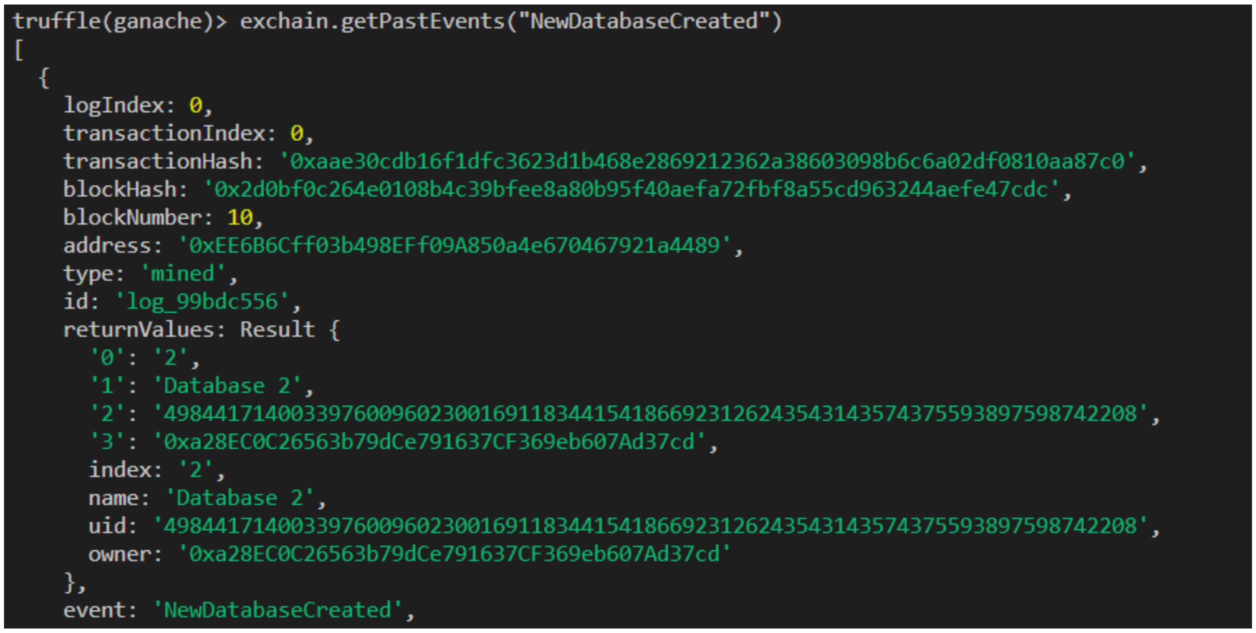
```
truffle(ganache)> exchain.readDB(uid_1, {from: accounts[2]})
Uncaught:
Error: Returned error: VM Exception while processing transaction: revert Caller is not a part of Exchain -
- Reason given: Caller is not a part of Exchain.
    at evalmachine.<anonymous>
    at Script.runInContext (node:vm:139:12)
    at runScript (/home/steven/.nvm/versions/node/v16.13.2/lib/node_modules/truffle/build/webpack:/package
s/core/lib/console.js:364:1)
    at Console.interpret (/home/steven/.nvm/versions/node/v16.13.2/lib/node_modules/truffle/build/webpack:
/packages/core/lib/console.js:379:1)
    at bound (node:domain:421:15)
    at REPLServer.runBound [as eval] (node:domain:432:12)
```



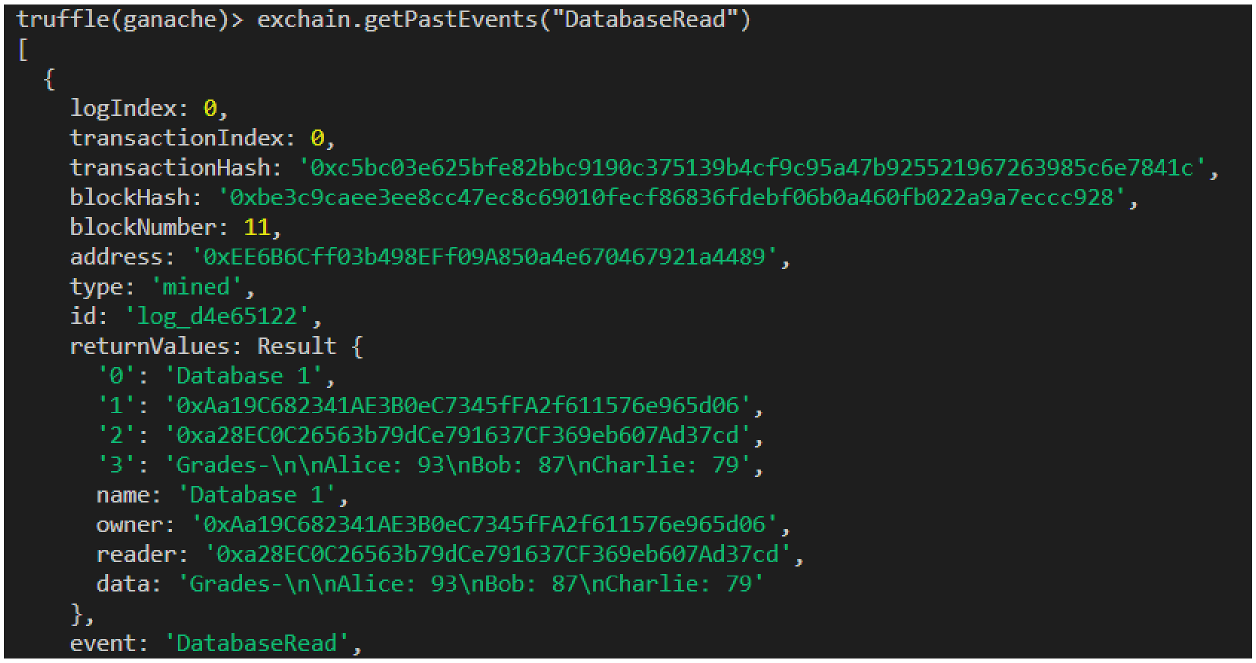
```
truffle(ganache)> exchain.getPastEvents("DatabaseRead")
[
  {
    logIndex: 0,
    transactionIndex: 0,
    transactionHash: '0x9c03826ad8e19cb52fced4f56d10393f926db862d5937fd200cc2cfe5e455bcc',
    blockHash: '0xda0434fa5c813bc3920b09793bca0db7a5f068ce70971eb9a51512fa4aa11d03',
    blockNumber: 8,
    address: '0xEE6B6Cff03b498EFf09A850a4e670467921a4489',
    type: 'mined',
    id: 'log_35acb802',
    returnValues: Result {
      '0': 'Database 1',
      '1': '0xAa19C682341AE3B0eC7345fFA2f611576e965d06',
      '2': '0xAa19C682341AE3B0eC7345fFA2f611576e965d06',
      '3': 'EMPTY',
      name: 'Database 1',
      owner: '0xAa19C682341AE3B0eC7345fFA2f611576e965d06',
      reader: '0xAa19C682341AE3B0eC7345fFA2f611576e965d06',
      data: 'EMPTY'
    },
    event: 'DatabaseRead',
```



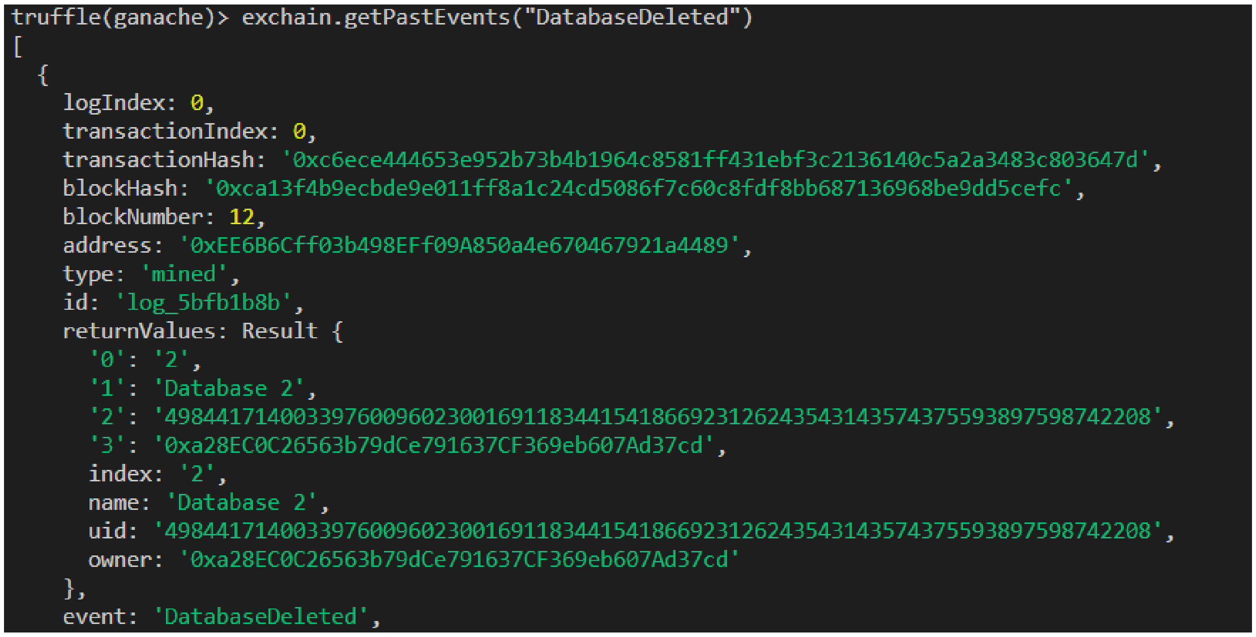
```
truffle(ganache)> exchain.getPastEvents("DatabaseUpdated")
[
  {
    logIndex: 0,
    transactionIndex: 0,
    transactionHash: '0x703790a6463a089c13373bd522e742d07ca283c7f1178e99ae29068708a68896',
    blockHash: '0xd5aa327333e560e582a04ef56b032c773699946c2bd049367e3bc06fb22111dc',
    blockNumber: 9,
    address: '0xEE6B6Cff03b498EFf09A850a4e670467921a4489',
    type: 'mined',
    id: 'log_6c42224a',
    returnValues: Result {
      '0': '1',
      '1': 'Database 1',
      '2': '1650918877',
      '3': 'Grades-\n\nAlice: 93\nBob: 87\nCharlie: 79',
      index: '1',
      name: 'Database 1',
      nextTimeToUpdate: '1650918877',
      newData: 'Grades-\n\nAlice: 93\nBob: 87\nCharlie: 79'
    },
    event: 'DatabaseUpdated',
```



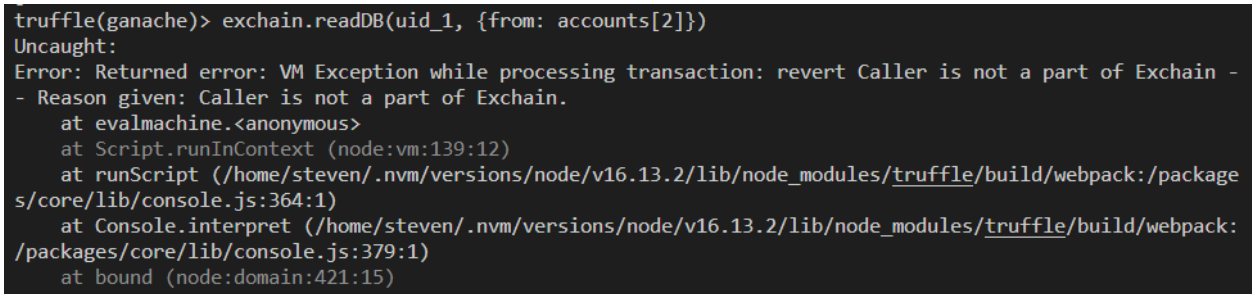
```
truffle(ganache)> exchain.getPastEvents("NewDatabaseCreated")
[
  {
    logIndex: 0,
    transactionIndex: 0,
    transactionHash: '0xaae30cdb16f1dfc3623d1b468e2869212362a38603098b6c6a02df0810aa87c0',
    blockHash: '0x2d0bf0c264e0108b4c39bfee8a80b95f40aefa72fbf8a55cd963244aefe47cdc',
    blockNumber: 10,
    address: '0xEE6B6Cff03b498EFf09A850a4e670467921a4489',
    type: 'mined',
    id: 'log_99bdc556',
    returnValues: Result {
      '0': '2',
      '1': 'Database 2',
      '2': '49844171400339760096023001691183441541866923126243543143574375593897598742208',
      '3': '0xa28EC0C26563b79dCe791637CF369eb607Ad37cd',
      index: '2',
      name: 'Database 2',
      uid: '49844171400339760096023001691183441541866923126243543143574375593897598742208',
      owner: '0xa28EC0C26563b79dCe791637CF369eb607Ad37cd'
    },
    event: 'NewDatabaseCreated',
```



```
truffle(ganache)> exchain.getPastEvents("DatabaseRead")
[
  {
    logIndex: 0,
    transactionIndex: 0,
    transactionHash: '0xc5bc03e625bfe82bbc9190c375139b4cf9c95a47b925521967263985c6e7841c',
    blockHash: '0xbe3c9caee3ee8cc47ec8c69010fecf86836fdebf06b0a460fb022a9a7eccc928',
    blockNumber: 11,
    address: '0xEE6B6Cff03b498EFf09A850a4e670467921a4489',
    type: 'mined',
    id: 'log_d4e65122',
    returnValues: Result {
      '0': 'Database 1',
      '1': '0xAa19C682341AE3B0eC7345fFA2f611576e965d06',
      '2': '0xa28EC0C26563b79dCe791637CF369eb607Ad37cd',
      '3': 'Grades-\n\nAlice: 93\nBob: 87\nCharlie: 79',
      name: 'Database 1',
      owner: '0xAa19C682341AE3B0eC7345fFA2f611576e965d06',
      reader: '0xa28EC0C26563b79dCe791637CF369eb607Ad37cd',
      data: 'Grades-\n\nAlice: 93\nBob: 87\nCharlie: 79'
    },
    event: 'DatabaseRead',
```



```
truffle(ganache)> exchain.getPastEvents("DatabaseDeleted")
[
  {
    logIndex: 0,
    transactionIndex: 0,
    transactionHash: '0xc6ece444653e952b73b4b1964c8581ff431ebf3c2136140c5a2a3483c803647d',
    blockHash: '0xca13f4b9ecbde9e011ff8a1c24cd5086f7c60c8fdf8bb687136968be9dd5cefc',
    blockNumber: 12,
    address: '0xEE6B6Cff03b498EFf09A850a4e670467921a4489',
    type: 'mined',
    id: 'log_5bfb1b8b',
    returnValues: Result {
      '0': '2',
      '1': 'Database 2',
      '2': '49844171400339760096023001691183441541866923126243543143574375593897598742208',
      '3': '0xa28EC0C26563b79dCe791637CF369eb607Ad37cd',
      index: '2',
      name: 'Database 2',
      uid: '49844171400339760096023001691183441541866923126243543143574375593897598742208',
      owner: '0xa28EC0C26563b79dCe791637CF369eb607Ad37cd'
    },
    event: 'DatabaseDeleted',
```



```
truffle(ganache)> exchain.readDB(uid_1, {from: accounts[2]})
Uncaught:
Error: Returned error: VM Exception while processing transaction: revert Caller is not a part of Exchain -
- Reason given: Caller is not a part of Exchain.
    at evalmachine.<anonymous>
    at Script.runInContext (node:vm:139:12)
    at runScript (/home/steven/.nvm/versions/node/v16.13.2/lib/node_modules/truffle/build/webpack:/package
s/core/lib/console.js:364:1)
    at Console.interpret (/home/steven/.nvm/versions/node/v16.13.2/lib/node_modules/truffle/build/webpack:
/packages/core/lib/console.js:379:1)
    at bound (node:domain:421:15)
```