Blockchain-based Online Contracting System

Ashee Jain, Srujana Sabbani, Shrawan Kumar, Kapilkumar Kathiriya
22111073, 22111083, 22111080, 22111028
asheejain22, srujanas22, shrawank22, kapilkmr22}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

***Abstract*—This document describes the workflow, architecture, features implemented and hurdles faced during the implementation of a multi-stakeholder Blockchain-based online contracting system. In contracting system, agreements are negotiated and managed from their creation to their execution by the party to which they have been assigned. The use of a contract management software program can reduce risk and ensure compliance by expediting the process of bringing a contract into the platform and quickly identifying potential problems within it. It is possible to produce risk analysis reports, compliance tracking, and more by analyzing all contracts across the business using the software.**

***Index Terms*—Blockchain, smart contract, Ethereum, contracting system**

I. Introduction

Contracts have been used to procure goods or services by government or private companies from a long time. The contract contains the essential information and binding agreements between the project issuer and the party implementing the project. It is essential that contracts be formulated with a multi-stakeholder approach for a successful project to be accomplished. However, business has always faced challenges when it comes to contract management. Coordinating the multiple activities associated with blockchain, including authoring, negotiations, renewal, enforcement/execution, and renewal, is the greatest challenge. A good deal of time must be invested in each of these activities. That is when blockchain comes into picture. Blockchain technology is based on smart contracts. Smart contracts provide network automation and allow the conversion of paper contracts into digital contracts. The legality of smart contracts can be compared to that of paper contracts. The blockchain-based contract management system eliminates the need for intermediaries, is verifiable and accurate, and offers enhanced security and data backup. All stakeholders benefit from such an automatic management system that provides transparency.

II. Methodology

The proposed system implements a multi-stakeholder blockchain-based contract issuing and management system with "M" tenders created by "N" parties. A set of 10 validators get selected on creation of a tender. The tender should get validated by atleast 6 validators to be considered a valid tender, otherwise, it gets suspended. The validator have to invest a fixed amount "validation fees" in the tender it verifies to eradicate favourism or other malicious verification of the tenders. Bids can be placed by other parties on the on-going tenders within the set deadline. Once the deadline is crossed, a bidder is selected based on the quoted amount. The top 5 bids are presented to the tender owner with the bidders' trust score, which is an indicator of the party's performance in previous projects. Once a bidder gets selected for a tender, an entity called project is created. Project has all the milestone details like number of milestones, timeline of each milestone etc which is set after agreement from both parties i.e., tender owner and bidder. Milestones can be marked completed by the bidder and on confirmation from the tender owner, it is marked completed in the system and a work score based on the number of days to complete the milestone is provided to the bidder. Finally, on successful completion of the last milestone the project and be marked completed by the tender owner. The validators get back their investments along with reward. At any point of time, if tender owner marks the project as halted the rewards applicable at that point of time are added to the bidder's account.Also, validators get back their investment without any reward. The reward given to bidder is equivalent to the average rating of work scores of all milestones. The reward provided to validators is 0.005% of their investment.

III. Architecture

This section contains the detailed description of the smart contracts and other attributes along with frontend integration architecture. Also, the workflow of entire project is mentioned in workflow section.

A. Smart Contracts Description

The details of the smart contracts in a sequential manner is described in this section.

1) **PartyContract.sol**
   a) **createParty(string memory _name, string memory _contactNumber, string memory _email, string memory _password, address _partyAddress) public isOwner(_partyAddress)**
   Function to create party, called when an individual or company registers itself on our platform.
   b) **updateParty(string memory _name, string memory _password, address _partyAddress) public isOwner(_partyAddress) isPartyExists(_partyAddress)**
   Function to update details of an already existing party, can be called only by the owner of the party account.

c) **function deleteParty(address _partyAddress) public isOwner(_partyAddress) isPartyExists(_partyAddress)**
Function to delete an already existing party, can be called only by the owner of the party account.

2) **TenderContract.sol**

a) **createTender(address _partyAddress, uint256 _budget, string memory _title, string memory _description, uint256 _deadline, uint256 _totalMilestones) isOwner(_partyAddress)**
Function to create a tender, called by existing party.

b) **sortByTrustScore() public view returns(address[] memory)**
Function to sort all existing parties by their trust score.

c) **validate(uint256 _tenderId, bool _vote)**
Function to validate a project, can be called only by a party which is selected as a validator for that tender.

d) **isValid(bool[] memory _validationVotes) public returns (bool)**
Function to check if the tender got minimum validation votes.

e) **getAllActiveTenders() public view returns (Tender[] memory, uint256)**
Function to get all active tenders on the platform.

f) **getMyTenders(address _partyAddress) public view returns (Tender[] memory)**
Function called by a party to get all its tenders.

g) **updateTender(address _partyAddress, uint256 _tenderId, uint256 _budget, string memory _title, string memory _description, uint256 _deadline, uint256 _totalMilestones) public isTenderOwner(_partyAddress, _tenderId)**
Function to update the tender details, can be done only by the tender owner before it gets open for bidding.

h) **deleteTender(address _partyAddress, uint256 _tenderId) public isTenderOwner(_partyAddress, _tenderId)**
Function to delete the tender, can be done only by the tender owner before it gets open for bidding.

3) **BidContract.sol**

a) **createBid(address _bidderAddress, uint256 _tenderId, string memory _bidClause, uint256 _quotedAmount)**
Function to create bid on an active tender within deadline. Tender owner and validators of that tender cannot place a bid.

b) **getMyBids(address _bidderAddress) public view isOwner(_bidderAddress) returns(Bid[] memory, uint)**
Function called by a party to get all its bids placed on any tender.

c) **getAllBids(address _partyAddress, uint256 _tenderId) public view isTenderOwner(_partyAddress, _tenderId) returns(Bid[] memory)**
Function called by tender owner to get all bids placed on that tender.

d) **updateBid(address _bidderAddress, uint256 _bidId, string memory _bidClause, uint256 _quotedAmount) public isBidOwner(_bidderAddress, _bidId)**
Function to update bid details only within bidding deadline.

e) **deleteBid(address _bidderAddress, uint256 _tenderId ,uint256 _bidId) public isBidOwner(_bidderAddress, _bidId)**
Function to delete bid only within bidding deadline.

f) **selectTopBids(address _partyAddress, uint256 _tenderId) public view isTenderOwner(_partyAddress, _tenderId) isValidTender(_tenderId) returns (Bid[] memory)**
Function to select lowest 5 bids based on quoted amount.

g) **finaliseWinnerBid(address _partyAddress, uint256 _tenderId, uint256 _bidId) public isTenderOwner(_partyAddress, _tenderId)**
Function to finalise winner bid, called when the tender owner selects a bid out of the top 5 suggested. The winner bid's status is changed to accepted, while all other get rejected.

h) **filterBidsByQuotedAmount(Bid[] memory bidList) public returns(Bid[] memory)**
Function to filter the bids based on quoted amount.

4) **MilestonesContract.sol**

a) **setMilestoneTimeline(uint[] memory _milestoneTimePeriods, uint256 _bidId, uint256 _tenderId) public**
Function to set the milestones timeline by the tender owner.

b) **markMilestoneComplete(uint256 _tenderId, uint _milestoneNumber, uint _days) public**
Function to mark milestone completed called by bidder(winning bid's party).

c) **approveMilestoneCompletion(uint256 _tenderId, bool approval) public**
Function called by tender owner validate the milestone completion claim by bidder.

d) **projectCompleted(uint256 _tenderId) public**
Function to mark project completed, after all milestones are completed.

e) **projectHalt(uint256 _tenderId) public**
Function to halt the project, called by tender owner.

f) **addMilestone(uint256 _tenderId, uint256 _days) public**
Function to add new milestones as and when required, only if the project is ongoing.

g) **filterProjectsByMilestones(Project[] memory projectList) public view returns(Project[] memory)**

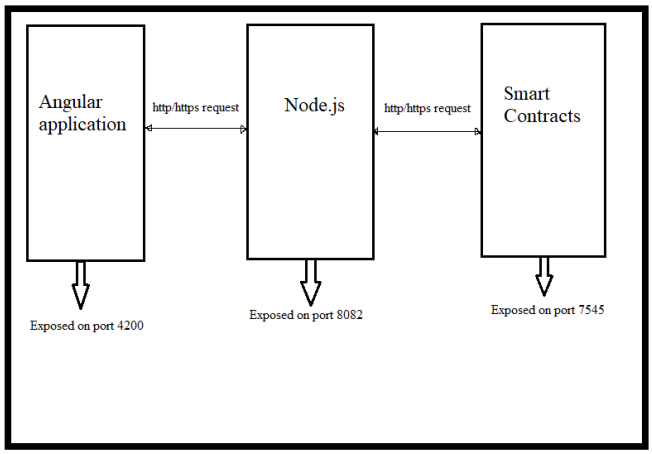

Fig. 1. Frontend integration with smart contracts

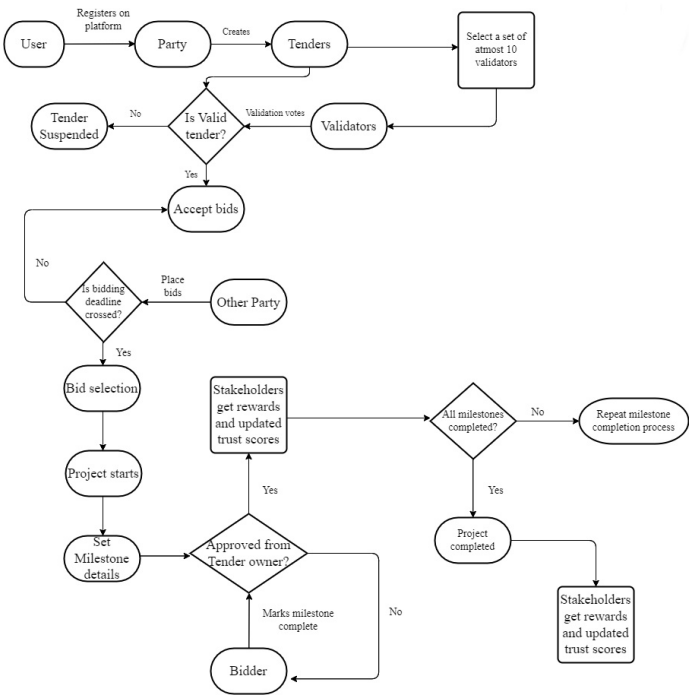

Fig. 2. Workflow of the project

Function to filter projects by milestones completed.

B. Frontend Integration

The smart contracts interact with Angular application through Node.js as shown in the figure 1

C. Workflow Diagram

The workflow of the entire project is shown in the figure 2

IV. Implementation of the Project

In this section, we describe the features we implemented in the project as well as the quality attributes we adhered to during their implementation. The future work section describes all the features we were unable to implement due to time constraints but would like to work on in the future.

A. Features implemented

The following features have been implemented in the project -

1) **Party creation: -** When an individual or company registers on our platform a party is created, with an initial balance of 50 tokens.
2) **Tender creation: -** A party can create a tender on the platform of budget not more than double of its current balance.
3) **Validation of tender: -**This tender is then validate by a set of 10 selected validators based of validator selection criteria. The validators validating a tender have to invest 10 tokens in it.This eradicates the malpractices by validators like validating tenders of favoured parties etc.
4) **Bid creation: -** Once the tender is validated, it is open to accept bids from other parties within the set deadline. The quoted amount of the bid can't exceed the budget of the tender.
5) **Selection of bids: -** When the deadline is crossed, the placed bids are sorted based on their quoted amount in ascending order and the 5 lowest bids are presented to the tender owner for selection.
6) **Start of project: -** On selection of a bidder by the tender owner, project is created which contains the milestone details like number of milestones, time period of each milestone etc, which are agreed upon by both the parties. The bidder owner will get half the quoted amount on start of project and remaining amount on each milestone completion.
7) **Milestone completion: -** Bidder can mark a milestone complete but milestone will only be considered completed when bidder's claim is verified by the tender owner.
8) **Project status: -** Project can be marked complete by the tender owner only if all milestones are completed. Project can be halted/suspensed at any point of time by the tender owner. However, this will have a negative impact on their trust score value.
9) **Reward System: -**
   Here,
   Tender owner = issuer party of the tender
   Bidder = the party who won the bidding
   Validators = parties selected to validate the tender
   a) **Validator Rewards: -** Validators get 0.005% of their investment as reward on successful completion of project along with the invested money and a positive trust score. However, if the project halts they only get their investment back along with negative impact on their trust score value.
   b) **Bidder Rewards: -** Bidder will get rewards based on the number of days it took them to complete the work.

c) **Tender owner Rewards: -** Tender owner will get positive trust score on successful completion of project and a negative trust score if the project halts. They will also need to pay penalty to the bidder.

d) **Reward Calculation Metric: -**

i) workScore = (No. of days decided to complete milestone - Actual days) / No. of days decided to complete milestone

ii) newTrustScore = oldTrustScore + average of workScores of all milestones

B. Quality Attributes adhered

The following quality attributes were taken into consideration while implementing this project -

1) System is scalable to 100 users.
2) The project description character limit is atmost 200.
3) Each tender has atmost 10 validators, which are different for every tender.
4) More than half of the validators should validate a tender for it to be considered valid.
5) The quoted amount of bid can't exceed the budget of the tender.
6) A maximum of 15 bids can be placed on an ongoing tender.
7) Bidding deadline is 24 hours after tender validation.
8) Validation deadline for validators is 6 hours.
9) Party should have atleast half of the project budget it is creating.

C. Security Attributes implemented

The following security attributes were implemented in this project to maintain the privacy of all participants -

1) The tender owner cannot be a validator for it's own tender.
2) Bidders cannot view other bids to prevent gaining insights for guessing a good quote price. The bid details can only be viewed the bidding party and the tender owner.
3) Parties can't see details of other parties, one can only view the active tenders created by parties. Hence, incorporating user anonymity in the system.
4) Only the issuer party of a tender has rights to modify any details of the tender even though it is visible to all.

D. Implementation failures and successes

1) During UI to contracts integration through NodeJs we have tried various method such as using docker , truffle, importing json files and abi address to web3 in angular app, so on. At last, NodeJs with truffle helped in integrating the whole project.
2) The next hurdle we encountered was segregating the contracts and modularising the code for better readability. The large solidity code file were fiving error on deploying in truffle as the files exceeded the 24kB size and a Spurious Dragon error was shown. To confront this problem, we have used inhertance, and importing of contracts into another contracts, then used setters and getter function to change the state variables correspondingly. Also, we used mapping initially inside structs, which were removed and handled the data without maps properly.
3) Token functionality was a problem in the start, but we solved it by importing ERC20 interfaces and using the functions by ovveriding and super keyword.

E. Future Work

The following modifications can be made in future, which were not implemented now -

1) The bid selection algorithm can be improved.
2) We can implement validator's criteria using Proof-Of-Stake to improve the trust-worthiness of the system.
3) Tokens can be implemented in a proper way, as of now only the balance variable gets updated.

V. Result

This section includes the screenshots of the various modules of the frontend of the project. The screenshots shows the some of the features and constraints implemented. The screenshot are sequentially ordered from user login page to the deletion of the tender.

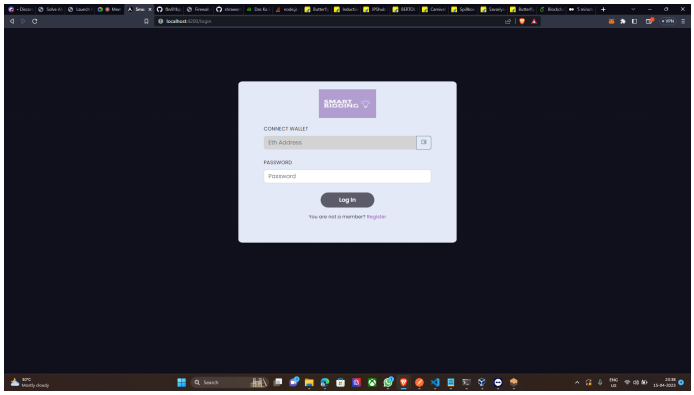

Fig. 3. User login page

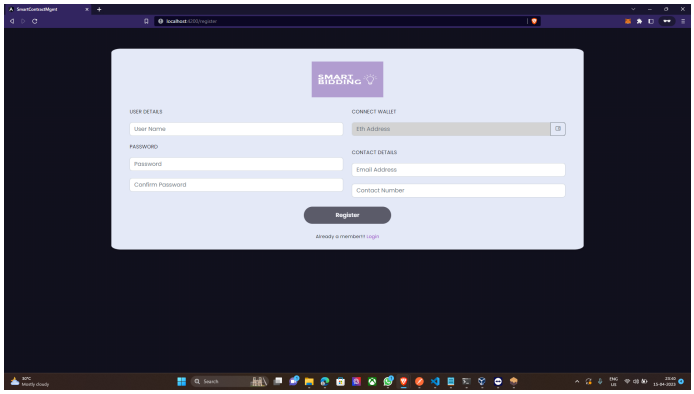

Fig. 4. User registration page

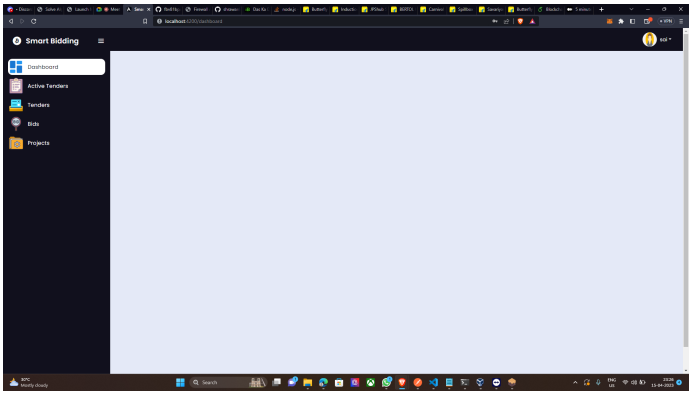

Fig. 5. User homepage

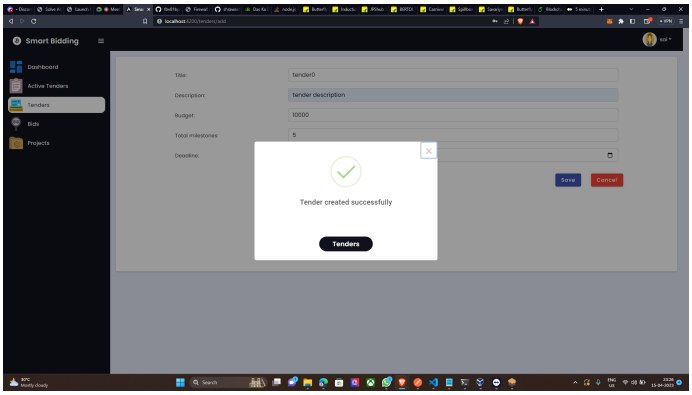

Fig. 6. Successful Tender creation

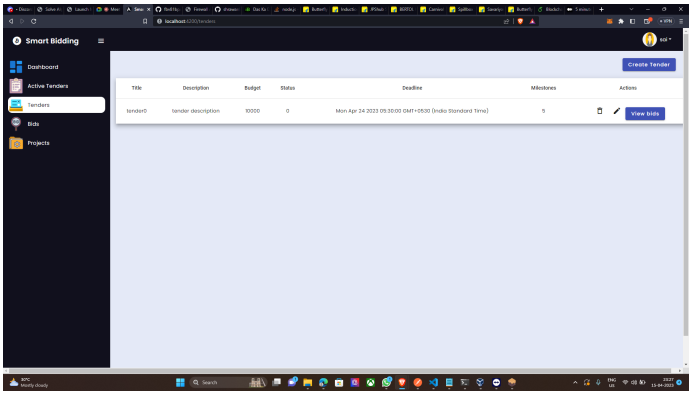

Fig. 7. List of self-created tenders

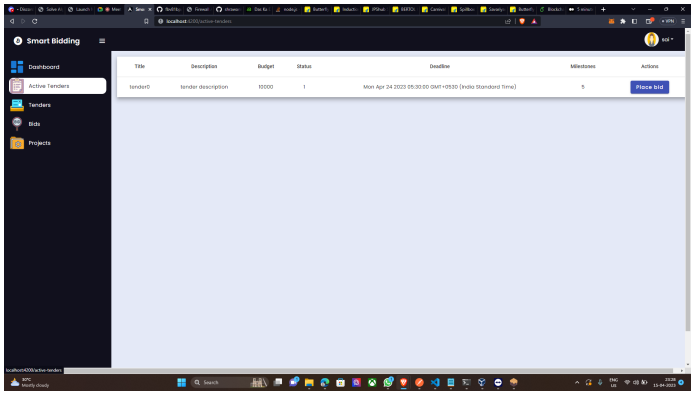

Fig. 8. List of Active Tenders

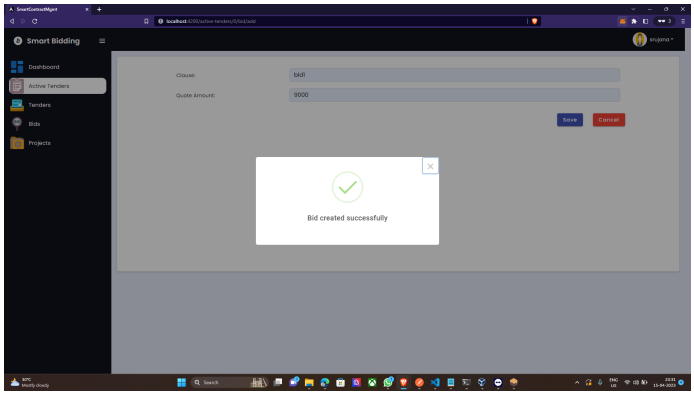

Fig. 9. Bid creation

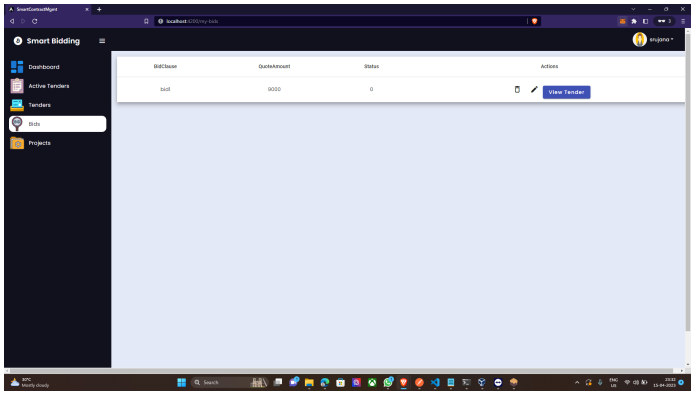

Fig. 10. List of self-created bids

VI. Individual Contribution

The individual contribution of each member of the project is listed in the table: I

VII. Detailed discussion logs

This section contains the logs of discussions with TAs as well as internal team meetings.

1) **Discussions with TA: -**
   a) Doubt's regarding validator's criteria were discussed with Sumit in person.
   b) Meet with Mohit to clarify doubts about how to integrate Token.sol with other contracts as and when required.
   c) Discussion with Sumit over call when facing issues to integrate frontend with smart contracts and also doubts regarding token transfer from validator account when they validate to tender account.
   d) Meet with Sumit a few more times during project classes to show the progress of the project.
2) **Internal team meetings: -** We had several online and offline meetings throughout the semester to discuss on various aspects of the project like project design, tech-

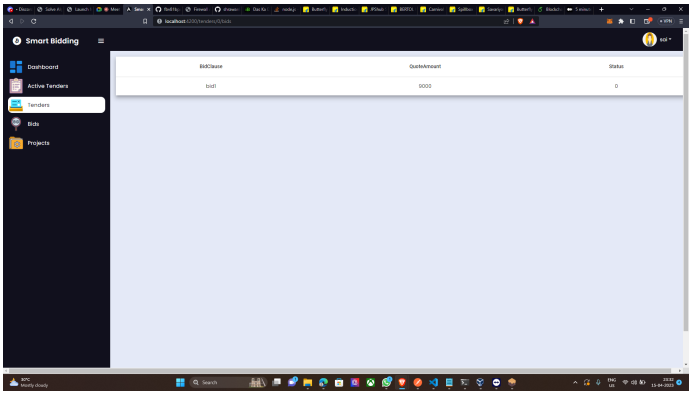

Fig. 11. Bids placed on a tender

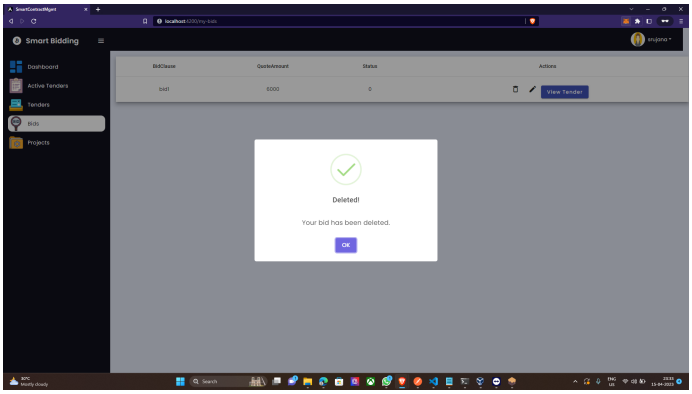

Fig. 12. Bid deletion

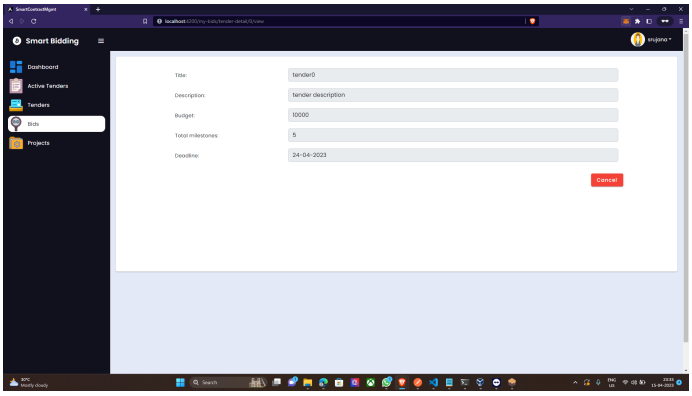

Fig. 13. Tender updation

nologies to be used, implementation of all the required features etc.

REFERENCES

[1] https://dev.to/willkre/create-deploy-an-erc-20-token-in-15-minutes-truffle-openzeppelin-goerli-33lb

[2] https://github.com/agbanusi/Music-share-platform-through-Blockchain

[3] https://docs.openzeppelin.com/contracts/3.x/erc20

[4] D. Mali, D. Mogaveera, P. Kitawat and M. Jawwad, "Blockchain-based e-Tendering System," 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2020, pp. 357-362, doi: 10.1109/ICICCS48265.2020.9120890.

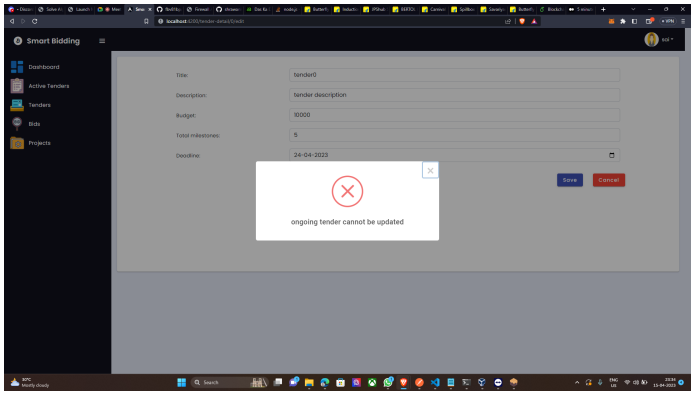

Fig. 14. Updation of ongoing tenders

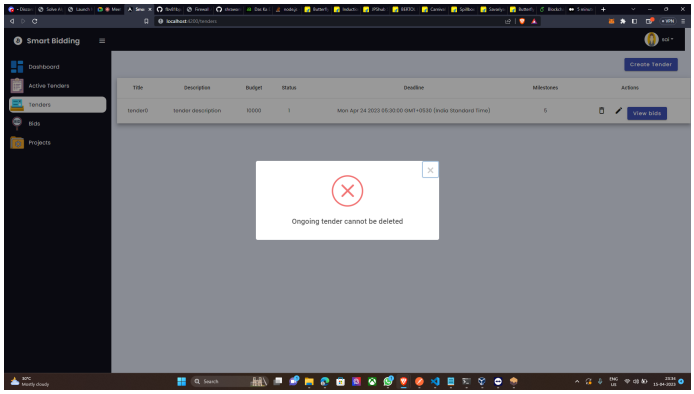

Fig. 15. Deletion of ongoing tenders

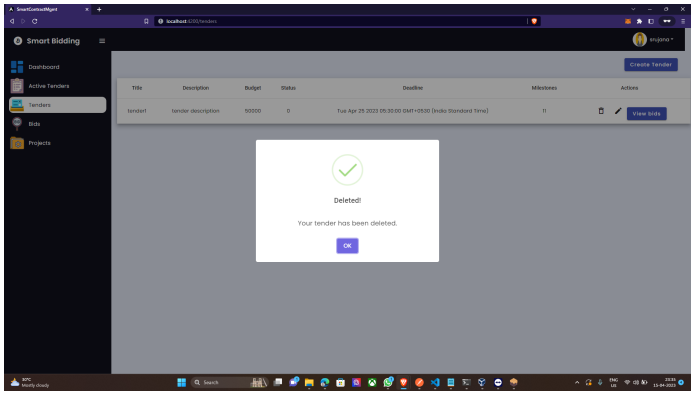

Fig. 16. Deletion of inactive(not validated) tenders

[5] Pawar, R. et al. (2023) "Contract/tendering system using blockchain," International Journal for Research in Applied Science and Engineering Technology, 11(3), pp. 471–474. Available at: https://doi.org/10.22214/ijraset.2023.49144.

[6] Kiyeng, D., Karume, S.M. and Masese, N. (2021) "Design of blockchain based smart contract for tendering," International Journal of Computer Applications Technology and Research, 10(10), pp. 222–225. Available at: https://doi.org/10.7753/ijcatr1010.1002.

Student Name	**Contribution of work**
Shrawan Kumar	PartyContract.sol, Token implementation, Functions to create bid and getting bid details in BidContract.sol, Test script for PartyContract.sol
Kapilkumar Kathiriya	TenderContract.sol, Token implementation, Test script for TenderContract.sol
Ashee Jain	Functions for validator selection, Functions sortByTrustScore, validate, isValid and validator selection in createTender function in TenderContract.sol, MilestoneContract.sol, several other getter, setter functions in all smart contracts, Functions to filter bids and projects, sort parties using trust score, Rewards implementation in smart contracts, Designed & Implemented the workScore and trustScore metrics for party rating, Documentation - Report & Presentation.
Srujana Sabbani	Frontend angular app setup, System design and flow diagrams, worked PartyContract.sol, TenderContract.sol and BidContract.sol, Segregated the contracts, worked on winner bid selection, Backend NodeJS setup, written APIs to call function in smart contracts, Integrated UI with NodeJs and then integrated NodeJs to Ethereum(using Truffle, Ganache), developed complete Ui- register flow, login flow, complete dashboard for tenders, bids, projects components and services with alerts and validations, connected Metamask wallet with UI, written Github - Readme file for project deployment.

TABLE I
INDIVIDUAL CONTRIBUTION LIST