DS Journal of Cyber Security
Volume 3 Issue 3, 1-14, Jul - Sep 2025
ISSN: 2584-0665 / https://doi.org/10.59232/CYS-V3I3P101

Original Article

Integrating Smart Contracts into University Registration for Data Integrity and Transparency

Thuc Le[1], Thai Le[2,3], Hai Tran[4*]

[1]*Graduate Student in Information Technology, Ho Chi Minh City University of Education, Vietnam.*
[2]*Faculty of Information Technology, University of Science, Ho Chi Minh City, Vietnam.*
[3]*Vietnam National University, Ho Chi Minh City, Vietnam.*
[4]*Faculty of Information Technology, Ho Chi Minh University of Education, Ho Chi Minh City, Vietnam.*

*haits@hcmue.edu.vn

Received: 07 June 2025; Revised: 08 July 2025; Accepted: 01 August 2025; Published: 15 August 2025

Abstract - A university registration system needs transparency, high security, and high integrity of the data to prevent fraudulent actions on registration. Blockchain technology is a technology focused on the secure, transparent, and integrity of the data. This technology is suitable for the requirements of a registration system. This study proposes a method for integrating the blockchain into the registration system by using the smart contract to design the storage of data on the blockchain, and the functions with the logic process for the actions related to registration. This paper will discuss the architecture design, implementation, and testing to show the potential of applied blockchain in the registration system.

Keywords - Blockchain, Data integrity, Data transparency, Smart contract, University registration system.

1. Introduction

Nowadays, every university has a registration system [1] for managing the information of classes. Almost all these were built on the web-based model [2] with the server-client architecture [3]. One of the limitations of the traditional server-client model is a lack of transparency and a lack of security if not carefully designed [4]. In the context of the registration system, the lack of transparency can lead to abusive actions like intervening directly in the database, and a lack of security can lead to abusive actions like hacking or taking advantage of bugs in the registration system. This makes for an unfair registration and should not be in a registration system. So, transparency and data integrity in the registration system need to be enhanced.

Blockchain is a technology that focuses on protecting the integrity of data by using a decentralized network and an immutable ledger. Blockchain provides the smart contract feature for customizing the application to store the data on the blockchain. Integrating these technologies into data systems can help enhance transparency and integrity. Blockchain with smart contracts has been studied to bring many benefits, such as enhancing security and transparency and reducing fraud for registration [5, 6]. However, the above research only points out the advantages and the model to use without giving the implementation. The data on blockchain is public for anyone to see, so the design of the registration system using the smart contract must be careful to preserve privacy.

Furthermore, the data stored on blockchain is not completely the same as that of a traditional database, so the design implementation also has many differences.

As stated above, the registration system should enhance the transparency and integrity of the data, which is the motivation for this paper. Blockchain with the smart contract is the key to solving. So, this study explores the way blockchain can be integrated into the registration system. Propose a solution to integrate a smart contract into the registration system and demonstrate the feasibility of the proposed solution.

2. Background and Related Work

2.1. Background

Recently, blockchain technology [7] has emerged as a technology with superior security, data integrity, and transparency. Blockchain uses a peer-to-peer network [8] architecture without a centralized server in the server-client model. All nodes in the blockchain are the same, which means no node can unilaterally intend to change the data. It keeps the integrity of the data. The data in blockchain is mutable, and anyone can see all transaction history, so blockchain is highly transparent.

All execution in the blockchain network must be executed and verified by many nodes, which makes the executed result trusted, and hence, the blockchain is highly secure. Blockchain is a really promising technology and has been researched to be applied in various fields like finance [9, 10], healthcare [11, 12], supply chain management [13, 14], and education [15-17].

The smart contract [18] is a feature in the blockchain network that allows developers to build applications that utilize the computation of blockchain to store data on the blockchain. Because an application built on blockchain has a decentralized property, it is called a Decentralized Application (dApp). This is a key feature for integrating blockchain into the application. Note that not every blockchain can support the smart contract feature. Only blockchains that support the smart contract feature can deploy a DApp, and each blockchain has many different ways to support a DApp. A DApp successfully deployed on this blockchain can not be sure to deploy successfully on another blockchain.

Smart contracts include a design for data storage (variables and types of data) and logic for processing the data. Ethereum [19] is a famous blockchain that supports smart contracts and is used by a large number of developers. Ethereum supports the smart contract written in the Solidity language [20].

The Ethereum blockchain has a core component called the Ethereum Virtual Machine (EVM) [21] for executing the smart contract. Before the EVM executes, smart contracts are compiled into EVM bytecode. When the function in the smart contract is triggered, nothing can affect the processing, and the predefined logic will be executed, which ensures the high integrity of data when developing a DApp.

2.2. Related Work

As a potential technology, applying blockchain in the university's systems brings many effective [22]. In university, blockchain can enhance academic credential management, incentivize student motivation through rewards based on complete achievements [23]. The key to integrating blockchain is the smart contract. Using the hashing technique and using a smart contract to store the hash of academic certificates [24-26], diplomas [27], and verifying by comparing the hash.

For scholarship review, a trusted system is needed to retrieve and verify data of students, which can use smart contracts to build a system with high trust and high transparency [28, 29]. Using encrypted tokens from tokenization enhances the security for payments in the university. The token and actions for payment were managed by logic implementations on the smart contract [30]. Some subjects in university exams can be performed on the web system, and smart contracts can store students' exam results, increasing transparency [31, 32].

3. Proposed Solution

3.1. Definition of Objects in the Registration System

In each semester at university, students can consider selecting the subjects for study in that semester. A student can enroll in a subject if they can meet all the requirements of the subject. If the subject has requirements for the required subject Student must completed all the required subject before enrol, e.g. student want to enroll subject "Advanced Data Structure and Algorithm" must completed the required subject "Basic Data Structure and Algorithm" and "Basic Programming" at the time want to enroll the "Advanced Data Structure and Algorithm" subject.

Each subject has many classes in a semester. The difference between the classes is the time, the location, or the lecturer. Each university may have some different conditions for enrolling in the class, but in general, each class has some basic conditions for registration:

- A class has a certain number of slots, and a student only enrols in a class if it has at least one slot without being enrolled by any student (has an empty slot).
- The class has a deadline for registration. The student can only enrol or unregister from the class before the deadline.
- Because the class has a limit on the number of slots, the university can prioritize the student completion of the class on time in the expected schedule of the education program. This means, in the time for priority enrollment, only students on the priority list can enroll. After this time, all students without priority can enroll.

Students can enroll in the class using a class registration system. A class registration system should have basic functions:

- For managers: Manage the information on registration of students and the status of the subject student studying (completed or not passed).
- For students: Can track their studying status and see the appropriate class when registering. In the process of enrollment, the system must have correct logic.

3.2. System Overview

The smart contracts are used to organize the data and functions to manage data on the blockchain. For convenience, the word contract will be implied as a smart contract unless otherwise noted in the remainder of this paper.

The role system for managing permissions will be designed in the smart contract. The authority account of the system is controlled by the Externally Owned Account (EOA) [33] authority of the blockchain. An EOA must be assigned a role to interact with the system. The design will use three smart contracts for managing data: Role contract for managing the Role of the address, Class contract for managing the information of courses, and Student contract for managing students' registration information.

3.3. Roles and Permissions

There are three roles in the proposed method: Admin, Manager, and Student. Each Role has specific permissions in the system. Each account on the blockchain will be granted a role to interact with the system. An account without a role can not perform any actions on the system.

3.3.1. Student

One student has a unique account on the system, unique by the address on the blockchain. Each student's account will be granted the Student role. Only the account has the Role of a Student who can enroll in the class. When the student enrolls in a class, the system will check all requirements of the class and return the result for enrollment. Only the student can be granted the Student role, and the Role is managed by the academic affairs officer. An academic affair can grant and revoke the Role of a Student anytime.

3.3.2. Manager

An academic affairs officer will be granted the Role of Manager. The Manager's Role is to manage the students and the class.

To Manage the Student: First, an academic affairs officer can grant and revoke the student's Role (in case of violation, like indiscipline, being forced to leave school, etc.). This action can be done at any time. Next, academic affairs has permission to update the studying status of the subject when a student completes a subject.

For Managing the Class: The account has a Manager role that can create and modify the class. The class was created with an initial disabled status. The student can not enroll in a class that is disabled. The purpose of the disabled status is for the Manager to double-check after the class is created. After that, the Manager can activate the class, and students can register for the class. The action to disable and activate the class can be done anytime. In each class, the Manager can remove a student from the enrollment list for a special reason.

The manager role can only be set by an account that has the Admin role and can have many Manager accounts.

3.3.3. Admin

The admin role was designed to manage the manager role. Admin has permission to grant or revoke the Manager role. Note this: It looks like the Admin has a higher privilege than the Manager, but the Admin can only manage the Manager role and can not manage the Student role.

The system has only one Admin account. The account that deploys the smart contract will be set as Admin in the initialization. Admin can transfer to another account. The Admin is normally held by the information technology department of an educational institution.

3.4. Smart Contract

The system's role mechanism and actions will be implemented on a smart contract. The Solidity language will be used to implement the smart contract. Below are the key features of Solidity for implementation.

3.4.1. Variable

The variables power the ability to custom store the information on the blockchain. The address type on the smart contract represents a user on the system. Address and map data structure with key is the address, which is very effective for storing the information of roles and information of each class, each student, and each mapping (address => data). For managing roles, use mapping (address => bool) roleName, and mark if the address was granted the roleName as True. When needing to check if an address has been granted a role, only check if the value of the address on mapping is True or False: if True means the address has been granted a role, and vice versa.

Similarly, for the student information and class information, using a mapping (address => []data_type), where data_type is the list of appropriate data, the data type is based on the information wanted to store, e.g. use a list of strings to store the subject ID, use a list of integers to store the timestamp. In addition to using a map, some data will be stored in a primitive type, e.g. Admin role is only one, just use a simple variable address admin_address to store the address of the Admin, when checking if the address is Admin, just check if the address is equal to the admin_address variable.

3.4.2. Modifier

This is a function that can be applied to another function. The modifier will be executed before the function is executed. Modifiers will be used to control access and pre-check conditions. For access control, modifiers will used to implement check the Role of address when the address calls a function, e.g. modifier onlyAdmin to check whether an address is Admin and apply it for the functions where only Admin can call (like set Manager, transfer admin), if a call from an address is not Admin, modifier will return fail and can not execute the admin functions. In the system, some conditions need to be checked in multiple functions, e.g. check if the class ID exists when creating a new class or enrolling in a class, just implement a modifier isExistClassId and call isExistClassId where class ID checking is required. Based on the return of the check from isExistClassId, the code after the modifier will be executed or not.

3.4.3. Function

All processing logic will be implemented in the functions. The functions are the place to implement the restriction for the action stated in Section 3.1. Function, combined with the modifiers, will check the required restrictions. If the checking conditions are met, the function will retrieve and return the data or modify and store new data on the blockchain. If not, all states of data on the blockchain will be reverted to the state before the function was called.

3.5. Potential of Applying Blockchain to the Registration System

In the traditional system, an admin or Manager may have permission to modify the data, which can lead to abusive actions that influence enrollment results. All modified data history can only be viewed if database access is available. Additionally, if the log of history does not log who performs the modification, no way to know who abuses the power. All actions will be recorded when integrating the blockchain into the system to see who performed what action. An admin or Manager can not directly modify the data. The modification will be performed through the function with permission, and the data will only be modified following the predefined logic. The enrollment is only performed by the student.

On the blockchain, anyone can see the function call history, which will log the address that called the function. It makes the system transparent, so anyone can self-validate the integrity of the data. This also helps to detect bugs in logic and provides timely notification to the Admin to pause and update the smart contract. Notice that, by technical support, a smart contract may pause or update, and a question has been asked whether the Admin can modify the logic and perform abusive actions on data. Remember, in the proposed design, anyone will be able to check the address of all roles in the system, and on the blockchain, all actions will be logged. This means the update smart contract action will be logged, and anyone can know that the contract has been updated by the Admin. Based on this, even an admin can not abuse the actions.

4. Implementation and Testing

4.1. Smart Contracts Implementation Details

4.1.1. Role Contract

The information address's Role will be stored in the Role contract, and the functions for managing the Role, as defined (check admin/manager/student, change Admin, add/remove Manager, add/remove student as stated in Section 3.1), will be implemented in this contract. The Role contract will be deployed first, and after that, the Class contract and Student contract will be deployed with the initialize parameter. The variable Role roleContract is the address of the Role deployed contract through the constructor. In the Class contract and Student contract, when needing to check the Role of address, the contract will just call roleContract.checkAdmin(msg.sender), roleContract.checkManager(msg.sender), roleContract.checkStudent(msg.sender) (msg.sender is the special variable Solidity supports to get the address of the function caller). This is to check the Admin, Manager, and student, respectively.

Table 1. ClassInformation struct

Field	Type	Description
classId	string	A unique string to identify class, e.g. dsa_24_summer.
subjectCode	string	Code of subject, e.g DSA.
isActive	bool	The status of class, False if the class is not available for registration, and vice versa.
totalSlot	integer	The maximum students can enroll in a class.
enrolled	address[]	List of students who have been enrolled in the class.
totalEnrolled	integer	The slot number has been enrolled.
listRequiredSubjectCode	string[]	The list of subject codes of required subjects must be completed before enrolling on a subject.
priorityList	address[]	List priority students who can enroll on a subject.
PriorityDeadline	integer	Timestamp for the deadline of priority enrollment. If the class has no priority enrollment, then set priorityDeadline = 0.
RegistrationCloseTime	integer	Timestamp for the deadline of registration class.

4.1.2. Class Contract

The implementation of this contract involves storing the class information and the logic process of registration. Custom struct ClassInformation is defined as being used to manage the information of a class. Table 1 shows the details of the essential variable ClassInformation.

All actions related to the creation/modification of classes will be implemented with only the Manager modifier for those who have the manager role and can perform them. In the Class contract, notice that:

- In a Solidity smart contract, when any revert is triggered, the state (data of variables in the function) before the function is called will be reverted.
- The current timestamp is based on the time of the block containing the transaction, which was produced on the blockchain.

Table 2. StudentInformation struct

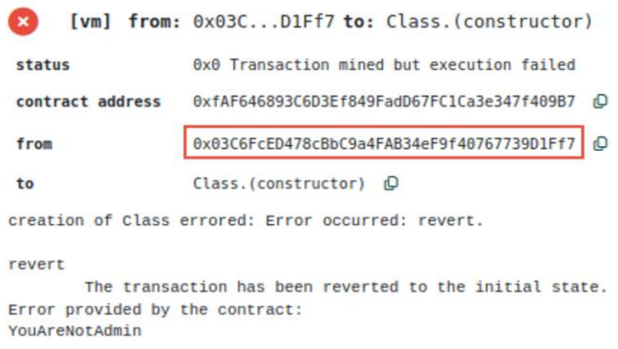
Field	Type	Description
Completed Class	string[]	The list of class ID student has been completed.
Completed Subject	string[]	The list of subject codes the student has completed.
Registered Class	string[]	The list of class ID student has been registered (it can be completed or not).
Registered Subject	string[]	The list of subject codes the student has been registered for (it can be completed or not).

Table 3. List of addresses used in testing

Address	Role
0x5B3...eddC4	Admin
0xAb8...35cb2	Manager
0x4B2...C02db	Student
0x787...cabaB	Student

4.1.3. Student Contract

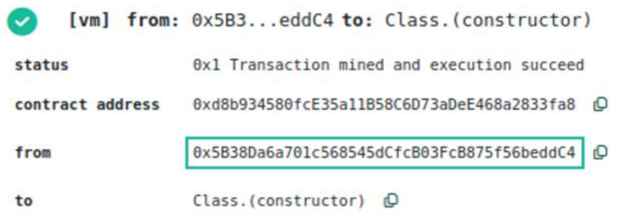
This contract stores the information on the class status and subject status of the student. Custom struct StudentInformation is designed to manage the information. Table 2 contains the details for the StudentInformation struct. The student contract contains the implementation of the functions corresponding to the fields in StudentInformation to get the information about the class and the subject. This information is also used by the class contract to process the registration. This information will be updated by the corresponding functions and only performed by the Manager. When a student calls the enrollment function, the information on the student's address will be automatically retrieved from the variable message. Sender. This makes sure only the student who uses their own account can enroll in the class. After that, the student contract will send a call to the class contract to handle the registration, and the class contract will only execute when a call is made from the student contract; there will be no middle actions for providing a specific address. This logic ensures that no one can take an illegal registration. The Manager designed the function on the Student contract to update students' studying when students completed a class and completed a subject: updateCompletedClass, removeCompletedClass, for updating when students completed the class, and the remove function for a mistake in update. These two functions will update the completedClass variable. Similarly, updateCompletedSubject and removeCompletedSubject will update the data on the completedSubject variable.



```
[vm] from: 0x03C...D1Ff7 to: Class.(constructor)
status              0x0 Transaction mined but execution failed
contract address    0xfAF646893C6D3Ef849FadD67FC1Ca3e347f409B7
from                0x03C6FcED478cBbC9a4FAB34eF9f40767739D1Ff7
to                  Class.(constructor)
creation of Class errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Error provided by the contract:
YouAreNotAdmin
```

(a)



```
[vm] from: 0x5B3...eddC4 to: Class.(constructor)
status              0x1 Transaction mined and execution succeed
contract address    0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  Class.(constructor)
```

(b)

Fig. 1(a) Error when not using an admin account for deployment got the error YouAreNotAdmin, and (b) Successful when using an admin account for deployment.

4.2. Testing

RemixIDE is used for testing to show the potential of using blockchain on the system by running scenario tests. For convenience, Table 3 is the setup for scenarios with the list of accounts and roles.

4.2.1. Role Contract Testing

In the proposed design, the Class contract and Student contract can be deployed only by the Admin, and the Role is managed by the Role contract. The plan is to test using an account that does not have the Admin role and an admin account to deploy the Class contract. Figure 1 is the test result. According to the test about the role contract, the role system designed works correctly, and the modifier can prevent an address without the required Role from executing restricted role functions.

4.2.2. Class Creation Testing

Here are the scenarios for test class creation:

- Test to create a simple class with no required subjects and no set priority students. This test checks the function of handling an invalid timestamp when inputting a timestamp in the past for RegistrationClosed. The result of the test is shown in Figure 2.
- Test to add a class with priority registration. This test tests the ability to handle errors when inputting an invalid list of priority students. The result of the test is shown in Figure 3.

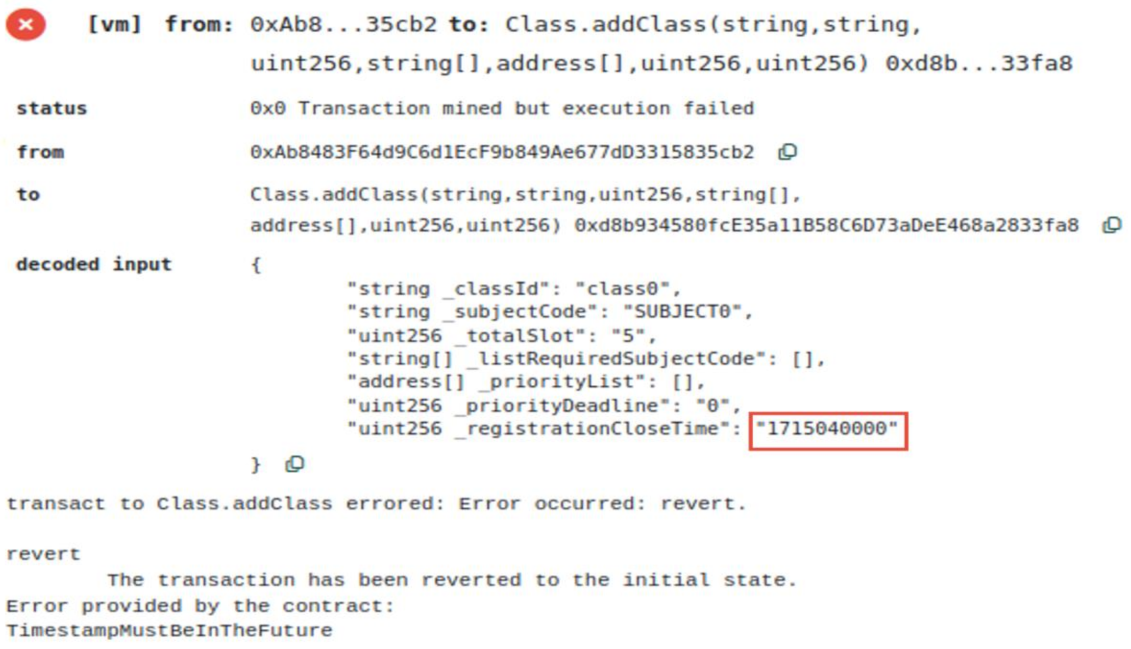
- Test required subject. This test tests the ability to handle errors when inputting an invalid list of required subjects in the input. The result of the test is shown in Figure 4.

Class creation testing shows that the proposed system, based on blockchain, can prevent invalid input (timestamp, invalid student address, invalid subject code), which can lead to errors in registration.

4.2.3. Registration Testing

For convenience in this test, call the address 0x4B2...C02db Student 1, and the address 0x787...cabaB is Student 2. Here is the test scenario for registration:

- *The enrollment in the class has been closed for registration.* First, create a class and wait for registration to close. After that, use the student account to enrol in class. Figure 5 is the result of the test.

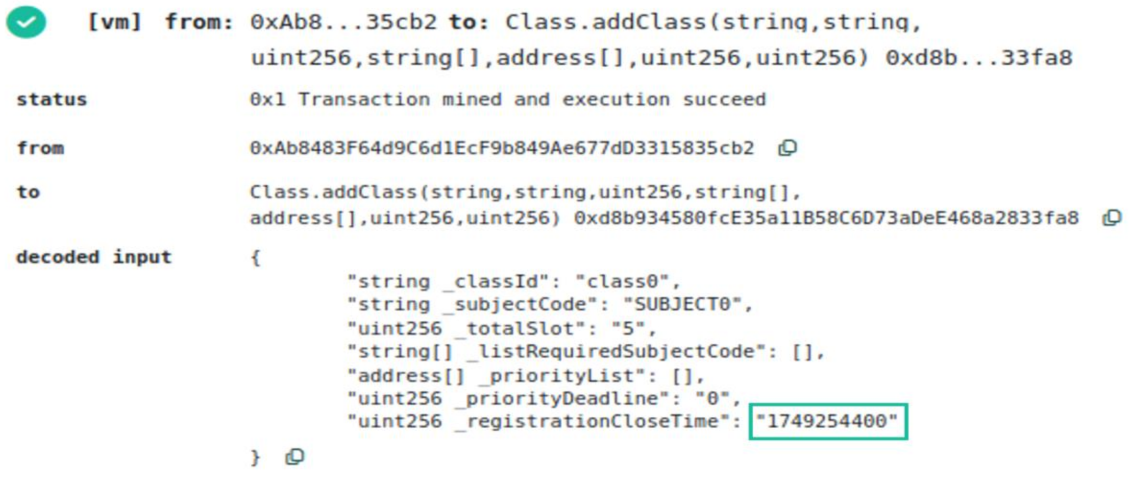


```
[vm]  from: 0xAb8...35cb2 to: Class.addClass(string,string,
            uint256,string[],address[],uint256,uint256) 0xd8b...33fa8

status          0x0 Transaction mined but execution failed
from            0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to              Class.addClass(string,string,uint256,string[],
                address[],uint256,uint256) 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
decoded input   {
                        "string _classId": "class0",
                        "string _subjectCode": "SUBJECT0",
                        "uint256 _totalSlot": "5",
                        "string[] _listRequiredSubjectCode": [],
                        "address[] _priorityList": [],
                        "uint256 _priorityDeadline": "0",
                        "uint256 _registrationCloseTime": "1715040000"
                } 
transact to Class.addClass errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Error provided by the contract:
TimestampMustBeInTheFuture
```

(a)

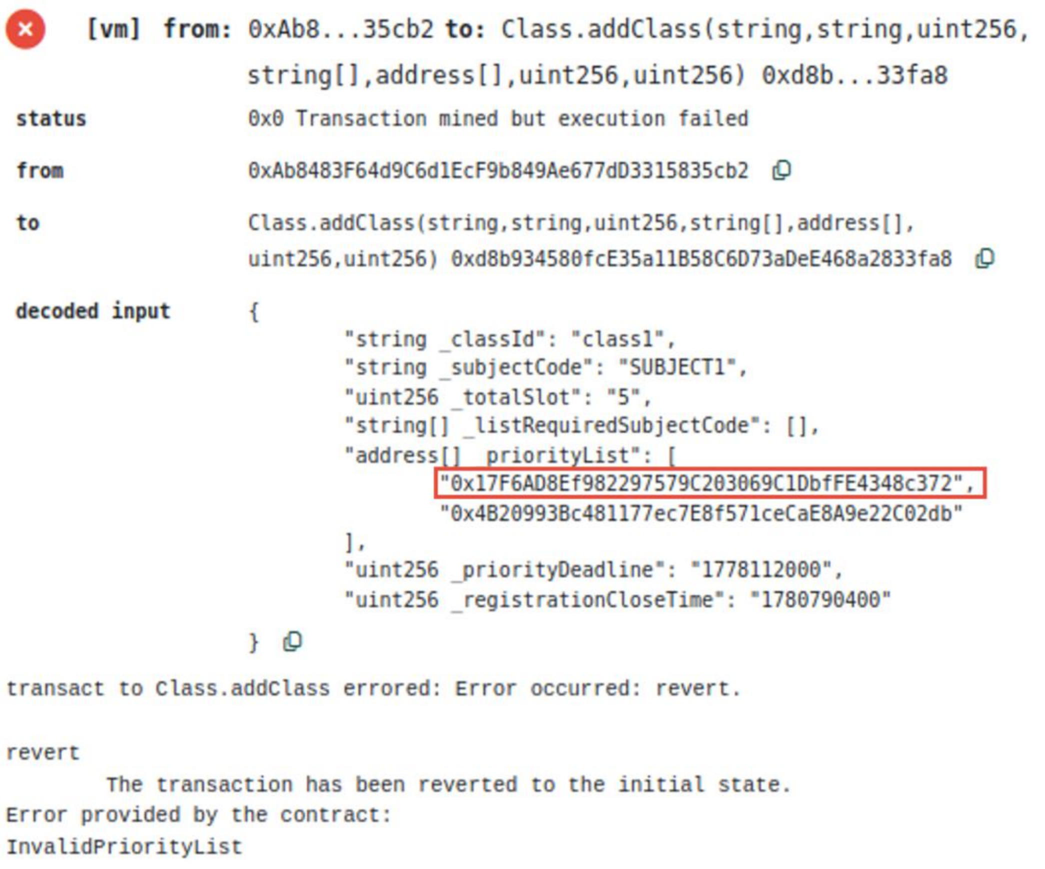


```
[vm]  from: 0xAb8...35cb2 to: Class.addClass(string,string,
            uint256,string[],address[],uint256,uint256) 0xd8b...33fa8

status          0x1 Transaction mined and execution succeed
from            0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to              Class.addClass(string,string,uint256,string[],
                address[],uint256,uint256) 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
decoded input   {
                        "string _classId": "class0",
                        "string _subjectCode": "SUBJECT0",
                        "uint256 _totalSlot": "5",
                        "string[] _listRequiredSubjectCode": [],
                        "address[] _priorityList": [],
                        "uint256 _priorityDeadline": "0",
                        "uint256 _registrationCloseTime": "1749254400"
                } 
```

(b)

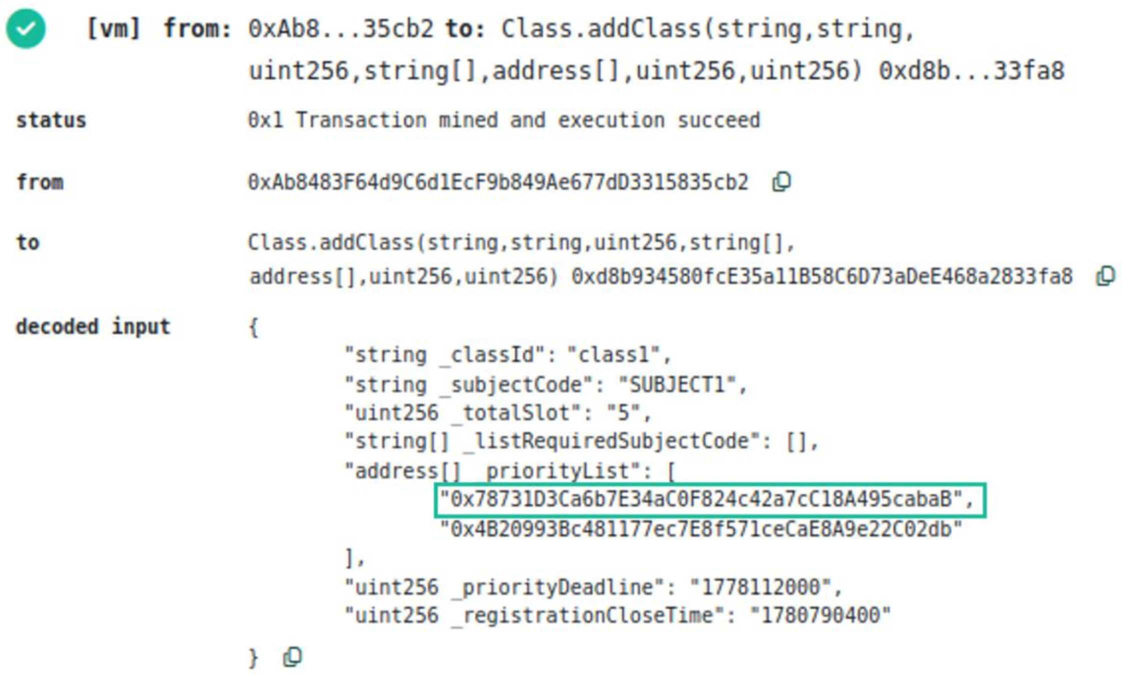
Fig. 2(a) Failed when creating a class with the timestamp in the past and got error TimestampMustBeInTheFuture, and (b) Success when creating a class with the timestamp greater than the current timestamp.



```
[vm] from: 0xAb8...35cb2 to: Class.addClass(string,string,uint256,
           string[],address[],uint256,uint256) 0xd8b...33fa8
status         0x0 Transaction mined but execution failed
from           0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to             Class.addClass(string,string,uint256,string[],address[],
               uint256,uint256) 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
decoded input  {
                       "string _classId": "class1",
                       "string _subjectCode": "SUBJECT1",
                       "uint256 _totalSlot": "5",
                       "string[] _listRequiredSubjectCode": [],
                       "address[] _priorityList": [
                               "0x17F6AD8Ef982297579C203069C1DbfFE4348c372",
                               "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db"
                       ],
                       "uint256 _priorityDeadline": "1778112000",
                       "uint256 _registrationCloseTime": "1780790400"
               }
transact to Class.addClass errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Error provided by the contract:
InvalidPriorityList
```

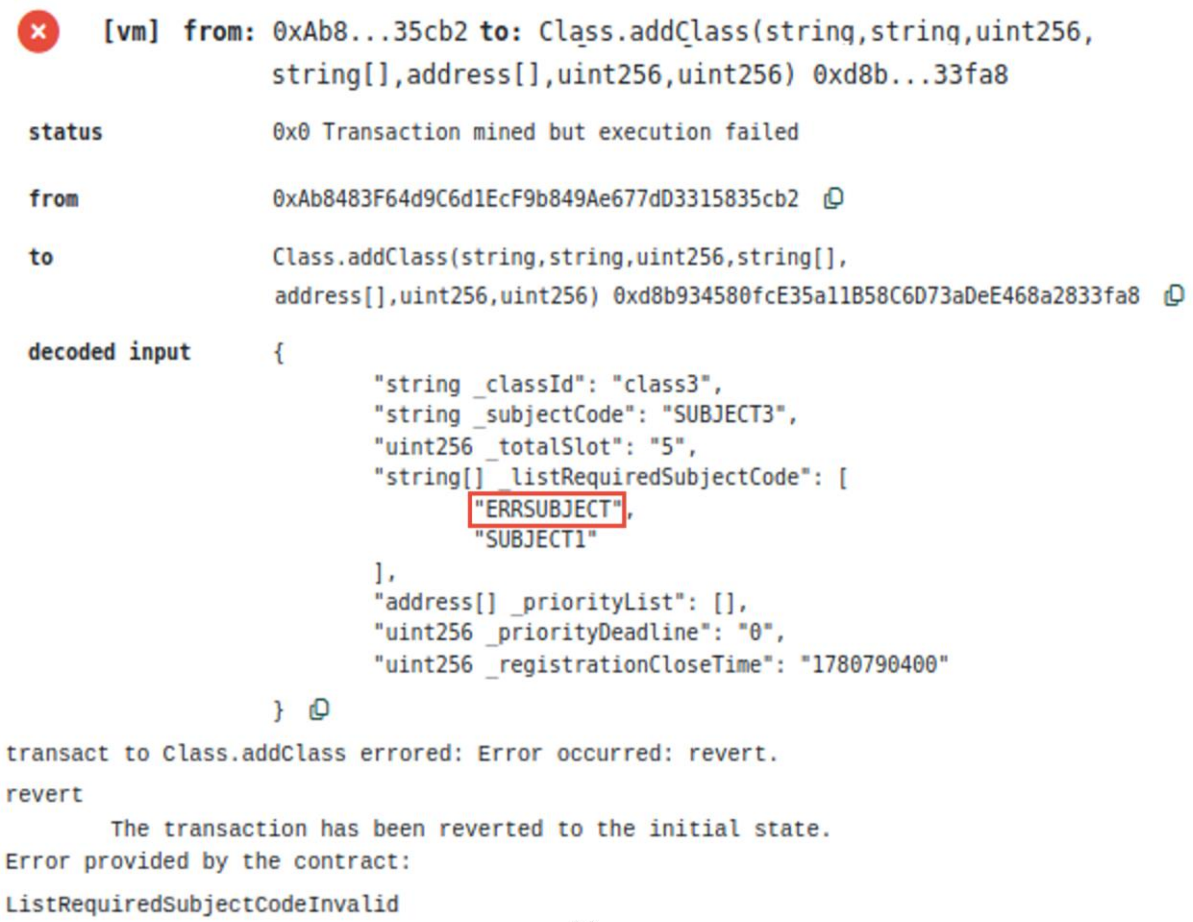
(a)



```
[vm] from: 0xAb8...35cb2 to: Class.addClass(string,string,
           uint256,string[],address[],uint256,uint256) 0xd8b...33fa8
status         0x1 Transaction mined and execution succeed
from           0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to             Class.addClass(string,string,uint256,string[],
               address[],uint256,uint256) 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
decoded input  {
                       "string _classId": "class1",
                       "string _subjectCode": "SUBJECT1",
                       "uint256 _totalSlot": "5",
                       "string[] _listRequiredSubjectCode": [],
                       "address[] _priorityList": [
                               "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB",
                               "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db"
                       ],
                       "uint256 _priorityDeadline": "1778112000",
                       "uint256 _registrationCloseTime": "1780790400"
               }
```

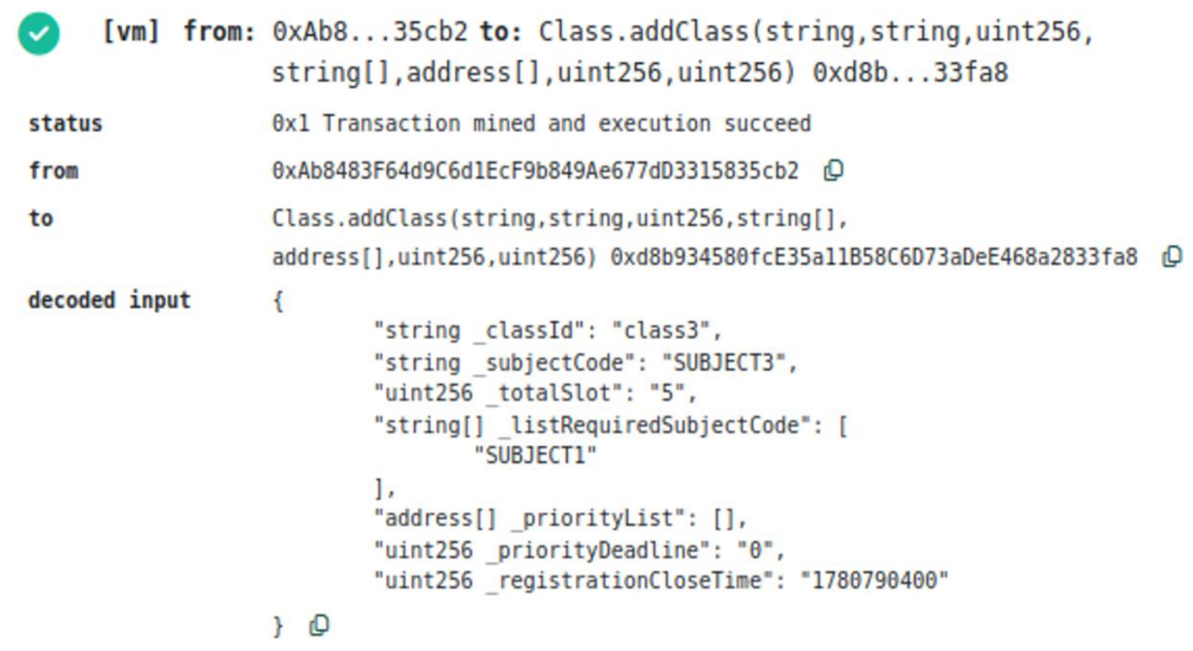
(b)

Fig. 3(a) Fail when the priority list has an invalid student and gets an error InvalidPriorityList, and (b) Success when all address in the priority list is student.



```
[vm] from: 0xAb8...35cb2 to: Class.addClass(string,string,uint256,
           string[],address[],uint256,uint256) 0xd8b...33fa8
status          0x0 Transaction mined but execution failed
from            0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to              Class.addClass(string,string,uint256,string[],
                address[],uint256,uint256) 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
decoded input   {
                        "string _classId": "class3",
                        "string _subjectCode": "SUBJECT3",
                        "uint256 _totalSlot": "5",
                        "string[] _listRequiredSubjectCode": [
                                "ERRSUBJECT",
                                "SUBJECT1"
                        ],
                        "address[] _priorityList": [],
                        "uint256 _priorityDeadline": "0",
                        "uint256 _registrationCloseTime": "1780790400"
                }
transact to Class.addClass errored: Error occurred: revert.
revert
        The transaction has been reverted to the initial state.
Error provided by the contract:
ListRequiredSubjectCodeInvalid
```

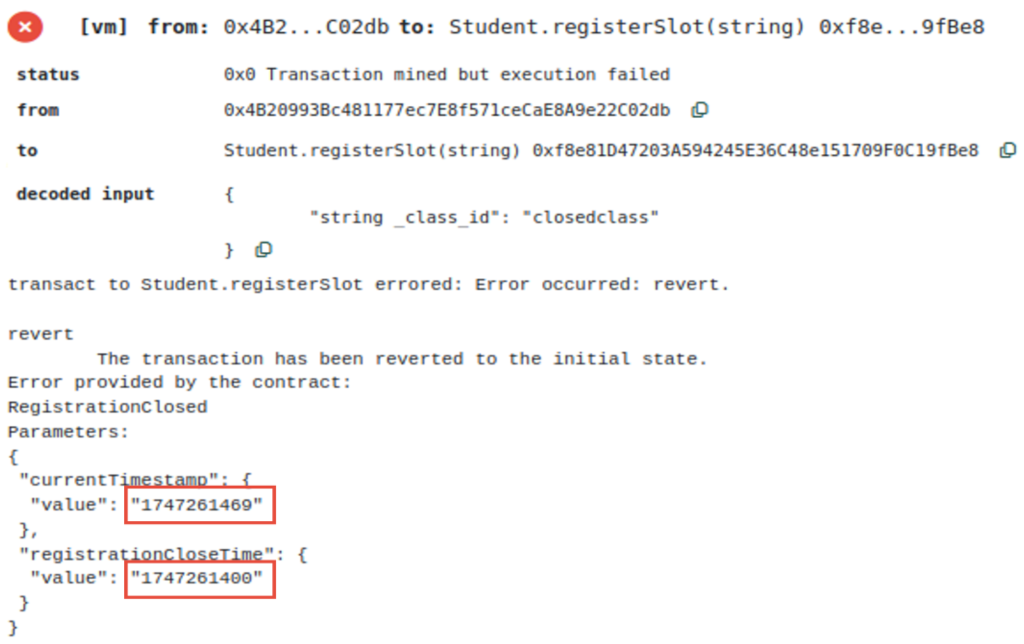
(a)



```
[vm] from: 0xAb8...35cb2 to: Class.addClass(string,string,uint256,
           string[],address[],uint256,uint256) 0xd8b...33fa8
status          0x1 Transaction mined and execution succeed
from            0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to              Class.addClass(string,string,uint256,string[],
                address[],uint256,uint256) 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
decoded input   {
                        "string _classId": "class3",
                        "string _subjectCode": "SUBJECT3",
                        "uint256 _totalSlot": "5",
                        "string[] _listRequiredSubjectCode": [
                                "SUBJECT1"
                        ],
                        "address[] _priorityList": [],
                        "uint256 _priorityDeadline": "0",
                        "uint256 _registrationCloseTime": "1780790400"
                }
```

(b)

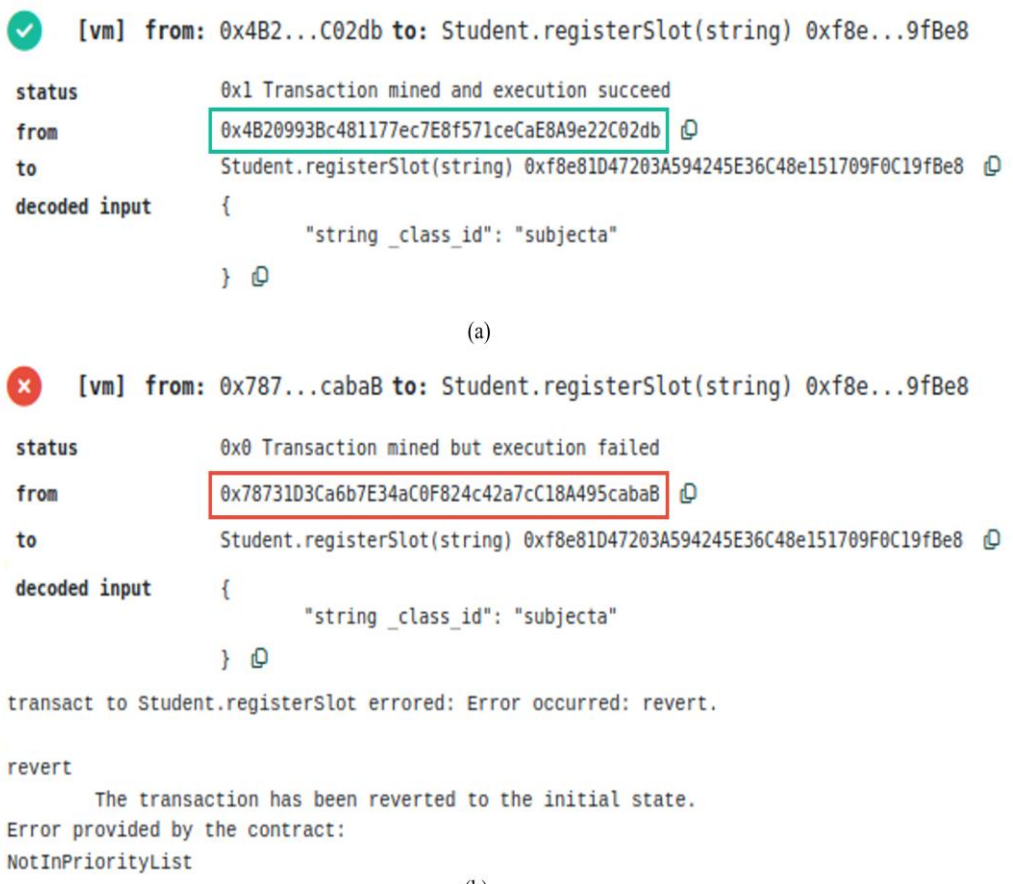
Fig. 4(a) Fail when the list has an invalid subject code and gets an error ListRequiredSubjectCodeInvalid, and (b) Success when all subject codes are valid.



```
[vm] from: 0x4B2...C02db to: Student.registerSlot(string) 0xf8e...9fBe8
status          0x0 Transaction mined but execution failed
from            0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
to              Student.registerSlot(string) 0xf8e81D47203A594245E36C48e151709F0C19fBe8
decoded input   {
                        "string _class_id": "closedclass"
                }
transact to Student.registerSlot errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Error provided by the contract:
RegistrationClosed
Parameters:
{
 "currentTimestamp": {
  "value": "1747261469"
 },
 "registrationCloseTime": {
  "value": "1747261400"
 }
}
```

Fig. 5 A student can not register when registration has closed. Time registered is greater than the deadline time for registration (1747261469 > 1747261400).



```
[vm] from: 0x4B2...C02db to: Student.registerSlot(string) 0xf8e...9fBe8
status          0x1 Transaction mined and execution succeed
from            0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
to              Student.registerSlot(string) 0xf8e81D47203A594245E36C48e151709F0C19fBe8
decoded input   {
                        "string _class_id": "subjecta"
                }
```

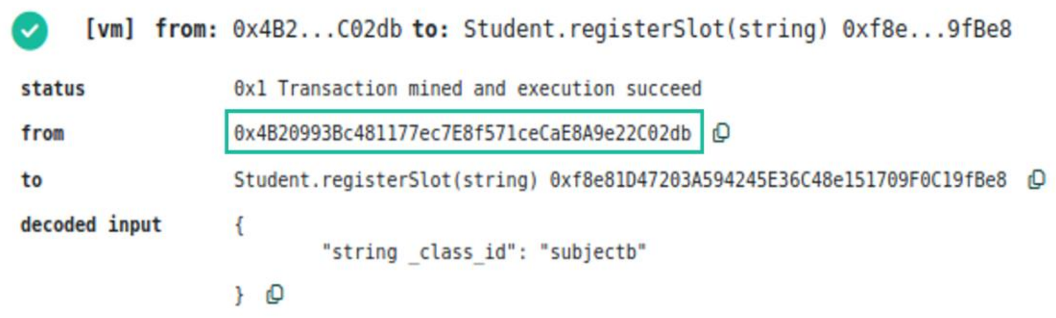
(a)

```
[vm] from: 0x787...cabaB to: Student.registerSlot(string) 0xf8e...9fBe8
status          0x0 Transaction mined but execution failed
from            0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB
to              Student.registerSlot(string) 0xf8e81D47203A594245E36C48e151709F0C19fBe8
decoded input   {
                        "string _class_id": "subjecta"
                }
transact to Student.registerSlot errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Error provided by the contract:
NotInPriorityList
```

(b)

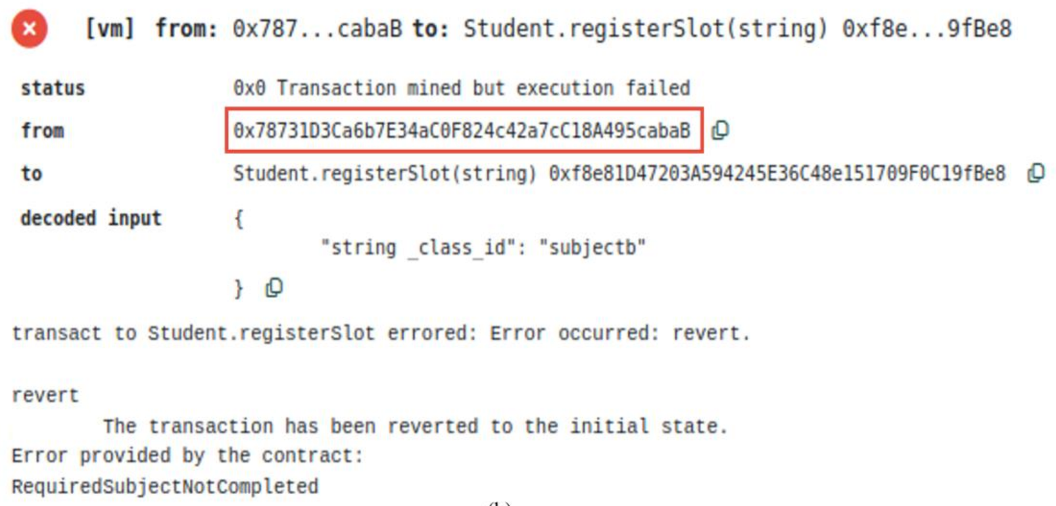
Fig. 6(a) Student 1 in priority can register successfully, and (b) Student 2 is not on the priority list and got an error NotInPriorityList when enrolled in the class.



```
[vm] from: 0x4B2...C02db to: Student.registerSlot(string) 0xf8e...9fBe8

status          0x1 Transaction mined and execution succeed
from            0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
to              Student.registerSlot(string) 0xf8e81D47203A594245E36C48e151709F0C19fBe8
decoded input   {
                        "string _class_id": "subjectb"
                }
```

(a)



```
[vm] from: 0x787...cabaB to: Student.registerSlot(string) 0xf8e...9fBe8

status          0x0 Transaction mined but execution failed
from            0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB
to              Student.registerSlot(string) 0xf8e81D47203A594245E36C48e151709F0C19fBe8
decoded input   {
                        "string _class_id": "subjectb"
                }
transact to Student.registerSlot errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Error provided by the contract:
RequiredSubjectNotCompleted
```

(b)

Fig. 7(a) Student 1 completed the required subject and registered successfully, and (b) Student 2 did not complete the required subject and got an error, RequiredSubjectNotCompleted when enrolling in class.

- Test the student's enrollment in the priority class. Create a class for SubjectA with priority for Student 1 and test enrollment with priority for two students. Figure 6 is the test result.
- Test the enrollment into the class, which requires the required subject. Create a class for Subject B with Subject A required, update Student 1 to complete Subject A, and test enrollment for two students. Figure 7 is the test result.

Registration testing shows that the proposed system can correctly handle the logic when students enroll in class. The timestamp obtained from the blockchain and used for processing the register is very valuable and can not be attacked by passing a timestamp for hacking. It is combined with no actors affecting when the function checks the conditions, which makes the system robust.

5. Conclusion

Integrating blockchain into the university registration system brings benefits for managing, tracking, and enhancing the security of the system. The proposed solution utilizes the smart contract feature to build the data storage and interact with the blockchain data. This study addresses the limitation of previous research, which only proposes a model and theoretical argument. This research provides details on the architecture design and implementation to show how to integrate blockchain into a registration system. The testing in some cases shows that the proposal can help satisfy the requirements for a registration system. In future work, the plan is to build a user interface (UI) for the system research, build a custom blockchain network that is suitable for the university, and integrate the registration system on this custom blockchain network.

References

[1] Mark C. Little et al., *The University Student Registration System: A Case Study in Building a High-Availability Distributed Application using General Purpose Components*, Advances in Distributed Systems, pp. 453-471, 2000. [CrossRef] [Google Scholar] [Publisher Link]

[2] Ruben Estevez et al., "A Model for Web-Based Course Registration Systems," *International Journal of Web Information Systems*, vol. 10, no. 1, pp. 51-64, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[3] Haroon Shakirat Oluwatosin, "Client-Server Model," *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 16, no. 1, pp. 67-71, 2014. [Google Scholar] [Publisher Link]

[4] Xiaowei Li, and Yuan Xue, "A Survey on Server-Side Approaches to Securing Web Applications," *ACM Computing Surveys*, vol. 46, no. 4, pp. 1-29, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[5] Megha Upreti et al., "Blockchain Based Registration Model for Higher Education," *Blockchain Frontier Technology*, vol. 1, no. 1, pp. 68-73, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[6] Luong Thuy Tien, and Luong Anh Linh, "Blockchain in Higher Education: Smart Contract Application in Credit Student Registration," *The University of Danang - Journal of Science and Technology*, vol. 22, no. 5B, pp. 11-15, 2024. [Google Scholar] [Publisher Link]

[7] Massimo Di Pierro, "What is the Blockchain?," *Computing in Science & Engineering*, vol. 19, no. 5, pp. 92-95, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[8] John Buford, Heather Yu, and Eng Keong Lua, *P2P Networking and Applications*, Morgan Kaufmann, 2009. [Google Scholar] [Publisher Link]

[9] Mohd Javaid et al., "A Review of Blockchain Technology Applications for Financial Services," *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 2, no. 3, pp. 1-18, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[10] Hanjie Wu et al., "Blockchain for Finance: A Survey," *IET Blockchain*, vol. 4, no. 2, pp. 101-123, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[11] J. Andrew et al., "Blockchain for Healthcare Systems: Architecture, Security Challenges, Trends and Future Directions," *Journal of Network and Computer Applications*, vol. 215, pp. 1-36, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[12] Anton Hasselgren et al., "Blockchain in Healthcare and Health Sciences-A Scoping Review," *International Journal of Medical Informatics*, vol. 134, pp. 1-10, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[13] Udit Agarwal et al., "Blockchain Technology for Secure Supply Chain Management: A Comprehensive Review," *IEEE Access*, vol. 10, pp. 85493-85517, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[14] Maciel M. Queiroz, Renato Telles, and Silvia H. Bonilla, "Blockchain and Supply Chain Management Integration: A Systematic Review of the Literature," *Supply Chain Management: An International Journal*, vol. 25, no. 2, pp. 241-254, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[15] Amr El Koshiry et al., "Unlocking the Power of Blockchain in Education: An Overview of Innovations and Outcomes," *Blockchain: Research and Applications*, vol. 4, no. 4, pp. 1-19, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[16] Preeti Bhaskar, Chandan Kumar Tiwari, and Amit Joshi, "Blockchain in Education Management: Present and Future Applications," *Interactive Technology and Smart Education*, vol. 18, no. 1, pp. 1-17, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[17] Faiza Loukil, Mourad Abed, and Khouloud Boukadi, "Blockchain Adoption in Education: A Systematic Literature Review," *Education and Information Technologies*, vol. 26, no. 5, pp. 5779-5797, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[18] Zibin Zheng et al., "An Overview on Smart Contracts: Challenges, Advances and Platforms," *Future Generation Computer Systems*, vol. 105, pp. 475-491, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[19] Vitalik Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," *Whitepaper*, 2014. [Google Scholar] [Publisher Link]

[20] Chris Dannen, *Solidity Programming*, Introducing Ethereum and Solidity, Apress, Berkeley, CA, pp. 69-88, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[21] Yoichi Hirai, "Defining the Ethereum Virtual Machine for Interactive Theorem Provers," *Financial Cryptography and Data Security International Conference on Financial Cryptography and Data Security*, Sliema, Malta, pp. 520-535, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[22] Ricardo Raimundo, and Albérico Rosário, "Blockchain System in the Higher Education," *European Journal of Investigation in Health, Psychology and Education*, vol. 11, no. 1, pp. 276-293, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[23] Guang Chen et al., "Exploring Blockchain Technology and its Potential Applications for Education," *Smart Learning Environments*, vol. 5, no. 1, pp. 1-10, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[24] Shivani Pathak et al., "Smart Contract for Academic Certificate Verification Using Ethereum," *Advanced Computing and Intelligent Technologies*, pp. 369-384, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[25] Jiin-Chiou Cheng et al., "Blockchain and Smart Contract for Digital Certificate," *2018 IEEE International Conference on Applied System Invention (ICASI)*, Chiba, Japan, pp. 1046-1051, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[26] Yassynzhan Shakan et al., "Verification of University Student and Graduate Data using Blockchain Technology," *International Journal of Computers Communications & Control*, vol. 16, no. 5, pp. 1-16, 2021. [Google Scholar] [Publisher Link]

[27] Nero Chaniago, Aulia Arif Wardana, and Parman Sukarno, "Electronic Document Authenticity Verification of Diploma and Transcript using Smart Contract on Ethereum Blockchain," *Register: Scientific Journal of Information Systems Technology*, vol. 7, no. 2, pp. 149-163, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[28] Punam Bedi et al., "Smart Contract based Central Sector Scheme of Scholarship for College and University Students," *Procedia Computer Science*, vol. 171, pp. 790-799, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[29] Jadhav Swati, and Pise Nitin, "CryptoScholarChain: Revolutionizing Scholarship Management Framework with Blockchain Technology," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 8, pp. 652-659, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[30] I Ketut Gunawan et al., "Smart Contract Innovation and Blockchain-Based Tokenization in Higher Education," *Journal of Education Technology*, vol. 5, no. 4, pp. 636-644, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[31] Ashis Kumar Samanta, Bidyut Biman Sarkar, and Nabendu Chaki, "A Blockchain-Based Smart Contract Towards Developing Secured University Examination System," *Journal of Data, Information and Management*, vol. 3, no. 4, pp. 237-249, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[32] Apoorv Jain et al., "Smart Contract enabled Online Examination System Based in Blockchain Network," *2021 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore India, pp. 1-7, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[33] Yunjie Chen, and Zhihai Rong, "Evolution of the External Owned Account Trading Network on Ethereum," *2020 Chinese Automation Congress (CAC)*, Shanghai, China, pp. 3369-3373, 2020. [CrossRef] [Google Scholar] [Publisher Link]