ValuED: A Blockchain-based Trading Platform to Encourage Student Engagement in Higher Education

Aydin Abadi[a,1], Jin Xiao[2], Roberto Metere[b,d,3] and Richard Shillcock[c,4]

[a]University College London, UK
[b]Newcastle University, UK
[c]University of Edinburgh, UK
[d]The Alan Turing Institute, UK

ARTICLE INFO

Keywords:
Smart contract
Cryptocurrency
Higher education
Student experience
Student engagement
System implementation

ABSTRACT

The provision of higher education has been changing ever more quickly in the UK and worldwide, as a result of technological, economic, and geopolitical factors. The Covid-19 pandemic has accelerated such changes. The "student experience"—the interaction of students with their institution and with each other—has been changing accordingly, with less face-to-face contact. In this work, we have explored a way to improve student engagement in higher education. We describe "ValuED", a blockchain-based trading platform using a cryptocurrency. It allows students both to buy and sell goods and services within their university community and to be rewarded for academic engagement. ValuED involves a reputation system to further incentivise participants. We describe the implementation and piloting of this platform and draw conclusions for its future use. The platform's source code is publicly available.

1. Introduction

The nature of higher education in the UK and beyond has been changing increasingly rapidly along many dimensions. Student numbers have increased considerably. In 2017/18, a record 50.2% of English 17- to 30-year-olds participated in higher education. In the same year, the UK took 12% of all higher education students who were studying overseas at universities in the Organisation for Economic Co-operation and Development (OECD). The mix of cultural and sub-cultural backgrounds is greater than ever.

Since their UK introduction in 1998, tuition fees for higher education have risen. Currently, in England they are capped at £9,250 a year for the UK and Irish students. The situation varies in the other constituent nations of the UK. For international undergraduate students in the UK, annual tuition fees currently start at £9,250 and can rise to as much as £64,652 for medical degrees. This market in higher education has led to intense attention to "value for money" and, in particular, to the "student experience", construed not just in academic terms but in broader social terms [4, 25, 26, 23, 32, 35]. This diversity within the student body together with its perceived buying power has encouraged greater student choice both between and within degrees [19]. More options within the same degree course means reduced continuity within the student cohort; students find themselves in different groups in different places at different times. Students taking part-time employment during term-time has led to further social fragmentation of the student body [18, 31]. With the widespread availability of laptop computing, more delivery of academic content has acquired an online dimension. Lectures are widely expected to also be available online, to be watched off-campus [23, 29, 41]. The Covid-19 pandemic has affected all of these issues, in ways that are still unfolding. The continuity of many higher education practices has been severely disrupted. All of the dimensions discussed above have impacted upon how higher education students interact with higher education institutions and with each other.

In this work, we explore one mechanism designed to provoke and facilitate such interactions—a blockchain-based cryptocurrency capable of being integrated with both the academic and the social components of higher education. In particular, we describe "ValuED", which is a blockchain-based trading platform. It lets students (i) buy and sell goods and services within their university community and (ii) be rewarded for academic engagement. Furthermore, ValuED involves a reputation system to further incentivise participants. We have fully implemented a prototype of this platform,

[1]aydin.abadi@ucl.ac.uk
[2]xiaojin971212@gmail.com
[3]roberto.metere@ncl.ac.uk
[4]rcs@inf.ed.ac.uk

i.e., developed Ethereum-based back-end smart contract and front-end Graphical User Interface (GUI). We describe its implementation, analyse its cost, report the result of its piloting, and make its source code publicly available. The ValuED platform can have various use cases beyond provoking student engagement in higher education.

2. Related work

2.1. CamLETS

CamLETS is a barter system that enables its members to exchange skills, services, and goods without using money [11]. CamLETS's developers aimed to "enhance users' prosperity, build a sustainable community, utilise members' talents, nurture the unique quality of participants' neighborhood, and have fun". Members join the system by paying an initial fee as well as an annual subscription. The system uses a community currency called "cams". Each member is given an account balance, initially set to 0. They earn or spend cams by engaging in transactions with other members. Their cam balance is maintained by a centralised server accessible by only a small number of members who maintain the system. Members offer various goods or services such as accommodation, accounting, translation, or babysitting. Likewise, they list any goods or services they are interested in, to be paid for in cams. A member providing a service is paid at a fixed rate (usually 10 cams per hour) hard-coded into the system.

CamLETS is a centralized system. Therefore, it suffers various drawbacks, such as having a single point of failure, or having to trust the members who maintain the system. Members rely on their monetary balance remaining intact and their requests not being filtered, modified, or delayed. Entry and subscription fees may deter potential users.

2.2. Piazza

Other attempts to establish communication across distributed communities include "Piazza" [39] and "Top Hat" [34]. Piazza is a learning management platform that lets registered students post their questions to be answered by instructors or other students. It is designed to model face-to-face group discussions. It was launched in 2009 as a shared place in which students could engage in discussion outside the classroom, with speedy response times. There are two main roles in this system: learners and instructors. Once learners have enrolled in online classes and have logged on, they have access to questions and answers posted by their peers. Instructors (professors and tutors) can enhance students' motivation and engagement, can moderate the discussion, and can endorse accurate answers. Similar to Google Docs, Piazza's content is updated in real-time. It has attracted considerable attention from academics. However, its focus is limited to students' activities in the classes in which they have enrolled.

Top Hat provides tools that track lecture attendance, features interactive slides, and manage classroom discussions and questions. An instructor can create a virtual classroom for a course. All course activities, such as updating course materials, posting assignments, and tests, take place in this virtual classroom. Top Hat allows instructors to gauge the progress of students and assist them before, during, and after the class. Nevertheless, the service Top Hat provides is limited to a particular class. It cannot support peer-to-peer interactions.

Both Piazza and Top Hat are based on a centralised system, designed to help instructors deliver lectures and course-related material. They do not improve students' experience beyond the classroom.

2.3. Open-city app

The "Open-city app" [20] is a currency platform developed under the "City-of-Helsinki" project[21]. The vision of the project is to create a loyalty currency for cultural functions. Currently, the platform is based on a centralised system, rather than a blockchain. Users can earn tokens through volunteer works that benefit the community. The platform also offers a reputation system. It shows how much a user's behaviour contributes to the community. Users with higher reputations may, for instance, receive discounts on certain services or have more say in community-related matters.

2.4. Smileycoin

Smileycoin is a blockchain-based commercial cryptocurrency released in 2014 [33]. It is used as a financial incentive in education. In particular, its developers initially used it as a platform to reward students for their studies. The platform was integrated into the online learning platform "tutor-web" [38] and was successfully used by a large number of students between 2014 and 2017. Each coin is called a "Smileycoin". In late 2021, each Smileycoin was worth $0.000048, with an overall market value of $758,852. Its developers have tried to attract companies to participate in the project by buying local coins. However, the companies have preferred to donate rather than receive coins. On

this platform, students receive awards for getting top grades. The platform offers a chatroom in which students can buy or sell services using Smileycoins. This platform does not support Ethereum-like smart contracts, which considerably limits its functionality. Recently, Ansari *et al.* [8] proposed a student rewarding mechanism (similar to Smileycoin) but it does not offer all features that Smileycoin offers (e.g., students' chatroom); however, their platform has been built on (flexible) Ethereum smart contracts that potentially could support easier future enhancement.

Technical details about the above platforms have not been published so far; therefore, is not feasible to assess the security of them.

2.5. EduCTX

Turkanovic *et al.* [37] propose an interesting blockchain-based credit and grading platform, called "EduCTX", for higher education. In this platform, students receive credits for completing their course-related activities from the corresponding higher education institute at which they initially enrolled. This platform stores all grades a student receives, while keeping students' real identity hidden. Later, students can (mathematically) prove to a third party (a potential employer) the degrees they received for a specific course. Thus, EduCTX uses blockchain to keep a uniform record of students' credits. The platform is based on a relatively new blockchain called "Ark". This blockchain uses a certain consensus protocol called "Delegated Proof of Stake" (DPoS) which leads to an efficient consensus but it has been criticised for turning a blockchain (and accordingly any platforms that use it) into a centralised system [10]. Although EduCTX offers appealing features, it does not support any interaction among students for academic or non-academic activities.

2.6. Conclusion

In contrast to all of the systems reviewed above, our own platform, ValuED, uses a decentralised platform built on top of the well-known Ethereum blockchain which (a) has been proven to be secure, and (b) supports arbitrary smart contracts that allow users to create on-chain proposals for the goods or services they would like to receive or offer. ValuED offers a secure reputation mechanism that is able to reward students for their academic and social engagement.

3. Tools

3.1. Blockchain technology

Bitcoin [28] was introduced in 2009. It is the first and most prominent decentralised cryptocurrency, to date. Critically, Bitcoin offers a secure digital currency without the involvement of a single trusted authority (e.g., a bank or government). Bitcoin relies on an elegant technology called "blockchain". Blockchain is a distributed ledger (or database) where all queries for the transfer of digital coins, i.e., transactions, are recorded in a chain of blocks, where each block can be considered as a page in a ledger and contains a set of transactions, a pointer to the previous block, and some metadata such as a certain puzzle's solution. New blocks (or pages) are added to the chain as time passes and the nodes that maintain the blockchain, called "miners", create new valid blocks. The blocks of those miners that solve a certain moderately hard computational puzzle, called "proof of work", are added into the chain. To incentivise the miners to participate in the block creation procedure, the system automatically rewards those miners that manage to solve each puzzle by giving them a certain number of digital coins. Everyone can participate in the blockchain's network and become a miner. Every node in the network locally maintains a copy of the chain. It has been shown that if the majority of the nodes are honest, then honest nodes have the same view of the blockchain. Since Bitcoin's introduction, numerous decentralised cryptocurrencies have been proposed, such as Ethereum [40], Cardano [12], and Zcash [30].

Blockchain technology has various applications in areas beyond cryptocurrencies. For instance, it has been used in secure multi-party computation [5, 42], verifiable data storage at cloud computing [1, 3], time-stamping mechanisms [2, 15], healthcare [13, 17], charitable organizations [9, 36], and supply chains [16, 24] to name a few. As we discuss below, this technology is the backbone of (decentralised) smart contracts.

3.2. Smart contracts

Cryptocurrencies such as Bitcoin and Ethereum, in addition to offering a decentralised currency, support computations on transactions. In this setting, often a certain computational logic is encoded in a computer program, called a *smart contract*. Although the first decentralised cryptocurrency, Bitcoin, could support smart contracts, their

functionality was limited. Ethereum was proposed to address these limitations. To date, Ethereum is the predominant cryptocurrency framework enabling users to define arbitrary smart contracts. In this framework, contract code is stored on the blockchain and executed by all parties (i.e. miners) maintaining the cryptocurrency, when the program inputs are provided by transactions. The correctness of the program's execution is guaranteed by the security of the underlying blockchain components. To prevent a denial of service attack, the framework requires a transaction creator to pay a fee, called *gas*, depending on the complexity of the contract running on it. Nonetheless, Ethereum smart contracts suffer from an important issue; namely, the lack of privacy, as it requires every contract's data to be public, which is a major impediment to the broad adoption of smart contracts when a certain level of privacy is desired. To address the issue, researchers/users may utilise existing decentralised frameworks which support privacy-preserving smart contracts [22]. However, due to the use of generic and computationally expensive cryptographic tools, they impose a significant cost to their users. Alternatively, one can design efficient tailored cryptographic protocols that preserve (contracts) data privacy, even though non-private smart contracts are used.

Solidity is a new object-oriented, high-level language used to develop smart contracts [14]. This language, influenced by mainstream languages such as C++ and Python. With Solidity, developers can create contracts for users, with a simple token contract, including crowdfunding, voting multi-signature wallets, and blind auctions. As Solidity itself is free and open source technology, it motivates many developers to learn it, enabling them to move tokens of value in any Ethereum-based system privately.

3.3. Digital wallets

A digital wallet is a payment tool that creates, stores, and manages the wallet owner's keys. Every time a user requests it, the wallet creates a pair of public and private keys. The private key is used to sign transactions and spend digital coins. The public key is used by the blockchain network verifiers (e.g., miners) to verify a signed transaction. A user's public key has another use in the blockchain network; namely, its hash value is considered as the user's unique ID (or account number). Thus, a digital wallet can be considered as a tool to generate a unique ID for a user without requiring it to interact with any other party. A digital wallet can offer other services as well, such as signing transactions on behalf of its owners or keeping track of its owners digital money.

A digital wallet comes in hardware or software forms, also known as cold and hot wallets respectively. In this work, we use the latter form due to its popularity and ease of use. In particular, we use "MetaMask", which is a Google Chrome extension. It allows users to interact with the Ethereum blockchain effortlessly. It allows a user to create and hold their own Ethereum account details securely, using a specific username and password. This wallet can be easily connected to any tailor-made user interface designed to interact with a smart contract as long as appropriate libraries, such as "web3.js", are used in the user interface. Once the wallet is configured correctly in the user interface, every time a user sends a transaction via the user interface to the blockchain, the wallet is popped up to ask the user's permission for signing the transaction. The user can grant permissions by simply clicking a button. When the permission is granted, then the wallet signs the transaction on the user's behalf and sends the signed transaction to the blockchain.

3.4. Cryptographic tokens

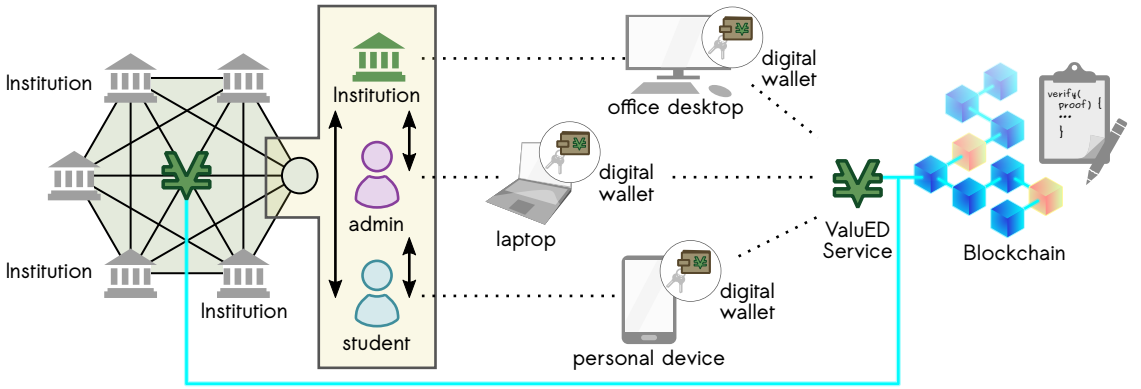
In general, a token is a representation of a particular asset or utility. It can be in analogue (e.g. paper-based) or digital format. A cryptographic token is an example of a digital token, which is often maintained by a specific cryptocurrency such as Ethereum. The value of a token can be dependent or independent of the underlying cryptocurrency's market value. Programmable cryptocurrency platforms such as Ethereum, have made it possible to easily program tokens as well. For instance, we can design a smart contract in such a way that a certain number of tokens is automatically transferred from a lecturer to a student, if the student can prove to the smart contract that they have attended a certain class. Usually, a smart contract keeps track of token transfer and maintains token balance. The security of a smart contract and its underlying blockchain ensures that an unauthorised party cannot manipulate the token balance. However, an insecure smart contract's design can lead to serious consequences, such as token manipulation and loss. One well-known example is the attack on a blockchain-based crowdfunding platform called "Decentralized Autonomous Organization" (DAO) that let the attackers take $60 million from the DAO smart contract. Later, to restore the stolen funds, Ethereum developers decided to split (or fork) the Ethereum blockchain into two parts, which raised serious disputes. Tokens are created through an initial coin offering, where they are distributed to those parties who have paid (in cryptocurrency). In this work, we use tokens and different token balances for various reasons. For instance, we use tokens to reward students when they (i) attend classes or (ii) provide goods and services to others. We also allow students to reward fellow students by sending a certain number of tokens in return for shared lecture

notes, for instance, or a second-hand textbook. In the case of student-to-student trades, each student's token balance is considered to be their reputation built over time. Each user is given a fixed number of tokens to start with, once registered in the system by the system administrator.

4. ValuED platform

In this section, we describe the ValuED platform in more detail. Fig. 1 depicts this platform's workflow.

Figure 1: High-level description of ValuED.



4.1. ValuED's design and basic functionality

Our approach to the ValuED's design is determined by the platform's intended participants and the potential interactions between them. There are three types of participants: a single manager, administrators, and students. We developed a web application, i.e., GUI, that interfaces with the ValuED smart contract and its users, Fig. 2 presents a screenshot of the ValuED GUI. It has been written in "JavaScript" programming language. The ValuED GUI lets parties interact with the smart contract smoothly without having to know all details of the smart contract. The smart contract enables the following interactions between the key parties: account registration, trade proposal dissemination, token distribution, attendance verification, and peer-to-peer trade among students. Parties and their interactions are illustrated in Fig. 3. Below, we describe each role and explain the level of authorisation granted to it.

4.1.1. Manager

The manager is the party who initially designs, implements, and deploys the ValuED smart contract, i.e., registers the contract in the blockchain: thus, only one manager is allowed. This smart contract has been designed to make the manager the only party who has the authorisation to register all other parties and allocate a role to each party, e.g., admin(s). The manager defines and encodes the platform's rules, and underlying incentive mechanism in the contract. The manager can be a single party, e.g., a trusted party appointed by a university, or a set of parties, e.g., a set of staff appointed by a consortium of universities. However, after the contract has been deployed, the manager cannot change the contract terms and rules. In this case, its power will be limited to only onboarding new users and allocating a role to them. The ValuED smart contract lets the manager delegate some of its responsibilities (e.g., students registration) to registered admins.

4.1.2. Admin

An admin can be considered as an assistant to the manager. In a university, the admin will refer to a secretary or an academic. Upon registration, an admin is able to enroll students, register other admins, create a random session key for each lecture, tutorial or event, and distribute tokens among the users. These operations are performed by calling certain functions of the smart contract. The initial number of tokens distributed among the users is defined by the admin. Certain admins, such as lecturers or tutors, may be charged with ensuring students are rewarded with tokens for

Figure 2: Screenshot of a trading page of a student.

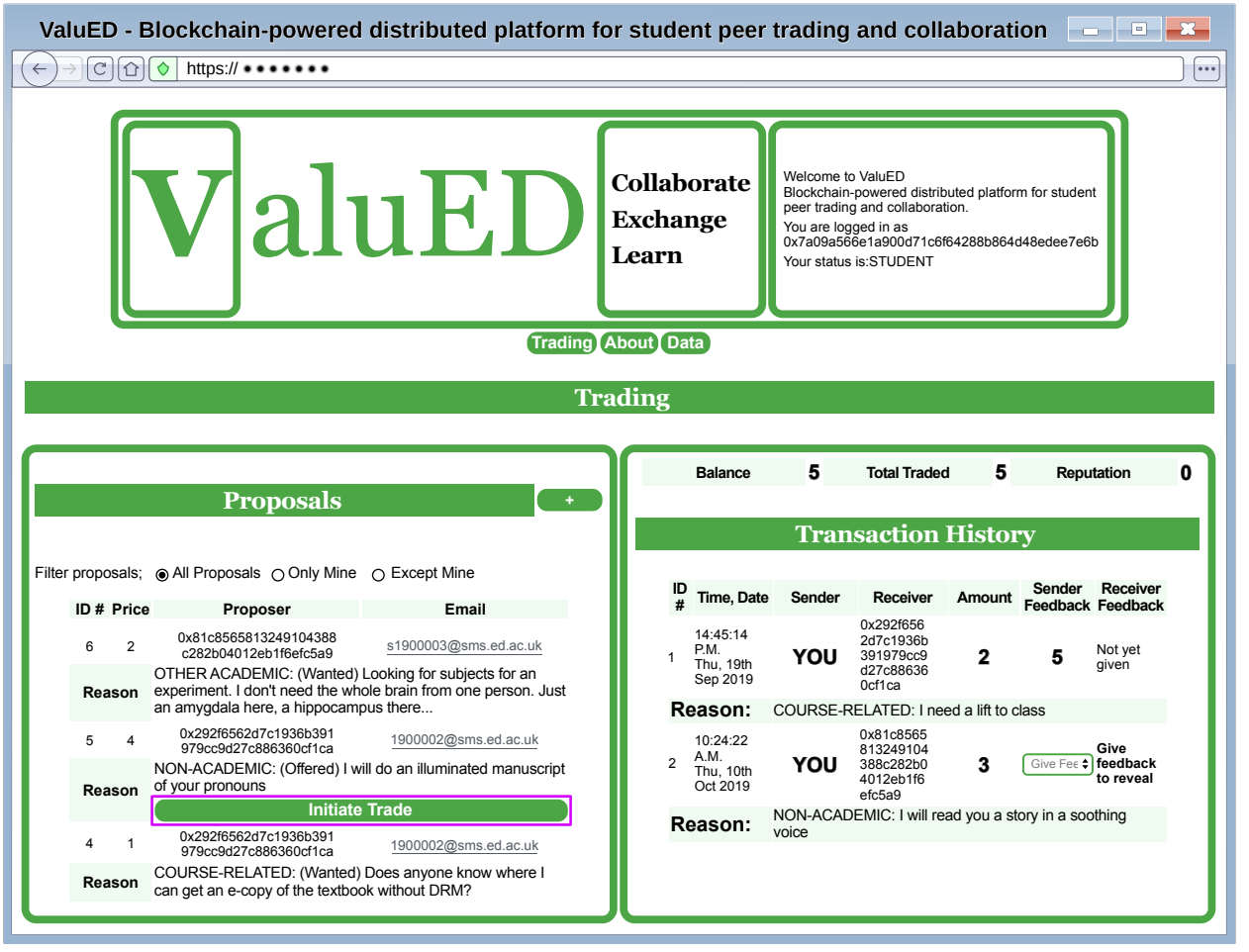
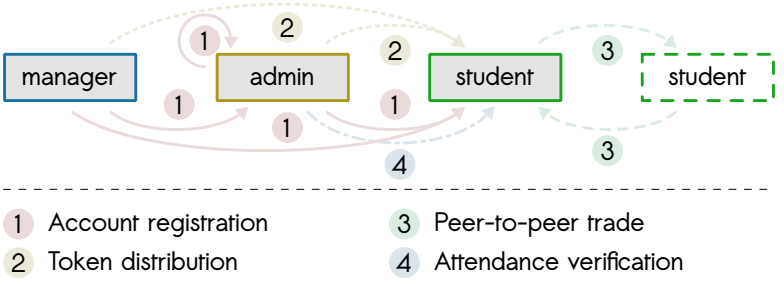


Figure 3: Role model of the ValuED platform.



successfully engaging with their course in some way. Attendance at tutorials and lectures is an example of academic engagement that cannot be given a formal mark but can be encouraged by a token reward.

4.1.3. Student

Students are the majority of users in the ValuED platform. They are initially registered in the smart contract by the manager or an admin. In the same phase, a fixed number of tokens is assigned to each student. Registered (or

authorised) students can participate in the ValuED trading system to spend and earn tokens. They can create proposals for offering or receiving specific goods or services, respond to proposals, send tokens to other students, leave feedback, or check in for attending a lecture, class or tutorial (using a fresh session key provided by the lecturer). A student's trading record (e.g., reputation, token balance, the total number of trades) is stored by the ValuED smart contract and remains unforgeable, due to the security of the blockchain and smart contract. Importantly, this record is public and can be retrieved by anyone, thus, enforcing transparency.

4.2. ValuED's architecture and implementation

4.2.1. Architecture

The architecture of ValuED requires users to have a digital wallet, e.g., MetaMask, allowing them to have a unique ID and to digitally sign transactions that they send to the blockchain. With their wallets, users can access ValuED functionality from a web application. The web application is a front-end that interfaces with a run-time environment for smart contracts in Ethereum. We can broadly subgroup such an environment into two modules: smart contract and ledger (data storage). The smart contract module is where the desired computational logic is encoded and it maintains users' token balances and ensures only authorized users can change certain variables or execute specific functions encoded in the smart contract. The ledger module is a standard distributed data storage that ensures the *immutability* and *security* of users' data. More details are provided in Appendix A.

Figure 4: The architecture of ValuED.

4.2.2. Implementation of a trading task

The interactions between roles are implemented based on the systemic design of smart contracts that are run in Ethereum. Once the contract is compiled, it exposes the information of the contract application binary interface (ABI) which is the standard way to interact with contracts in the Ethereum ecosystem. Defining the smart contract includes: (i) the declaration of entities and configuration of other items of the platform, and (ii) the declaration of functions to process their interactions. In the rest of this section, we describe the implementation details of a trading task, as an example of what the implementation of ValuED has involved. We discuss the whole implementation in Appendix C.

A trading task is initiated by a student who wants to trade with a peer. This particular task allows us to refine the concept of *reputation* introduced in Section 4.1. Students can trade what they wish by submitting a *proposal* to the ValuED platform. If the proposal is accepted, then a transfer of tokens is performed: the transfer details are defined in a *transaction*. Listings 1 and 2 show the code (written in Solidity) that defines proposals and transactions.

Listing 1: Definition of a proposal in the ValuED smart contract.

```
1    struct Proposal {
2        uint tokens;     /// Tokens required for this proposal (sent or received)
3        address creator; /// Who offers the proposal
4        string email;    /// Used to exchange extra information
5        string reason;   /// Description of the proposal
6        uint id;         /// The ID of the proposal
7        bool active;     /// An active proposal is one where tokens have not been transferred yet
8    }
```

Listing 2: Definition of a transaction in the ValuED smart contract.

```
struct Transaction{
    address sender;       /// Who sends the tokens
    address receiver;     /// Who receives the tokens
    string reason;        /// Description of the transaction
    int senderFeedback;   /// Feedback provided by the sender
    int receiverFeedback; /// Feedback provided by the receiver
    uint id;              /// The ID of the transaction
    uint tokens;          /// Tokens transferred in this transaction
    string creationTime;  /// Time of creation
}
```

A proposal includes the following parts.

1. *Number of tokens*. In this field, users (i.e., the proposal creators) can specify the number of tokens they are willing to consider for a trade. The specified number of tokens can be (a) the number of tokens they are willing to send in exchange for receiving a certain service or (b) the number of tokens they would like to receive in exchange for a certain service that they will deliver. In the former case, the underlying smart contract ensures that users cannot specify in the field a number of tokens that is more than the total tokens they possess.

2. *Email address of a proposal creator*. In this field, users who have created the proposal can specify their email addresses and other contact information. This information allows those who are interested in the proposal to contact the creator for more information beyond what has been stated in the proposal, before they formally commit to the proposal and engage in the trade.

3. *Categories of a proposal*. This field provides two options for the user: (a) "wanted", and (b) "offered". The wanted option is selected when the user wants to receive a certain service in exchange for a fixed number of tokens. After the wanted proposal is created and sent to the blockchain, that specified number of tokens is locked. Later, if a service provider is interested in the proposal, it engages in the same proposal and initiates a trade to express its interest (by invoking a function of the smart contract). In this case, the locked tokens are transferred to the token balance of the service provider. In contrast, the offered option is chosen when the user wants to offer a certain service in exchange for a fixed number of tokens. In this case, the service provider sends its proposal to the blockchain. Later, if anyone is interested in the proposal, it engages in it and initiates a trade. Upon initiating a trade, the number of tokens that are specified in the proposal is transferred from the service recipient's token balance to the service provider's. In both wanted and offered proposals, after the tokens are transferred, the service can be delivered "out-of-bounds" and each party can score its counter-party on the same trade. The score is added to the parties' public reputation list. Moreover, in general, the contract can allow the users to terminate their proposal and unlock their tokens after a certain time period if no one shows any interest in their proposal.

4. *Nature of the trade*. In this field, the user can select one of the following options: (a) course-related, (b) other academic, and (c) non-academic. For instance, academic trades can include proof-reading an assignment, lending textbooks or lecture notes, peer tutoring, inter-year mentoring, and so on. Non-academic trades might be a lift onto campus, cat-sitting, dog-walking, befriending, coffee meetups, and so on.

5. *Description*. In this field, the user can provide a detailed description of the service they would like to offer or receive, giving the potential trading partner a clear view of the trade.

Once the proposal has been created and all of its fields have been set, the user sends it (in a transaction form) to the blockchain. After a short period of time, the proposal will appear at the trading pool that is visible to all users within the network and ValuED framework. Later on, those students who are interested in a certain proposal can contact the proposal's creator and obtain additional information. They can negotiate on when the tokens should be transferred, i.e., before or after delivering/receiving the service.

If the proposal is of the type "wanted", then the user, who wants to engage in the trade and provides a certain service, sends their account address (e.g., via email) to the proposal creator who finalises the published proposal by inserting the service provider's account address in the published proposal. Once the address is inserted, the proposal is finalised, and the number of tokens specified in the proposal is automatically transferred from the proposal creator's

account to the service provider's account. After the proposal completion, the proposal will no longer be activated and will not accept any new participants. Note that in the case where the proposal is of the type "wanted", the smart contract ensures that only the proposal creator can insert the service provider's address into the proposal.

On the other hand, if the proposal is of the type "offered", then the user who would like to receive a certain service finalises the published proposal by accepting the proposal. In this case, the specified number of tokens is transferred automatically from the service recipient's account to the service provider's account. In this case, the smart contract ensures that only the service recipient who has enough tokens can finalise a wanted proposal. Again, after the proposal completion, the proposal will not accept any new participants. After the transaction between initiator and responder has been confirmed and they have exchanged tokens for the service, they can leave a feedback score that ranges from -5 to 5, to indicate the satisfaction level of their transactions, where 5 and -5 refer to the highest and lowest satisfaction level respectively. This score will be used as the basis for calculating the overall reputation value of each student in the ValuED framework. Each user's score is publicly available to offer transparency. A good reputation can help to increase the probability of a successful transaction, which will lead to more revenue for students who provide good quality services. At any point in time, a reputation $r^{(S)}$ of a student S is computed as:

$$r^{(S)} = \sum_{i=1}^{n} f_i$$

where n is the total number of trades the student has been involved in and f_i is the score given to the student for the i-th trade by their counter-party who engaged in that trade. The source code of ValuED's smart contract and GUI is available in [6] and [7] respectively.

4.2.3. Receiving rewards for attending lectures

ValuED allows students to collect tokens each time they attend a lecture. To facilitate this, before each j-th lecture begins, the admin allocates a "unique random session code", z_j, to that lecture (the idea of allocating a unique code to each session is akin to the notion of the session code used in most well-known online learning management systems such as "moodle" [27]). This session code is valid only for a certain time period. Before the lecture starts, the admin stores the encryption of the code (i.e., $\mathcal{H}(z_j)$, where $\mathcal{H}$ is a cryptographic hash function) in the smart contract. The reason that this code is encrypted before it is stored in the smart contract is to prevent students from retrieving the code directly from the smart contract without attending the related lecture.

At the end of each lecture, the tutor provides the code, z_j, in plaintext to the students who have a fixed time window to insert the session code into the ValuED GUI. Once each student provides a code, say z' , to the ValuED GUI, the GUI encrypts the code (i.e., $\mathcal{H}(z')$) and sends the result to the ValuED smart contract. The smart contract compares the result with the encrypted value previously stored by the admin for the same session. If the two match (i.e., $\mathcal{H}(z_j) = \mathcal{H}(z')$), then the smart contract sends a fixed number of tokens to the student as a reward.

4.3. Using ValuED

Having clearly in mind the architecture and the implementation of the platform, we can describe how to run and interact with it. The first step is setting up its environment. Users need to hold an account in MetaMask and the ValuED cryptocurrency must be up and running in the blockchain network by the servers (or nodes). One of the servers may host the web application, that is the user interface. In practice, a user suffices to connect to the web application server with a standard web browser that allows interaction with MetaMask (e.g., the MetaMask plugin for Chrome).

Once users are identified in ValuED through their MetaMask account, they can use the web application to perform any of the available actions, from user management tasks to trading tasks, in accordance with their user type and permissions. For example, they can accept an existing proposal or make a new proposal, and they can attend lectures or tutorials to gain tokens. Figures 9 and 10 illustrate students' interaction with the platform to create proposals and claim tokens for attending in lectures respectively.

In general, there are four main phases involved in the ValuED platform: (1) account creation, (2) user management, (3) peer-to-peer trade, and (4) teaching environment enrichment.

4.3.1. Account Creation

Each user (regardless of their future role) installs a digital wallet software, e.g., MetaMask, and creates an account which comes with a unique account address that can be considered as the user's ID. Each user sends their ID to the manager. In the ValuED platform, each user is recognised by their ID.

4.3.2. User Management

The user management phase includes the following steps:

1. Upon deployment of the ValuED's smart contract by the manager, the manager's account address is automatically registered in the contract as "the manager".
2. After the deployment, the manager registers admin users in the contract, given the account addresses they have provided. The manager can register all admin users at once, or separately.
3. When students have confirmed their course enrollment or wish to participate in the peer-to-peer trade framework, they send their student numbers along with their account addresses to the manager or the (course) admin.
4. The manager or admin performs a background check (out of the bound) to validate the student numbers. It registers the valid student numbers and account addresses in the contract, so they will become verified/authorised users of the platform. The user management process is the same before setup for both peer-to-peer trade and teaching environment enrichment.

4.3.3. Peer-to-peer trade

In this phase, students create proposals and trade with each other. We have already described this phase in detail in Section 4.2.

4.3.4. Teaching environment enrichment

The creation of tokens in the platform (in addition to the initial token distribution phase) relies on the teaching environment enrichment process, which is a way for students to collect tokens without the need to provide any services. In particular, as stated in Section 4.2.3, students can collect tokens by attending lectures. Moreover, the tutor may require students to carry out course-related activities prior, during or after each lecture session. Such activities may include, conducting research, presenting research in each session, and peer-reviewing fellow students' research (reports). In these cases, the tutor can award active students through our platform, in at least two ways; namely, (a) by sending tokens to those students who completed their tasks and met the specified requirements, or (b) by creating academic-related proposals and offering tokens to those students who wish to participate in the trade and meet the requirements specified in the proposals.

4.4. ValuED's cost analysis

In this section, we report the cost of deploying and using our platform. Since we do not use any computationally expensive primitives at the user interface, and do not require users to run any local software other than the wallet software which is very lightweight, we focus on the cost of interacting with the blockchain. In general, the cost and time for reading data from the blockchain are negligible. When users write data into the blockchain, there would be some Gas fee that includes the cost of deploying the ValuED smart contract and invoking this smart contract's functions.

In Table 1, we summarize the ValuED smart contract's deployment cost and the cost of invoking the primary functions of this smart contract. As the table indicates, deploying the ValuED smart contract to the blockchain imposes the highest cost, i.e., 0.003121 Ether or $10.11, as the entire contract's code needs to be stored on the blockchain. Among the functions, invoking the function for making a proposal imposes the highest cost, i.e., 0.000169 Ether or $0.55. The reason for this high cost is that this function requires (a) the party who calls it to send a large size data and (b) the smart contract to update several variables it holds. Fig. 7 presents a square pie chart depicting the percentage of the cost that each of the main seven operations imposes.

Figure 5: Interaction flow of peer-to-peer trade.

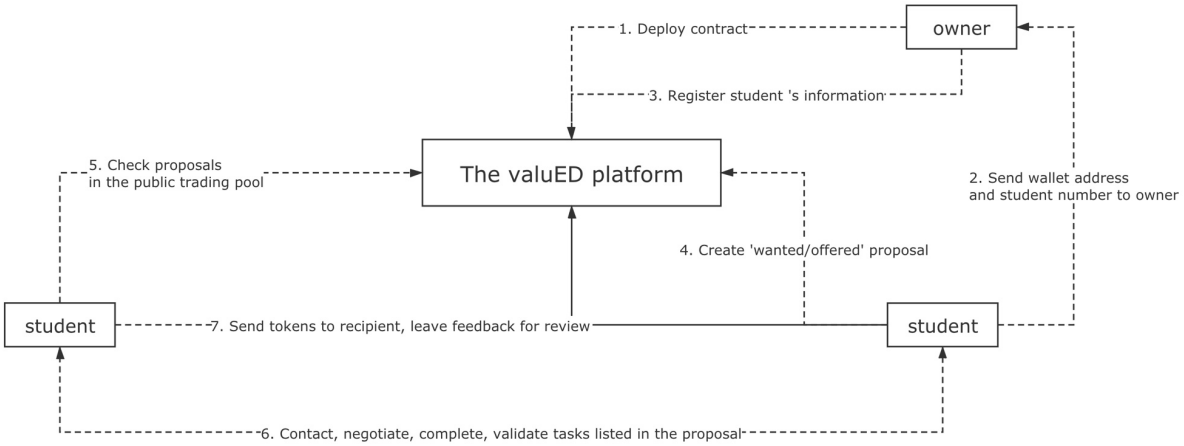
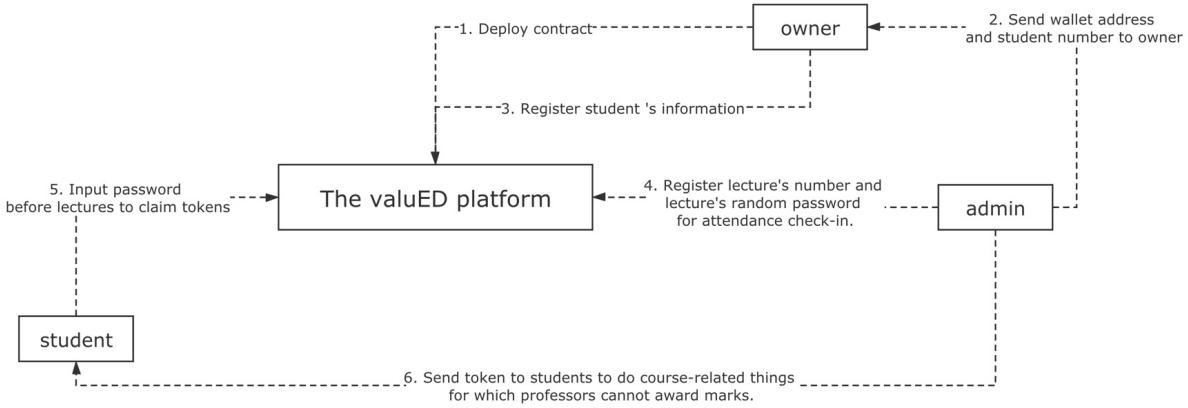


Figure 6: Interaction flow of teaching environment enrichment.



5. Conclusions and future work

We designed and implemented a blockchain-based trading platform, ValuED, with the goal of engaging students and others within a large university. We wished to increase the agency available to students and others and to enrich the educational and social milieu of the university. The interface allows participants to buy and sell whatever goods and services they find on the platform or have placed there. The platform is computationally more elaborate than the few previous comparable systems that we were able to find in the literature, with the exception of EduCTX. We have made the platform's source code available for anyone to use or develop.

What did we learn from this exploratory study? How might future versions develop?

Table 1
Concrete cost of ValuED smart contract deployment and invocations of its main functions.

Operation	Cost in Ether	Cost in US Dollar
Deploy contract	0.003121	10.11
Register student number	0.000043	0.14
Register student address	0.000071	0.23
Set current lecture number	0.000042	0.14
Allocate account for professor	0.000029	0.09
Register lecture ID	0.000044	0.14
Make proposal	0.000169	0.55

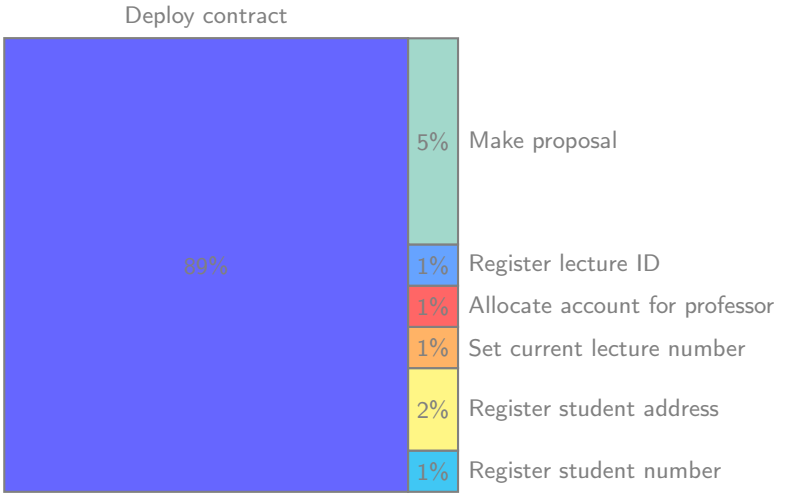


Figure 7: Cost of ValuED smart contract deployment and invocations of its main functions (in percent).

Our first conclusion was metatheoretic. One learns by doing. Most of the problems and possibilities we encountered were not initially envisaged when we began this project *ex nihilo*. Our most important conclusion came from a pilot launch of the scheme, not reported here, in a first year Cognitive Science class or some 200 students. Students typically feel that they have little if any free time which is not part of their social life or "part of the exam". To succeed, any such scheme must be able to attract and retain the engagement of students, with little or no effort on their part. This conclusion has a number of implications. Our initial implementation was necessarily on the University network, accessed only from desktop or laptop machines connected to that network. Access needs to be mobile and needs to draw on the existing electronic skills and activities of the students: access has to be by means of a smartphone.

Closely connected with this issue is our conclusion that the interface needs to be not just *intuitive* but *proactive*. By intuitive we mean that operating the user interface has to have as small a learning curve as possible. By proactive we mean that the scheme has to consistently "nudge" participants to notice what is going on in the scheme and to respond. The scheme has to become part of the social media habitually used by the participants.

There are several means by which participation might be encouraged. The goods and services available may be intrinsically valuable. For instance, the offer of lecture notes for a particular course, or a social trade of coffee for cake, involving a chat. One way of *forcing* induction into the scheme is to associate it with lecture and tutorial attendance. Academics typically cannot give students marks for attendance, and attendance monitoring is tedious and not really feasible for very large classes. However, attendance is useful information and academics can top up a student's trading currency in return for attendance as recorded by the student using their smartphone to photograph a "Quick Response" (QR) code briefly displayed at the beginning of the lecture.

It became clear to us that the boundaries for participation are porous. Students may be inducted into the scheme in their first year. In degrees such as the Scottish undergraduate degree, students typically take courses outside their main subject area in their first two years, meaning that students attending a Cognitive Science class may be from degrees as

diverse as Archaeology and Zoology. Carrying their engagement with the scheme into their second year presents them with opportunities such as selling used textbooks to incoming firstyears. By the same measure, they may continue with the scheme *beyond* their final year. In this way the scheme may present a way of retaining the engagement of alumni with the university and with each other. The participation of postgraduate students, postdocs, academic staff, and others employed by the university might also be considered.

Another aspect of the porosity of the scheme is the potential involvement of the university's own internal economy of catering, accommodation, volunteering, and so on, with students earning credits for volunteering at open days and perhaps spending them in the university's snack bars. This involvement with the university's own economy has to be contrasted with the fact that all students are necessarily part of the (non-university) local economy. We came to see the trading scheme as essentially open-ended.

We noted above that the "student experience" is now widely seen as part of the value of higher education. The scheme we have described could evolve into a way of quantifying this experience. A student presenting themselves to a prospective employer might say "I have an honours degree in X, with a mark of 70 for my final dissertation project, and I was in the highest decile of my year in terms of my ValuED profile. The latter means I engaged with and contributed to university life and work at a very high level all the way through my degree." ValuED is one way of quantifying and comparing the rather nebulous soft skills and motivational factors valued by most employers.

As we considered the possibilities of the trading scheme, the ethical implications became clearer. We raise these in the form of questions, at this stage. How do we monitor exactly what goods and services students and others might be offering? What ethical oversight and intervention is both appropriate and possible? How do we deal with students who succeed in *gaming the system* in one way or another (particularly possible in its early stages)? Perhaps we reward them and co-opt them into the enterprise? How do we ensure the security and privacy of the participants' information? What sort of choice of level of engagement do we allow? There will be students who "just want to study".

Finally, we learned that in the world of cryptocurrency even a project as innocuous as ours and as distant from speculative finance attracted an anonymous hacking attack from outside. Security and privacy are key concerns.

Acknowledgments

Aydin Abadi was supported in part by REPHRAIN and OxChain projects, under grant numbers EP/V011189/1 and EP/N028198/1 respectively. The author was at the University of Edinburgh during the initial design and implementation of the ValuED platform. We would like to thank Chris Speed for his precious feedback and for providing further funds from "Edinburgh Future's Institute". We also would like to thank Aggelos Kiayias for providing valuable feedback and resources. We would like to thank Dave Cochran who assisted us with the GUI development.

References

[1] Abadi, A., 2022. Smarter Data Availability Checks in the Cloud: Proof of Storage via Blockchain. Handbook of Research on Digital Transformation, Industry Use Cases, and the Impact of Disruptive Technologies, IGI Global.

[2] Abadi, A., Ciampi, M., Kiayias, A., Zikas, V., 2020. Timed signatures and zero-knowledge proofs - timestamping in the blockchain era-, in: Applied Cryptography and Network Security - 18th International ACNS, Springer.

[3] Abadi, A., Kiayias, A., 2021. Multi-instance publicly verifiable time-lock puzzle and its applications, in: 25th Financial Cryptography and Data Security, Springer.

[4] Andrade, P., Law, E.L., 2021. Improving student experience and learning performance with traditional instructional methods and new digital media, in: Human-Computer Interaction - INTERACT 2021 - 18th IFIP TC 13 International Conference, Bari, Italy, August 30 - September 3, 2021, Proceedings, Part III, Springer.

[5] Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L., 2014. Secure multiparty computations on bitcoin, IEEE Computer Society.

[6] Anonymous, 2021a. The smart contract's source code of the ValuED platform. https://github.com/anonymousValuED/ValuED/blob/main/ValuEd-Smart-Contract.sol.

[7] Anonymous, 2021b. The user interface's source code of the ValuED platform. https://github.com/anonymousValuED/ValuED/blob/main/UI/index.html.

[8] Ansari, M.R., Navratan, N., Umamaheswari, K., 2021. A study of awarding student achievement using blockchain. Linguistics and Culture Review 5, 823–836.

[9] Avdoshin, S., Pesotskaya, E., 2020. Blockchain in charity: Platform for tracking donations, in: Proceedings of the Future Technologies Conference, Springer. pp. 689–701.

[10] Bitcoin wiki, 2019. Delegated proof of stake. https://en.bitcoin.it/wiki/Delegated_proof_of_stake. Accessed: 2021-10-09.

[11] CamLETS, 1993. Local Exchange Trading System. https://www.camlets.org.uk/. Accessed: 2021-10-15.

[12] Cardano, 2021. Documentation for the Cardano ecosystem. Technical Report. https://docs.cardano.org/.
[13] Chen, H.S., Jarrell, J.T., Carpenter, K.A., Cohen, D.S., Huang, X., 2019. Blockchain in healthcare: a patient-centered model. Biomedical journal of scientific & technical research .
[14] Dannen, C., 2017. Solidity programming, in: Introducing Ethereum and Solidity. Springer, pp. 69–88.
[15] Estevam, G., Palma, L.M., Silva, L.R., Martina, J.E., Vigil, M., 2021. Accurate and decentralized timestamping using smart contracts on the ethereum blockchain. Inf. Process. Manag .
[16] Francisco, K., Swanson, D., 2018. The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency. Logistics 2, 2.
[17] Hölbl, M., Kompara, M., Kamišalić, A., Nemec Zlatolas, L., 2018. A systematic review of the use of blockchain in healthcare. Symmetry .
[18] Humphrey, R., 2006. Pulling structured inequality into higher education: the impact of part-time working on english university students. Higher Education Quarterly 60, 270–286.
[19] Jongbloed, B., 2006. Strengthening consumer choice in higher education, in: Cost-Sharing and Accessibility in Higher Education: A Fairer Deal?. Springer, pp. 19–50.
[20] Kaistinen, A., Riisiö, J., Arpalahti, J., Yrjölä, J., Siren, S., Koivunen, V., Ristimäki, V., 2015. Open City App. https://github.com/City-of-Helsinki/open-city-app. Accessed: 2021-10-17.
[21] Kalmi, T., 2018. Comparison of Blockchain-based Technologies for Implementing Community Currencies. Master's thesis. Aalto University. School of Science. URL: http://urn.fi/URN:NBN:fi:aalto-201811135739.
[22] Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C., 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, in: 2016 IEEE symposium on security and privacy (SP), IEEE. pp. 839–858.
[23] Llamas-Nistal, M., Mikic-Fonte, F.A., 2014. Generating oer by recording lectures: A case study. IEEE Transactions on education 57, 220–228.
[24] Longo, F., Nicoletti, L., Padovano, A., d'Atri, G., Forte, M., 2019. Blockchain-enabled supply chain: An experimental study. Computers & Industrial Engineering .
[25] Mann, S.J., 2001. Alternative perspectives on the student experience: Alienation and engagement. Studies in higher education .
[26] Montgomery, C., 2010. Understanding the international student experience. Macmillan International Higher Education.
[27] moodle, 2002. moodle website. https://moodle.org/. Accessed: 2021-11-27.
[28] Nakamoto, S., 2009. Bitcoin: A peer-to-peer electronic cash system .
[29] Odhabi, H., Nicks-McCaleb, L., 2011. Video recording lectures: Student and professor perspectives. British Journal of Educational Technology 42, 327–336.
[30] Peterson, P., Gaucher, M., 2021. Zcash Documentation. Technical Report. https://buildmedia.readthedocs.org/media/pdf/zcash/latest/zcash.pdf.
[31] Remenick, L., Bergman, M., 2021. Support for working students: Considerations for higher education institutions. The Journal of Continuing Higher Education 69, 34–45.
[32] Sabri, D., 2011. What's wrong with 'the student experience'? Discourse: Studies in the cultural politics of education 32, 657–667.
[33] Stefansson, G., Lentin, J., 2017. From smileys to smileycoins: Using a cryptocurrency in education. Ledger URL: https://ledgerjournal.org/ojs/index.php/ledger/article/view/103.
[34] Top Hat, 2009. Top Hat Platform. https://tophat.com/. Accessed: 2021-10-16.
[35] Trotter, E., Roberts, C.A., 2006. Enhancing the early student experience. Higher Education Research & Development .
[36] Trotter, L., Harding, M., Shaw, P., Davies, N., Elsden, C., Speed, C., Vines, J., Abadi, A., Hallwright, J., 2020. Smart donations: Event-driven conditional donations using smart contracts on the blockchain, in: OzCHI '20: 32nd Australian Conference on Human-Computer-Interaction, Sydney, NSW, Australia, 2-4 December, 2020, ACM.
[37] Turkanovic, M., Hölbl, M., Kosic, K., Hericko, M., Kamisalic, A., 2018. Eductx: A blockchain-based higher education credit platform. IEEE Access .
[38] Tutor-web, . Tutor-web platform. https://tutor-web.net/. Accessed: 2021-10-18.
[39] Vivekananthamoorthy, N., et al., 2019. Driving Success in e-Learning Portals: Piazza, a Multi-Faculty Collaborative Model. International Journal of Web-Based Learning and Teaching Technologies (IJWLTT) 14, 31–49.
[40] Wood, G., et al., 2014. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper .
[41] Young, J.R., 2008. The lectures are recorded, so why go to class. Chronicle of Higher Education 54, A1.
[42] Zyskind, G., Nathan, O., Pentland, A., 2015. Enigma: Decentralized computation platform with guaranteed privacy. arXiv preprint arXiv:1506.03471 .

A. Architectural design of ValuED

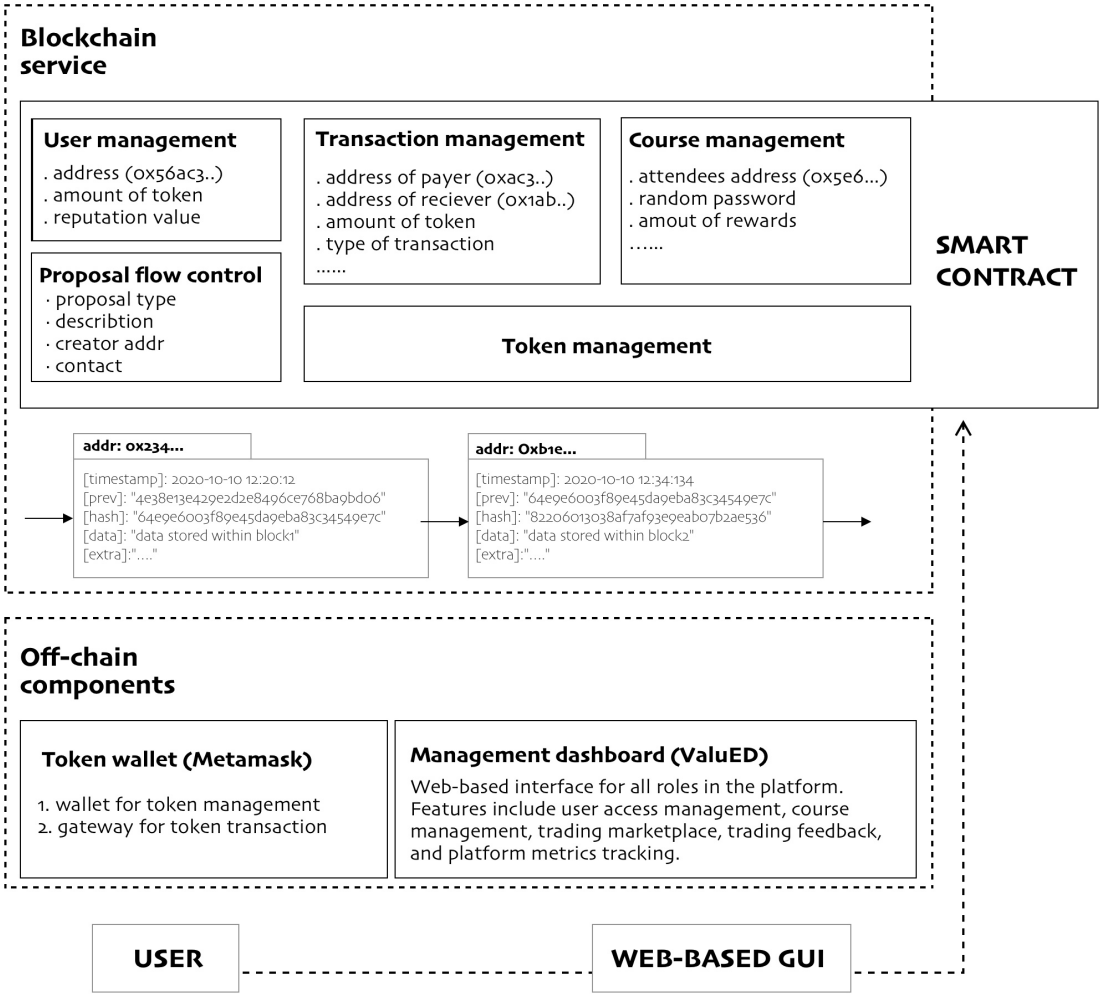
The architecture of our platform comprises three layers; namely, (a) application, (b) communicating, and (c) service layers. Below, we briefly explain each layer. The application layer is a web-based Graphical User Interface (GUI) which enables users to interact with the smart contract that has been deployed to the Ethereum (testnet) blockchain. This layer allows users to interact with the smart contract to change the data it maintains (e.g., token balance) or to invoke certain functions of the contract. Our GUI design ensures that the users can easily interact with the smart contract without having to know its code and the logic behind its design. Nevertheless, it requires users to install and use a digital wallet, e.g., MetaMask, in order to create IDs, and digitally sign transactions they send to the blockchain. Specifically, this lets users independently create unique public keys (or IDs) that are registered in the contract. The contract recognizes each

user by its ID and assigns a specific role to each ID holder. The users can install the digital wallet to create a secret (signing) key with which they can sign and authorise each transaction they send to the contract.

The communicating layer (a.k.a., the peer-to-peer network) is a middle layer for connecting blockchain service to web-based development protocols. It is responsible for inter-node communication and takes care of node discovery, transactions, and block propagation. Our platform does not require any changes to this layer. Furthermore, the service layer is a run-time environment for smart contracts in Ethereum. It involves the blockchain's security (e.g., block validation and signature verification) and consensus (e.g., proof of work or proof of stake) protocols. We can broadly categorize the service layer into two modules: smart contract and ledger (data storage). The smart contract module is where the desirable computational logic is encoded and it maintains users' token balances and ensures only authorized users can change certain variables or execute specific functions encoded in the smart contract. Moreover, the distributed ledger-based data storage module ensures the immutability and security of users' data.

Putting it all together, the desirable functions are encoded at the service level. They can be accessed by users from the application layer (e.g., GUI) while the communicating layer acts as middleware that connects the application layer to the service layer.

Figure 8: ValuED system architecture



B. ValuED implementation

B.1. Underlying procedure

Technically, the key process of our platform implementation is to define and deploy smart contracts on the blockchain infrastructure.

B.1.1. Design of functionalities

As we stated in section 4.2, interactions between roles were defined based on the systemic design of the ValuED smart contracts. Defining the smart contract includes two parts: (1) the declaration of entities (proposal, transaction) and configuration items of the platform, (2) the declaration of functions to process transactions and proposals. The following are the explanations of the main activities in our proposed platform.
Set-up: Several attributes were defined to describe the rules and others were used to represent states of our proposed platform. The difference between the two variables is that the properties used to define the rule cannot be changed by calling a method once deployed, whereas the value of the other variable is capable of changing the state after the contract is deployed. For example, attributes that define the rules include:

- `owner`: variable that stores the account address of the owner.
- `max_score`, `min_score`: variables to define the maximum and minimum score that can be given for each transaction.
- `lecture_tokens`: variable to define the fixed number of tokens that can be claimed as a reward.
- `Proposal`: structure of a proposal. See Listing 3.
- `Transaction`: structure of a transaction. See Listing 3.

The initialization process of the ValuED contract is summarized in Algorithm 1.

Algorithm 1: Declarations and initialization.

Declare static variables. `lecture_tokens`, `Proposal`, ...

Declare mapping relationships for state management. `valid_admins[]`, `proposals[]`, ...

Declare functions for business logic.

Initiate owner to the address of contract deployer.

Initiate `valid_admins[owner]` to true. (The deployer can be admin as well)

Initiate `max_score`, `min_score` and `lecture_tokens` to 5.

Listing 3: Structure defining a proposal and a transaction.

```
struct Proposal {
  uint numOf_tokens;          // the price of the proposal
  address creator_address;    // the creator's address
  string creator_emailAdress; // contact information
  string reason;              // description of the proposal
  uint proposal_ID;           // unique id
  bool active;                // status
}

struct Transaction {
  address sender;
  address reciever;
  string reason;              // description of the reason
  int TokenSender_feedback;   // feedback provided by the sender of tokens.
  int TokenReciever_feedback; // feedback provided by the receiver of tokens.
  uint transaction_ID;        // unique transaction id
  uint numOf_tokens;          // number of tokens sent in this transaction.
  string creation_time;       // timestamp
}
```

User registration: For the first time, when the ValuED smart contract is deployed (or sent to the blockchain), the address of the party who deploys the contract (a.k.a owner) is registered automatically in the smart contract. Later, this party can register other addresses and allocate the admin role to each of these addresses. The admins have the ability to register other addresses and allocate the student role to these addresses. The detailed process is as shown in Algorithm 2.

Algorithm 2: Managing users (admin and student).

```
Initiate arrays to maintain valid admin, valid student number, valid student address and student token balance.
 (valid_admins[], valid_student_num[], valid_student_addr[], student_token_balance[])
Initiate structure of Pair to maintain the state of valid student number. ( validStudent_num, token_assigned)
function add_admin (new_admin):
    if caller's address is equal to the owner then
        valid_admins[new_admin] ← true
function remove_admin (admin):
    if caller's address is equal to the owner then
        valid_admins[admin] ← false
function register_std_num (std_num):
    if caller's address is one of the valid admins then
        valid_admins[std_num].validStudent_num ← true
function register_std_addr (std_addr, std_num):
    if caller's address is one of the valid admins then
        if student number is valid and have not been registered then
            valid_student_addr[std_addr] ← true /* mark student number is registered */
            valid_student_num[std_num].token_assigned ← true /* allocate 10 tokens to the registered
               student */
            student_token_balance[std_addr] ← 10
```

Proposal creation. In our platform, each proposal is represented by an instantiating of the proposal template. When a proposal is created and successfully completed, the transaction details are recorded in the blockchain, this ultimately affects the token balance of the traders in the ledger. The process is illustrated in Algorithm 3.
Course management process: The teaching environment enrichment phase mainly involves the interactions between tutors and students and rewarding students with tokens. A previous study has proved, token-based incentives mechanisms can increase student engagement in courses. The incentive mechanism of our platform is shown in Algorithm 4. For ease of accessibility, we also present the ValuED platform's smart contract source code in Fig. 4.

C. ValuED's Graphical User Interface (GUI)

The ValuED GUI provides various functions to different role holders. To work with this GUI, users need to first create and set up an account via the digital wallet (i.e., MetaMask) and connect their wallet to the blockchain. The first party who should use the GUI is the owner which (as previously stated) registers a set of admins' addresses via this GUI. Next, each registered admin can access the GUI and uses a set of functions to fulfill its responsibility, as shown in Fig. 11. The GUI has two main parts.

1. *Page Header*: It illustrates the account address and role of the party who accesses the ValuED GUI (e.g., "ADMIN" in Fig. 11). The GUI automatically fetches the user's address (from the wallet), compares it with the registered address on the ValuED smart contract, and if it finds any match, it displays the corresponding role and address on the page header.
2. *Tabs*: They allow a user to access different content and functions of the ValuED GUI. This GUI offers four tabs. Below, we briefly explain each of them.
   (a) *Admin*: under this tab, there are the functions (e.g., register students' number or address, distribute tokens) that are available to the registered admin. Fig. 11 presents a screenshot of the Admin tab's content.

Algorithm 3: Proposal process.

```
function make_proposal (num_of_tokens, reason, email_address):
    if caller's address is one of the valid students then
        if caller's balance is not less than required number of tokens then
            proposal.tokens ← num_of_tokens
            proposal.reason ← reason
            proposal.email_address ← email_address
            proposal.creator ← sender // caller's address
            total_num_proposals ← total_num_proposals + 1
            proposal.id ← total_num_proposals
            proposal.active ← true
            proposals[total_num_proposals] ← proposal
function send_token (amount, recipient, reason, time, offer_ID):
    if sender's address is one of the valid students then
        if recipient's address is one of the valid students then
            if amount > 0 ∧ sender.balance ≥ amount then
                sender.balance ← sender.balance − amount
                recipient.balance ← recipient.balance + amount
                Store transaction's details in transactions
function leave_feedback (transaction_ID, score):
    if min_score ≤ score ∧ score ≤ max_score then
        if can_leave_feedback (sender, transaction_ID) then
            Set receiver's reputation value plus score
```

Algorithm 4: Course management process.

```
function set_currentlecture_number (num):
    if caller's address is one of the valid admins then
        Set proposal_'s value with numOf_tokens_, reason_, email_address // WRONG?
function register_lecture (lecture_number, password):
    if caller's address is one of the valid admins then
        hash_lectureID[lecture_number] ← bytes2(keccak256(bytes(password)))
function claim_token (password):
    if caller's address is one of the valid students then
        if password is correct then
            if caller has not claim token for this lecture yet then
                Set caller's balance to caller's balance plus lecture tokens
```

(b) *Proposals*: under this tab, a list of all proposals made so far is provided. Fig. 12 presents a screenshot of this tab's content.

(c) *About*: under this tab, a brief description of the ValuED platform and how it can be used are provided.

(d) *Data*: under this tab, a summary and visualization of all trades and transactions made via this platform are provided. Fig. 13 presents a screenshot of this tab's content.

The ValuED GUI also provides student users a page with various options. The student page has also two main parts (as shown in Fig. 2); namely, (a) the page header that indicates the user's status as "STUDENT" and its account address (if the student's address and detail have been registered in the ValuED smart contract), and (b) the Trading, About, and Data tabs.

Figure 9: Screenshot of the admin page.

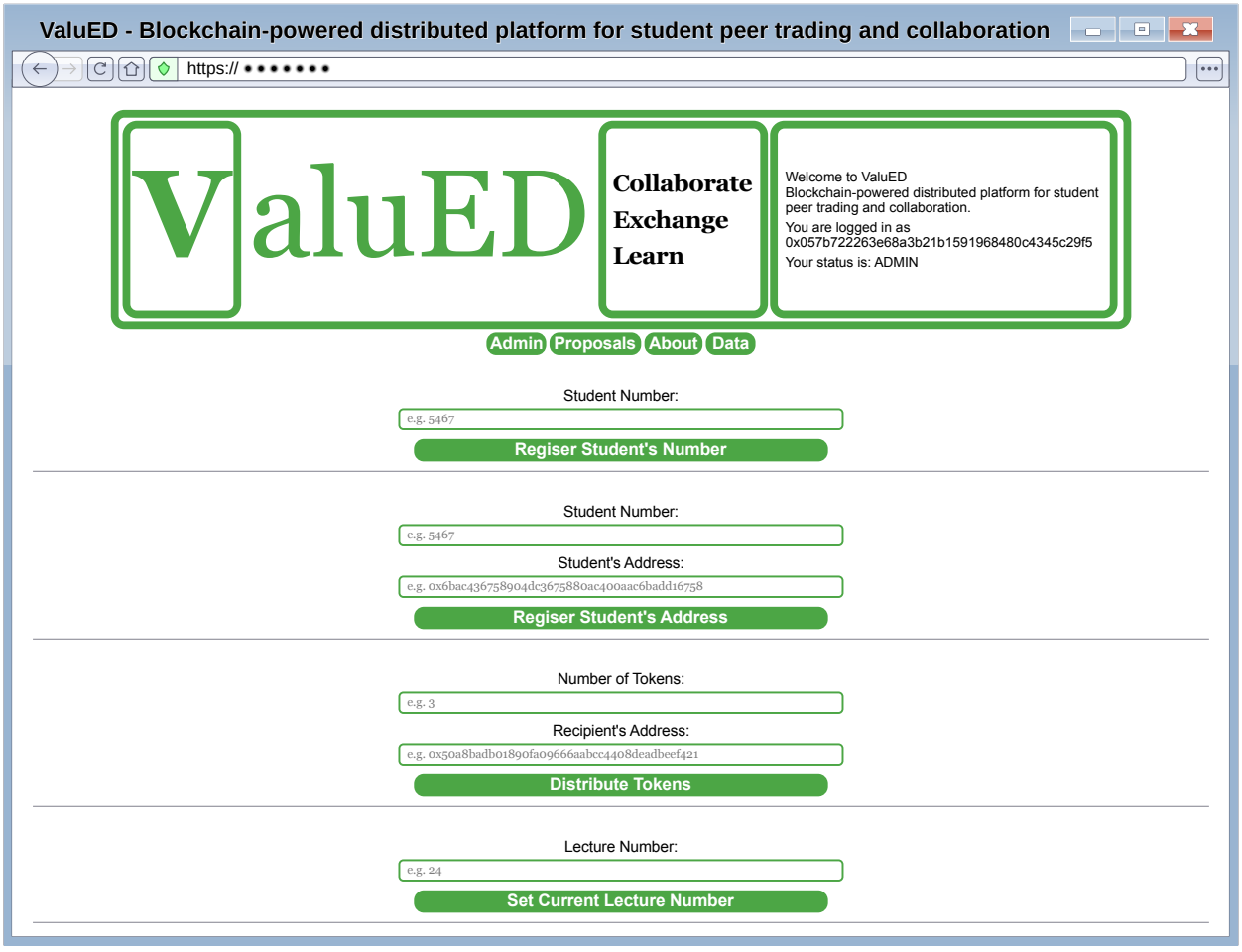


Figure 10: Screenshot of the Proposals page provided to an admin user.

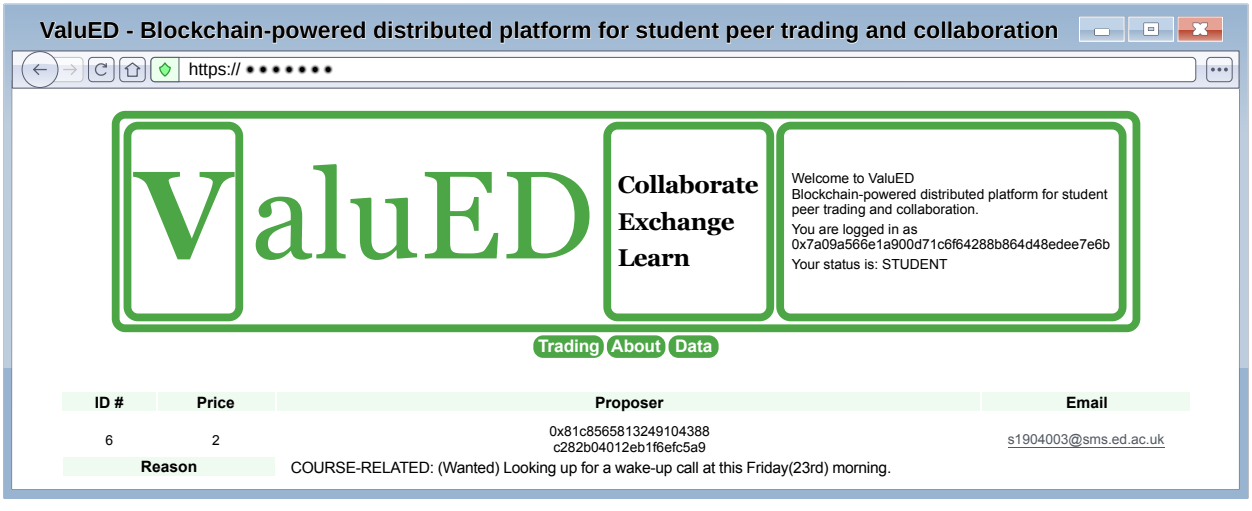
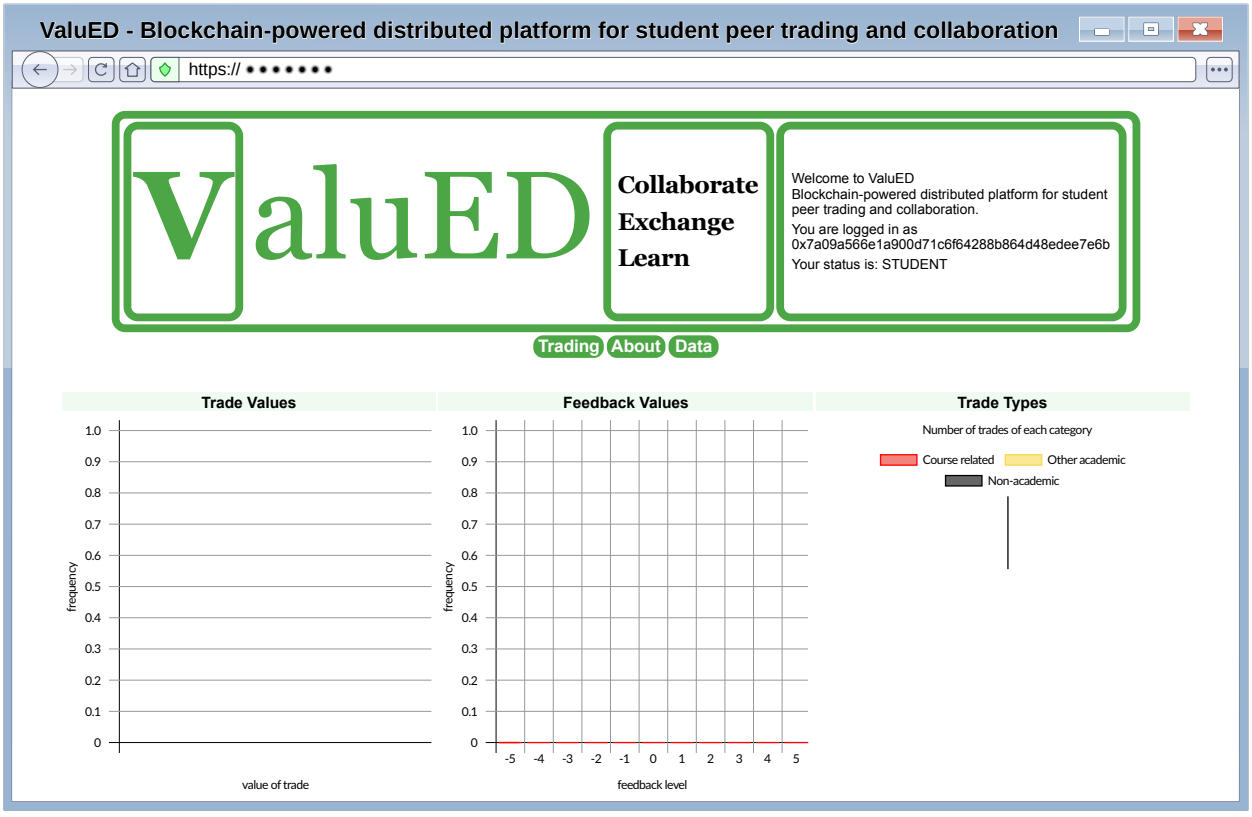


Figure 11: Screenshot of the data page.



As it is evident in Fig. 2, under the trading tab, a student can see information about its account on this platform, such as its token balance, total token traded, and its reputation. The ValuED GUI allows a student to view the tradings history using different filters. Moreover, it can use this page to create a new proposal, by clicking on the "+" symbol on the page, filling in the form provided to this student, and pressing the submit button to publish its proposal (to the blockchain).

Listing 4: ValuED smart contract in Solidity

```
// SPDX-License-Identifier: GPL 3.0
pragma solidity ^0.5.0;

/**
 * @author Aydin Abadi
 * @title ValuED contract
 */
contract ValuED {
    /* _________
     * Constants
     */
    int public constant NO_FEEDBACK = -10;                /// Constant outside range to signify no feedback provided
    int public constant MAX_SCORE = 5;                    /// Maximum score (feedback) allowed
    int public constant MIN_SCORE = -5;                   /// Minimum score (feedback) allowed
    uint public constant LECTURE_TOKENS = 5;              /// Amount of tokens that can be claimed is fixed
    uint public constant REGISTRATION_TOKENS = 10;        /// Amount of (welcome) tokens given (to students) at registration

    /* _______________
     * Management/Info
     */
    address public manager;                               /// The owner, deployer and manager of the contract
    mapping (address => bool) public validStudent;        /// Valid students (registered)
    mapping (address => bool) public validAdmin;          /// Valid administrators
    uint public currentLectureNumber;                     /// Current lecture number
    mapping (uint => StudentStatus) public studentStatus; /// Index: student numbers
    mapping (address => uint) public studentBalance;      /// Student's balance
    mapping (uint => bytes2) public hashLectureID;        /// lecture number => hash(lecture ID).
    mapping (address => uint) public claimed;             /// Binds a student's address to a lecture number to signify they claimed the related tokens
    mapping (uint => uint) public lectureParticipants;    /// Index: lecture number

    /* ___________________
     * Peer-to-peer trades
     */
    uint public proposalsCount;                           /// Last proposal ID
    mapping (uint => Proposal) public proposals;          /// Proposals

    uint public transactionsCount;                        /// Last transaction ID
    mapping (uint => Transaction) public transactions;    /// Transactions

    mapping (address => int) public reputations;          /// Users' current scores
    mapping (address => uint) public tradedTokens;        /// Total tokens traded by user


    /**
     * The status of students determines their registered status in the
     * current lecture.
     */
    struct StudentStatus {
        bool enrolled;   /// If the student is enrolled
        bool registered; /// It the student has been assigned tokens already
    }

    /**
     * The status of what student claimed relating to what lecture.
     */
    struct ClaimedStatus {
        address student;    /// Student that claimed tokens for a lecture
        uint lectureNumber; /// Lecture number of reference
    }

    /**
     * Valid users can propose any trades they wish and submit their proposal
     * that has the following structure.
     */
    struct Proposal {
        uint tokens;     /// Tokens required for this proposal (sent or received)
        address creator; /// Who offers the proposal
        string email;    /// Used to exchange extra information
        string reason;   /// Description of the proposal
        uint id;         /// The ID of the proposal
        bool active;     /// An active proposal is one where tokens have not been transferred yet
    }

```

```
74      /**
75       * A transaction is created when tokens are transferred from an users
76       * to another.
77       */
78      struct Transaction{
79          address sender;        /// Who sends the tokens
80          address receiver;      /// Who receives the tokens
81          string reason;         /// Description of the transaction
82          int senderFeedback;    /// Feedback provided by the sender
83          int receiverFeedback; /// Feedback provided by the receiver
84          uint id;               /// The ID of the transaction
85          uint tokens;           /// Tokens transferred in this transaction
86          string creationTime;   /// Time of creation
87      }
88
89      /**
90       * Contract constructor.
91       * The deployer (manager) will be administrator too.
92       *
93       * @param admin administrator to add (apart from the manager)
94       */
95      constructor(address admin) public {
96          manager = msg.sender;
97          validAdmin[admin] = true;
98          validAdmin[msg.sender] = true;
99      }
100
101     /**
102      * Only an admin will be allowed.
103      */
104     modifier onlyAdmin() {
105         require(validAdmin[msg.sender] == true);
106         _;
107     }
108
109     /**
110      * Only the manager will be allowed.
111      */
112     modifier onlyManager() {
113         require(msg.sender == manager);
114         _;
115     }
116
117     /**
118      * Add an administrator.
119      *
120      * @param admin administrator to add
121      */
122     function addAdmin(address admin) external onlyManager {
123         validAdmin[admin] = true;
124     }
125
126     /**
127      * Delete an administrator.
128      *
129      * @param admin administrator to delete
130      */
131     function delAdmin(address admin) external onlyManager {
132         validAdmin[admin] = false;
133     }
134
135     /**
136      * Allows a valid admin to send some tokens to students.
137      *
138      * @param student student whose balance will be increased
139      * @param tokens tokens given to the student
140      */
141     function distributeToken(address student, uint tokens) external onlyAdmin {
142         require(validStudent[student] == true);
143         studentBalance[student] += tokens;
144     }
145
146     /**
147      * This function is called when a list of students enrolled for the course
148      * is finalised.
149      *
150      * @param studentNumber student number
151      */
152     function enrollStudent(uint studentNumber) external onlyAdmin {
```

```
153         studentStatus[studentNumber].enrolled = true;
154     }
155
156     /**
157      * Binds a (invalid) student to a student number that has been previously
158      * enrolled and was not registered before.
159      *
160      * @param student student to register
161      * @param studentNumber student number assigned for current registration
162      */
163     function registerStudent(
164         address student,
165         uint studentNumber
166     )
167         external onlyAdmin
168     {
169         require(studentStatus[studentNumber].enrolled == true);
170         require(studentStatus[studentNumber].registered == false);
171         studentStatus[studentNumber].registered = true;
172
173         validStudent[student] = true;
174         studentBalance[student] = REGISTRATION_TOKENS;
175     }
176
177     /**
178      * Register a lecture identified by a string to the given lecture number.
179      *
180      * @param lectureNumber the lecture number
181      * @param lecture the string identifying the lecture
182      */
183     function registerLecture(
184         uint lectureNumber,
185         string calldata lecture
186     )
187         external onlyAdmin
188     {
189         hashLectureID[lectureNumber] = bytes2(keccak256(bytes(lecture)));
190     }
191
192     /**
193      * Set the current lecture/course number.
194      *
195      * @param lectureNumber the lecture number
196      */
197     function setCurrentLectureNumber(uint lectureNumber) external onlyAdmin {
198         currentLectureNumber = lectureNumber;
199     }
200
201     /**
202      * This function allows a student to claim a fixed number of tokens
203      * (LECTURE_TOKENS), if it could prove its attentance in a lecture
204      * (e.g. by uploading a QR code in the UI). If approved (in the UI), then
205      * the UI calls this function.
206      *
207      * @param lecture the lecture (ID string) related with the token claim
208      */
209     function claimToken(string calldata lecture) external {
210         require(validStudent[msg.sender] == true);
211         require(
212             hashLectureID[currentLectureNumber]
213             == bytes2(keccak256(bytes(lecture)))
214         );
215
216         /*
217          * Ensures the student has not already claimed any tokens for this
218          * lecture yet.
219          * TODO future?: not enough if current lecture number is set to a
220          * previous one.
221          * - admins are currently assumed to behave honestly)
222          * - currentLectureNumber is expected to change in weeks (>= 1 week)
223          */
224         require(claimed[msg.sender] != currentLectureNumber);
225
226         claimed[msg.sender] = currentLectureNumber;
227         studentBalance[msg.sender] += LECTURE_TOKENS;
228         lectureParticipants[currentLectureNumber]++;
229     }
230
231     /**
```

```
     * In the UI, each student should be able to see a list of active offers
     * he/she has made. This allows the student to fetch specific offer ID
     * used in sendTokens.
     * This function allows a student to post an offer on the UI. It can offer
     * to engage in an actitivy and specify how many tokens it is willing to
     * send or recieve.
     *
     * @param tokens the price of the proposal
     * @param reason the reason describing the proposal
     * @param email email to send tokens later if public keys are sent too
     * known
     */
    function makeProposal(
        uint tokens,
        string calldata reason,
        string calldata email
    )
        external
    {
        require(validStudent[msg.sender] == true, "Not a valid sender");
        require(studentBalance[msg.sender] >= tokens, "Not enough token");

        Proposal memory proposal;
        proposalsCount++;
        proposal.tokens = tokens;
        proposal.creator = msg.sender;
        proposal.email = email;
        proposal.reason = reason;
        proposal.id = proposalsCount;
        proposal.active = true;
        proposals[proposalsCount] = proposal;
    }

    /**
     * Send a token related to a proposal.
     *
     * @param tokens the amount of tokens to send
     * @param receiver the address to send tokens to
     * @param reason the description for the transaction
     * @param time the time for the transaction
     * @param proposalID the proposal ID that contains the (active) offer
     */
    function sendTokens(
        uint tokens,
        address receiver,
        string calldata reason,
        string calldata time,
        uint proposalID
    )
        external
    {
        require(msg.sender != receiver);
        require(validStudent[msg.sender] == true, "Not a valid sender");
        require(validStudent[receiver] == true, "Not a valid recipient");
        require(
            studentBalance[msg.sender] >= tokens,
            "Not enough tokens"
        );
        require(proposals[proposalID].active == true, "Not an active offer");
        require(tokens > 0);

        /*
         * Either the token recipient or the token sender should be in the
         * creator of the offer_ID
         */
        require(
            msg.sender == proposals[proposalID].creator
            || receiver == proposals[proposalID].creator
        );

        // Only active offers should be desplayed on the UI.
        proposals[proposalID].active = false;

        // Adjust balance and keep track of traded tokens
        studentBalance[msg.sender] -= tokens;
        studentBalance[receiver] += tokens;
        tradedTokens[msg.sender] += tokens;
        tradedTokens[receiver] += tokens;

```

```
311         // Create the transaction for sending tokens
312         Transaction memory transaction;
313         transactionsCount++;
314         transaction.sender = msg.sender;
315         transaction.receiver = receiver;
316         transaction.reason = reason;
317         transaction.tokens = tokens;
318         transaction.creationTime = time;
319         transaction.id = transactionsCount;
320         transaction.senderFeedback = NO_FEEDBACK;
321         transaction.receiverFeedback = NO_FEEDBACK;
322         transactions[transactionsCount] = transaction;
323     }
324 
325     /**
326      * The sender of the feedback needs to first check the list of the
327      * transactions and see which transaction it wants to leave feedback, then
328      * it needs to read the transaction ID.
329      *
330      * @param transactionID the transaction ID of reference
331      * @param score the feedback to leave
332      */
333     function leaveFeedback(uint transactionID, int score) external {
334         require (MIN_SCORE <= score && score <= MAX_SCORE);
335         (bool can, uint res) = canLeaveFeedback(msg.sender, transactionID);
336         require(can /* && (res == 1 || res == 2) TODO future? */);
337 //
338         if (res == 1) {
339             transactions[transactionID].senderFeedback = score;
340             reputations[transactions[transactionID].receiver] += score;
341         } else if (res == 2) {
342             transactions[transactionID].receiverFeedback = score;
343             reputations[transactions[transactionID].sender] += score;
344         }
345     }
346 
347     /**
348      * This function tells whether a participant of a transaction can leave
349      * a feedback (to the other participant).
350      *
351      * @param sender the sender of the feedback
352      * @param transactionID the transaction ID of reference
353      *
354      * @return (can, res)
355      */
356     function canLeaveFeedback(
357         address sender,
358         uint transactionID
359     )
360         view
361         internal
362         returns (bool can, uint res)
363     {
364         if (
365             transactions[transactionID].sender == sender
366             && transactions[transactionID].senderFeedback == NO_FEEDBACK
367         ) {
368             res = 1;
369             can = true;
370         } else if (
371             transactions[transactionID].receiver == sender
372             && transactions[transactionID].receiverFeedback == NO_FEEDBACK
373         ) {
374             res = 2;
375             can = true;
376 //          } else { // Not required?
377 //              res = 0;
378 //              can = false;
379         }
380     }
381 }
```