CSE 526 – Blockchain App Development
Term Project

Name: Gulfam Hussain
UB Person# 50315477
Email: gulfamhu@buffalo.edu

This is a term project document for Blockchain App Development course. It contains all the details regarding the project being developed in different phases following software development life cycles. The final Dapp is deployed on the Infura cloud for the use and interaction by the decentralized users.

Project Phases:

Phase 1: This phase includes the research and identification of a decentralized application to be worked upon. The output of this phase is to identify a major area and a problem statement which could be solved using blockchain technology.
Phase 2: The purpose of this phase is to analyze the problem and develop design diagrams like use case, contract diagram, finite state machine (FSM) diagram for this project. The output of this phase is to come up with a coherent design document with all the diagrams and explanation.
Phase 3: This phase includes Design, development, deploy and testing of the smart contract part of the Dapp. The output of this phase is to develop XYZ-contract codebase, where XYZ is your Dapp name.
Phase 4: This phase includes implementation of security and events in the developed smart contract. The output of this phase is to include and provide security in the smart contract.
Phase 5: This phase includes the design of web app for the developed decentralized application and host it on ganache to run it.
Phase 6: This phase includes the deployment of designed Dapp on the Infura cloud-like service for distribution and use of application by decentralized users.
Phase 7: This phase includes the final project documentation consists of all phases and required diagrams putting it all together.

Phase 1
Project Description

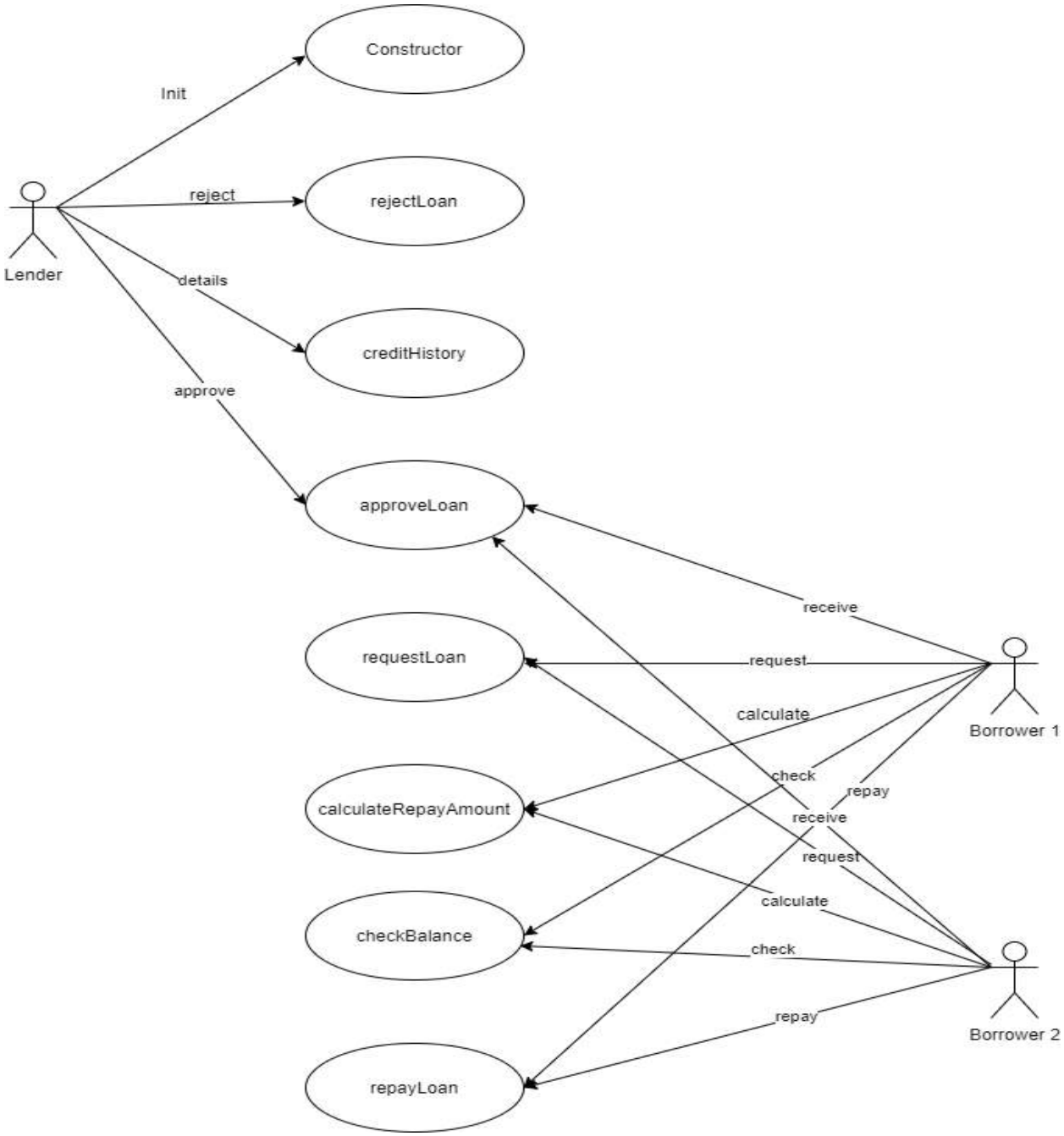
- **Major Area:** Financial Market Domain
- **Title:** Managing personal finance using Blockchain
- **Dapp name:** LoanPurse-Dapp
- **Clients:** Lenders, Borrowers(Requestors)
- **Abstract/Problem Statement:**

  - Financial markets are quite cumbersome for many of us and especially for the people who are in dire need of their finance but have a hard time arranging it. Banks and other middle entities make the process tedious, time-consuming and lending money from these entities make it costly by factors like the rate of interest. In the real world, I have often seen how difficult it is to manage personal finance in case of certain unprecedented situations. Arranging money from people who do not have any immediate relations makes the situation worse.
  - For instance, this pandemic has been hard on everyone. Managing personal finances for everyone has been a task. Students like us find it difficult to manage money after losing jobs and other sources of income.
  - This problem has a global scope where participants are decentralized and not necessarily known to each other. To address situations like these, I am planning to build a peer-to-peer decentralized application, where anybody can lend money to anyone over any place in the world enabling an opportunity for one user to get the return on his/her money and the other user to borrow money at a very reasonable rate without much hassle of involving the third party like banks, brokers, etc. This would enable and build trust between the lender and borrower which is an important aspect of blockchain. They do not need to give any fees or amount to a third party, and it would help them save their finance.
  - The transactions in this Dapp are recorded, verified, and validated. Hence all the transactions are secure. This Dapp will solve several similar issues and would be a boon to many in this pandemic and otherwise.

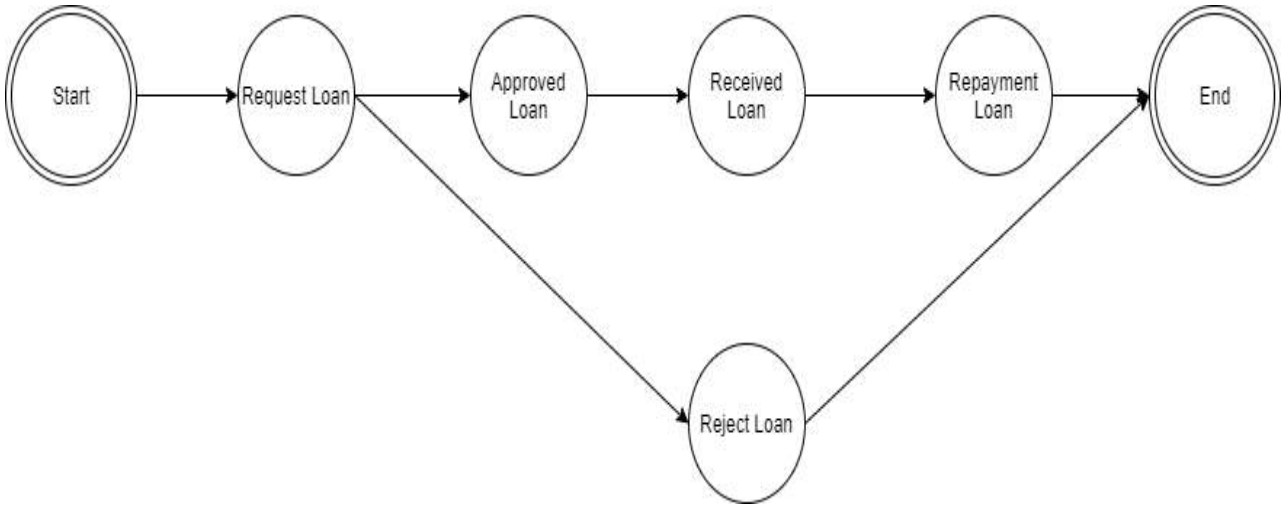
Phase 2
Design Diagrams

This phase includes the design diagrams of the selected project (LoanPurse- Dapp).

1) **Use case diagram:**



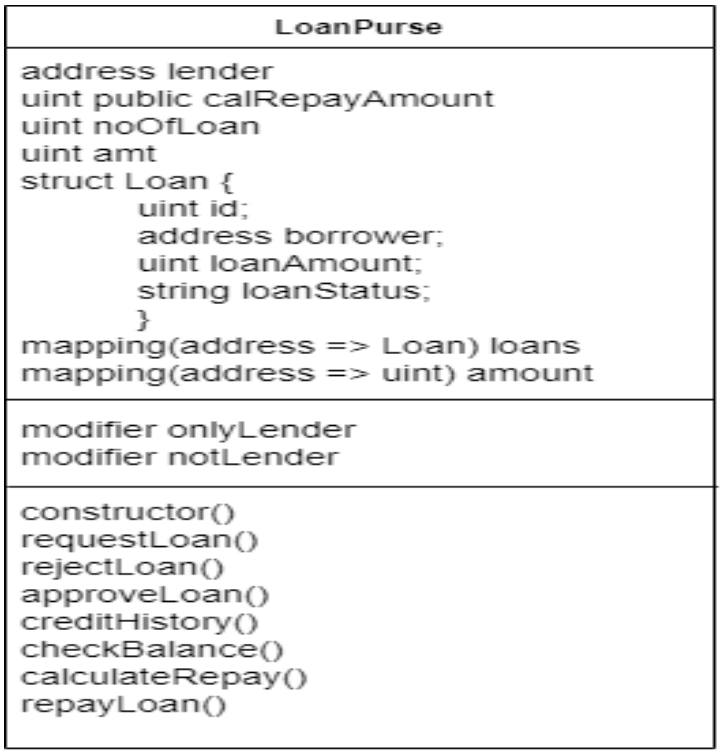
2) Finite State Machine (FSM) diagram:



States in the FSM above:

1) **Start/Init:** Initialization phase.
2) **Request Loan:** In this state, a borrower request for a loan from a lender.
3) **Approved Loan:** In this state, a lender approves the loan request from borrower and transfer the loan amount.
4) **Received Loan:** In this state, a borrower receives the loan amount from a lender.
5) **Repayment Loan:** In this state, a borrower repays the final loan amount to lender based on interest calculation and loan duration.
6) **Reject Loan:** In this state, a lender can reject the loan request from borrower.
7) **End/Done:** End phase of the state.

3) Contract diagram:



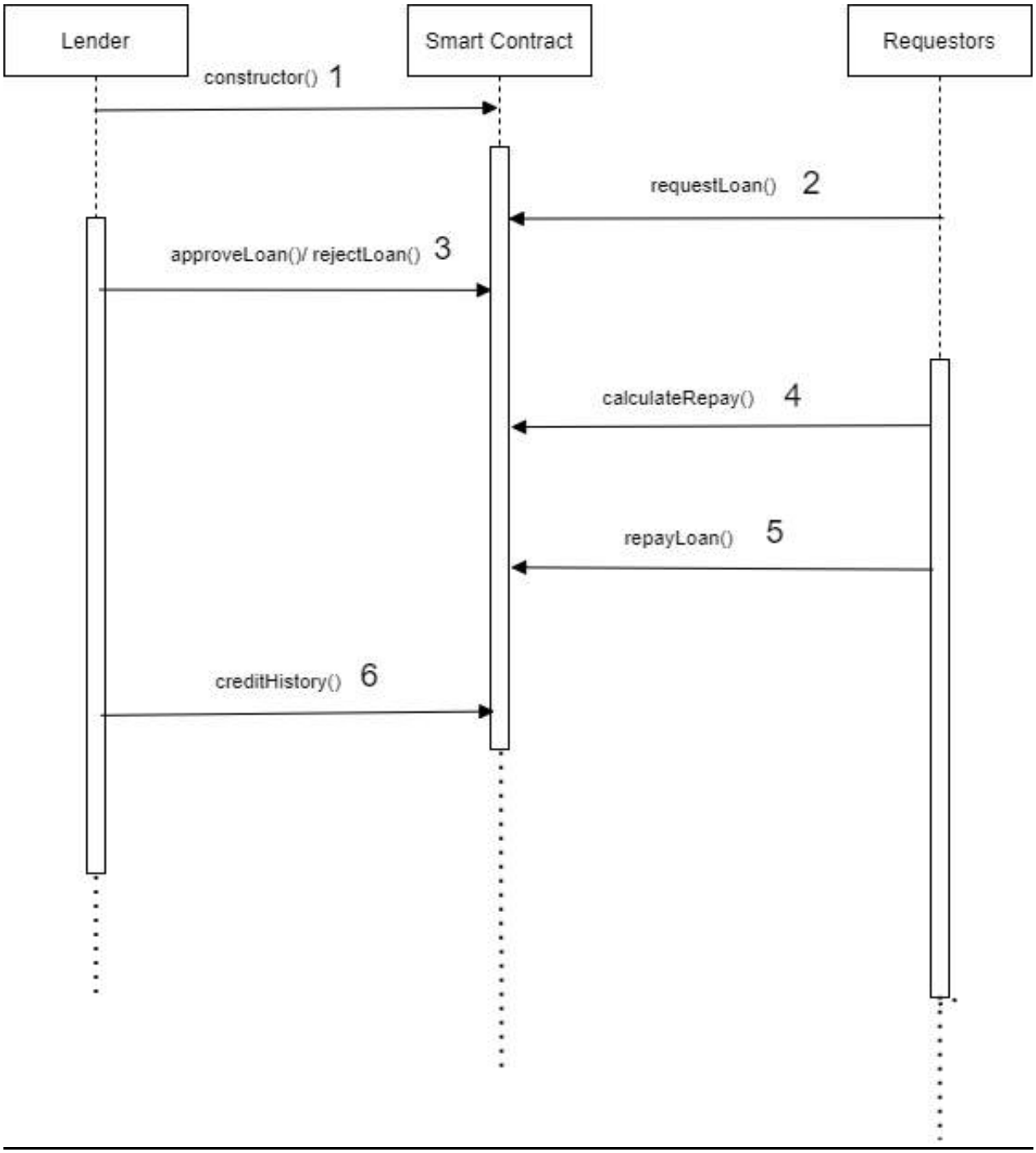
Modifiers:

1) **onlyLender:** modifier to validate functions/activities that can only be done by a lender like approve loan, reject loan etc.
2) **notLender:** modifier to validate functions/activities that can only be done by a borrower like request for loan, receive the loan etc. and cannot be done by lender

Functions:

1) **constructor() :** A default constructor is used to deploy the smart contract.
2) **requestLoan() :** Function where borrowers can request for a loan
3) **approveLoan() :** Function where lender can approve loan and transfer the loan amount to borrower
4) **rejectLoan() :** Function where lender can reject loan request
5) **creditHistory() :** Function to check credit history of borrowers
6) **checkBalance() :** Function to check the balance after the loan is transferred
7) **calculateRepay() :** Function to calculate the repay amount the accrued loan amount
8) **repayLoan() :** Function where borrowers repay the loan after the loan duration

4) Sequence diagram:



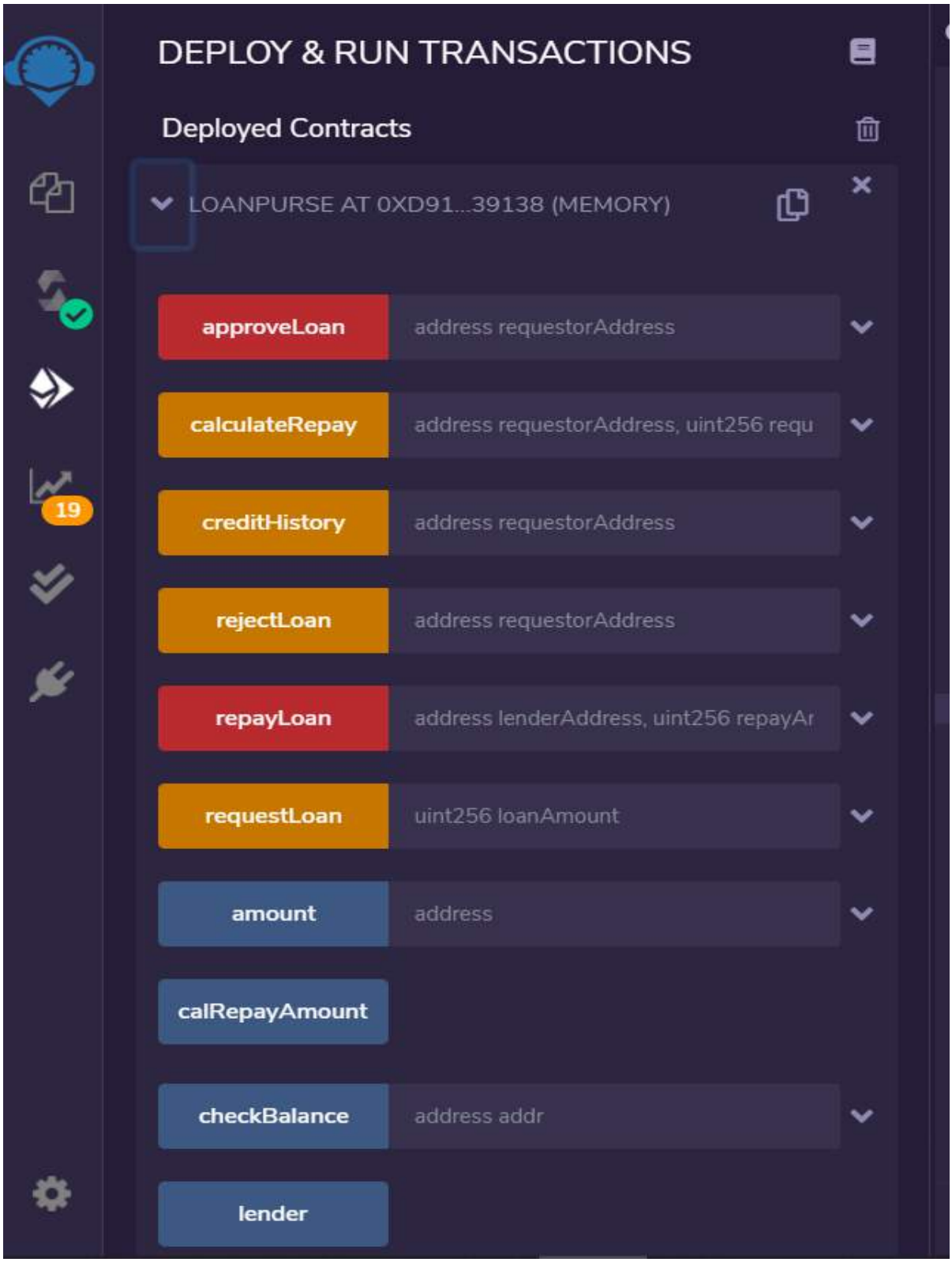
The above diagram shows the sequence in which the users of smart contract interact with SC. The orders shown in the diagram is being followed while interacting with the smart contract and in the same manner all the transactions are performed.

Phase 3
Design & Development of Smart Contract(Dapp)

Steps for Execution (Loan Process):

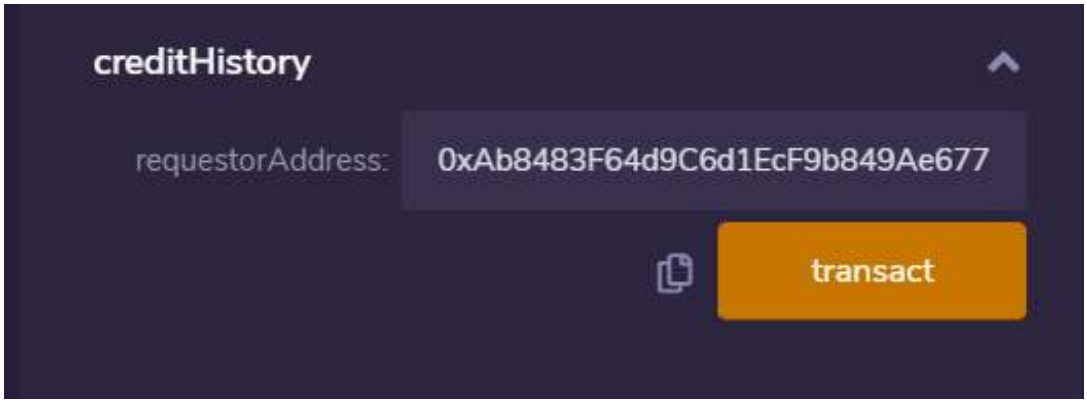
- Lender (Owner) is the one who deploys the smart contract.
- Borrower(Requestor) requests for the loan amount (e.g., 20 ETH) from the lender through requestLoan function.
- Lender then approves the requested loan amount from the borrower by transferring the same requested amount using approveLoan function and by entering the borrower's address.
- In case, the loan amount is approved, the requested amount gets deducted from the Lenders' account and gets added to borrower's account.
- Lender can also reject the loan request from the borrower using the rejectLoan function by entering the borrowers' address.
- In case, the loan amount is rejected by the lender, the account balances of both lender and borrower remain same as earlier except very less transaction charges.
- Using creditHistory function, we can check the details about the history of loans about the borrower like number of existing loans taken by borrower, requested loan amount and the status of the loan amount whether it is in requested state or in Approved state or the request has been rejected.
- Lender button shows the address of the lender/owner of the smart contract which helps to identify in case if someone wants to know who is the lender/deployer of that smart contract.
- Amount button shows the last requested loan amount by the borrower
- checkBalance button displays the current balance of the entered account address whether it is borrower or lender.
- Through calculateRepay function & calRepayAmount button, the borrower can calculate the amount to be paid back to the lender based on predefined rate of interest.
- The same calculated repay amount is being used by the borrower to pay to the lender through repayLoan function. Once the loan is repaid, the account balances of lender and borrower get updated.

- **SC deployed by lender:**

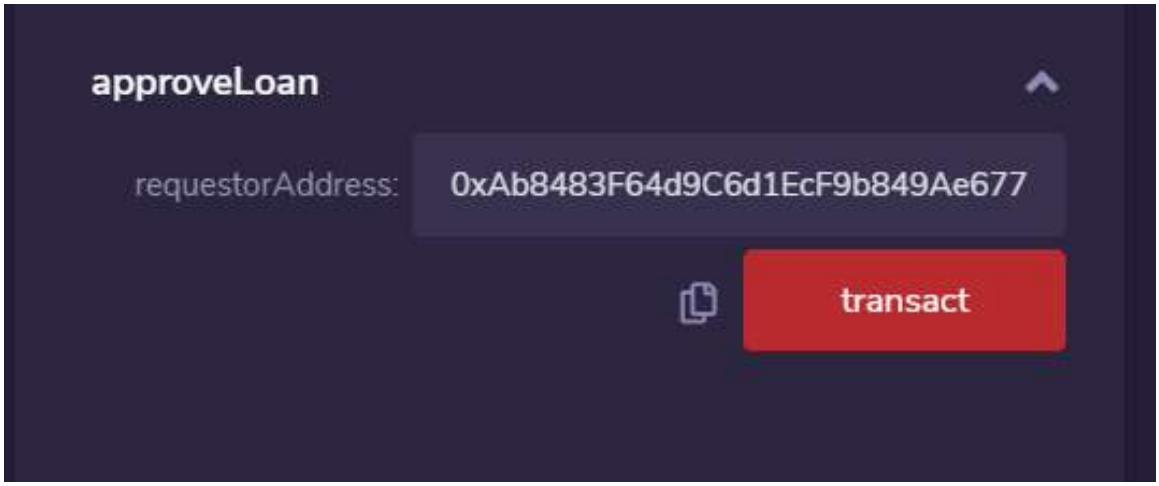


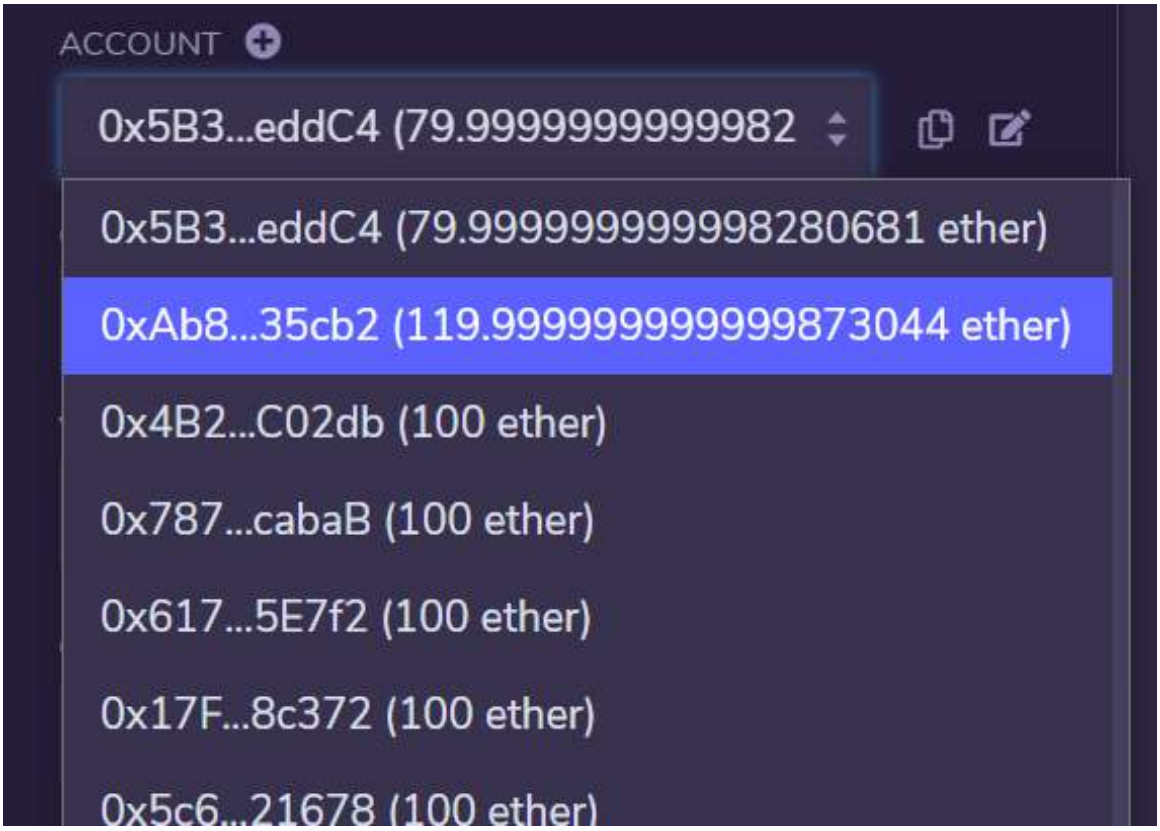
- **Loan request by borrower by filling out the details (loan amount):**

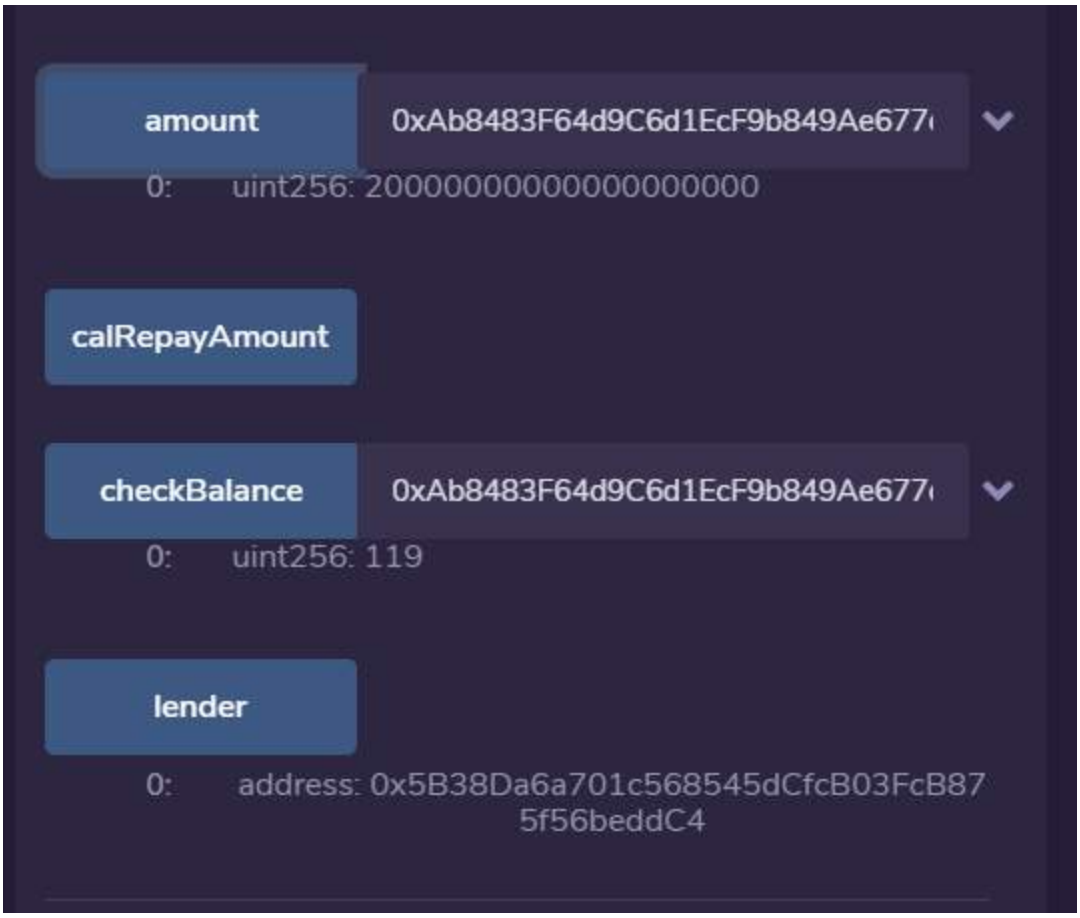
- **Credit history and loan status of borrower's account:**



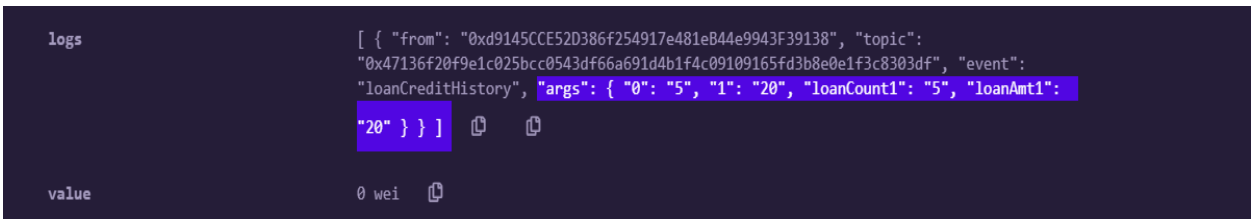
- **Loan approved by lender to the borrower by transferring the requested loan amount:**



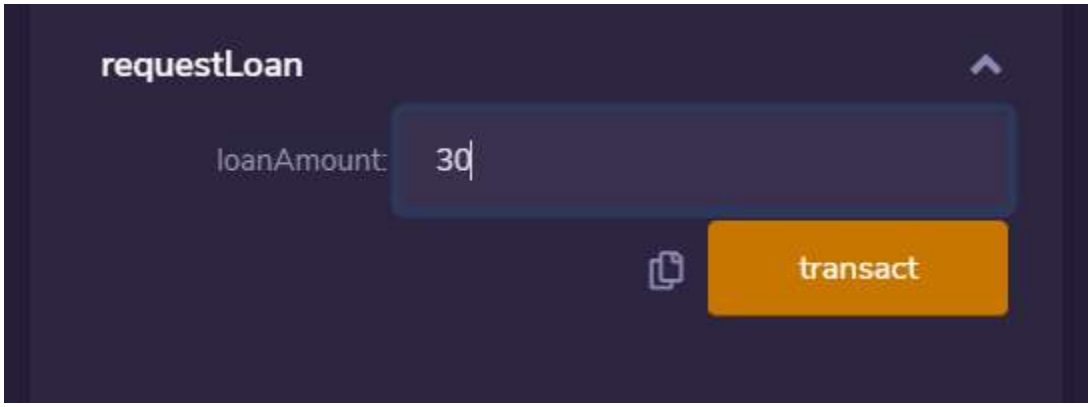


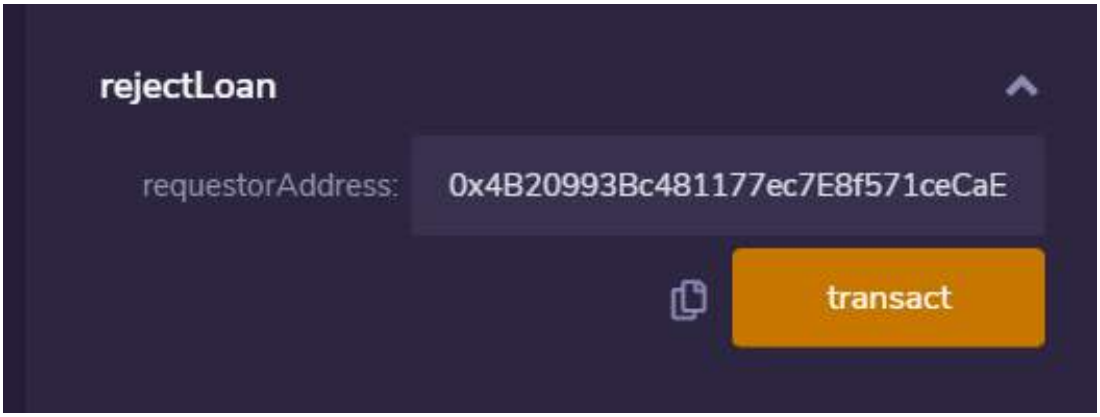


- **Credit history shows the details of loan request:**

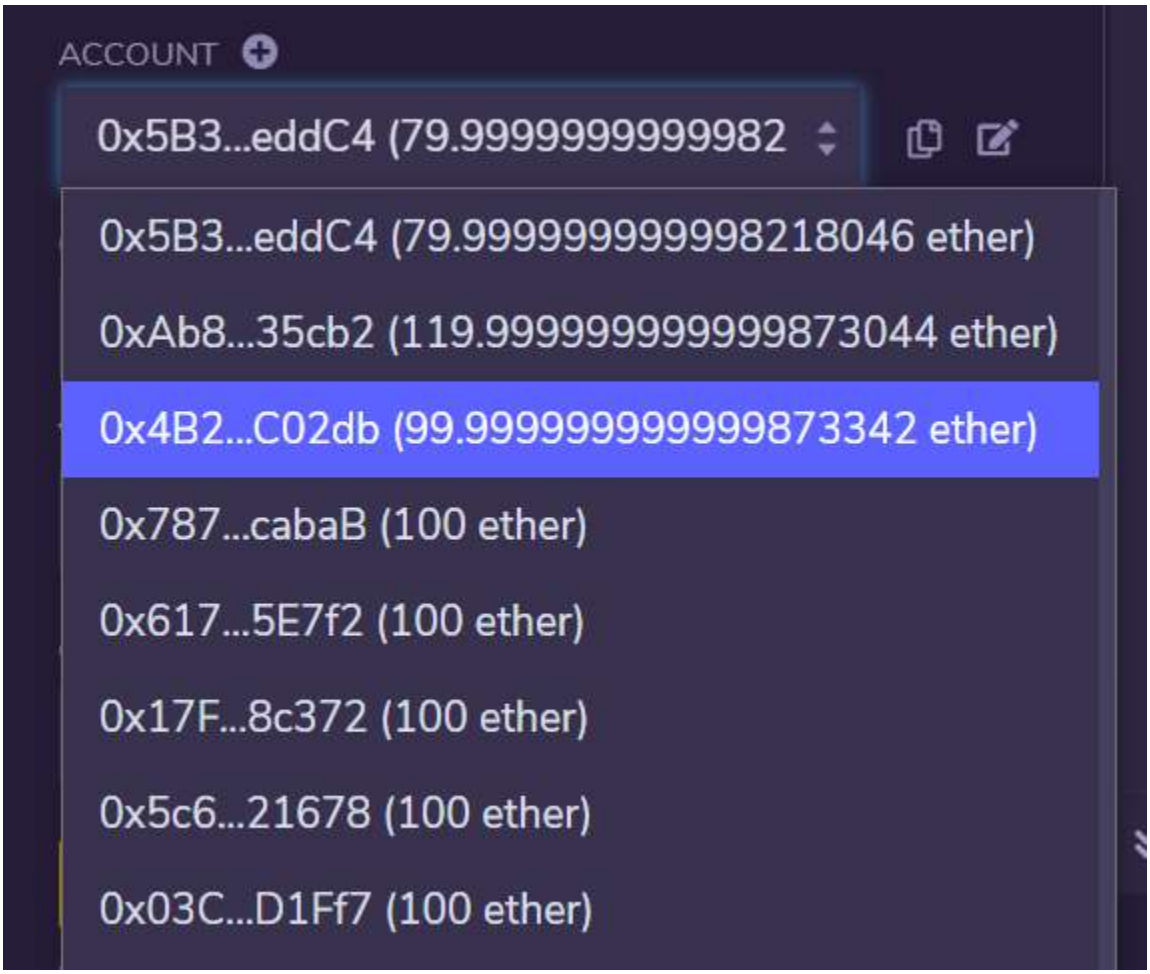


- **<u>Another Requestor requests for loan and this time Lender rejects the loan request:</u>**

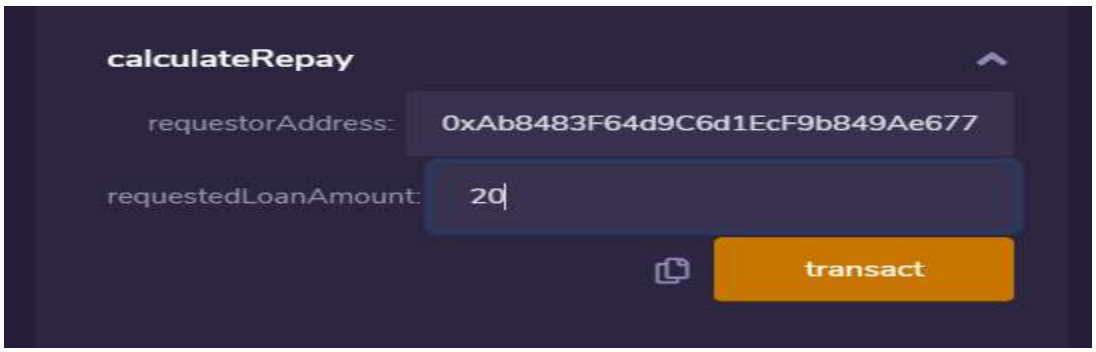




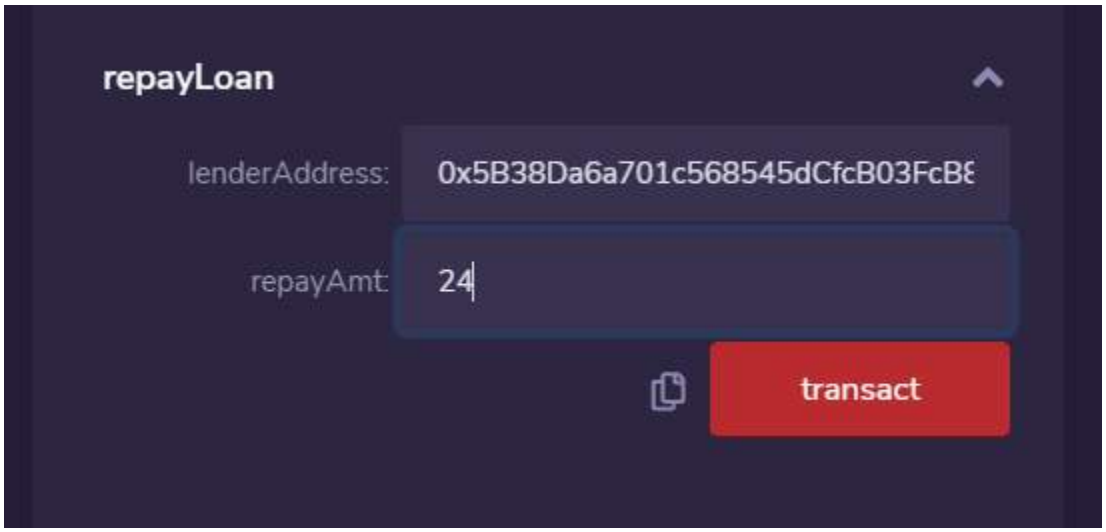
- **Loan gets Rejected and there is no change in the balance except a very little amount of transaction charges:**



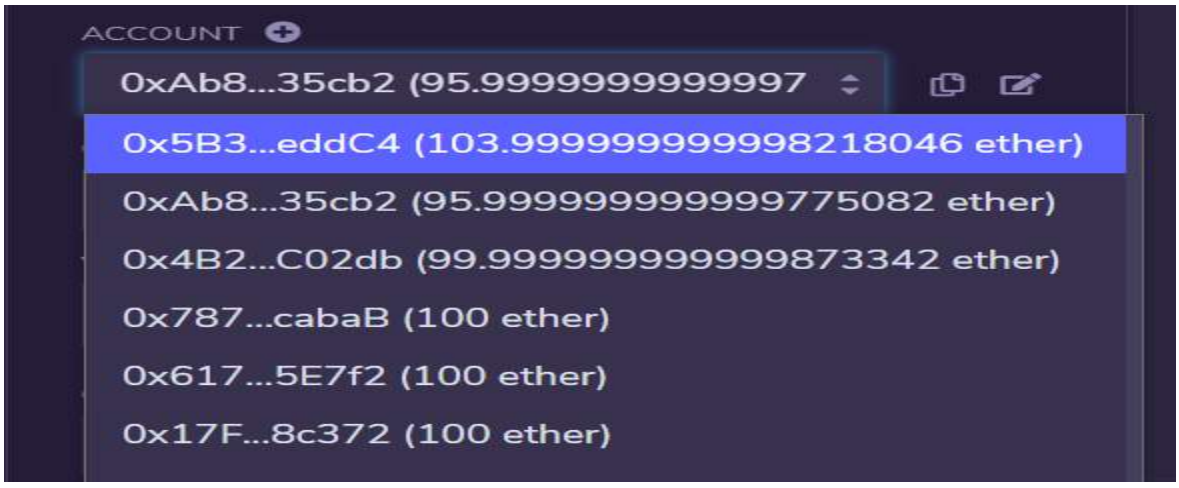
- **Requestor calculates the repay amount to be paid (e.g., repay amount on 20 Ether loan amount):**



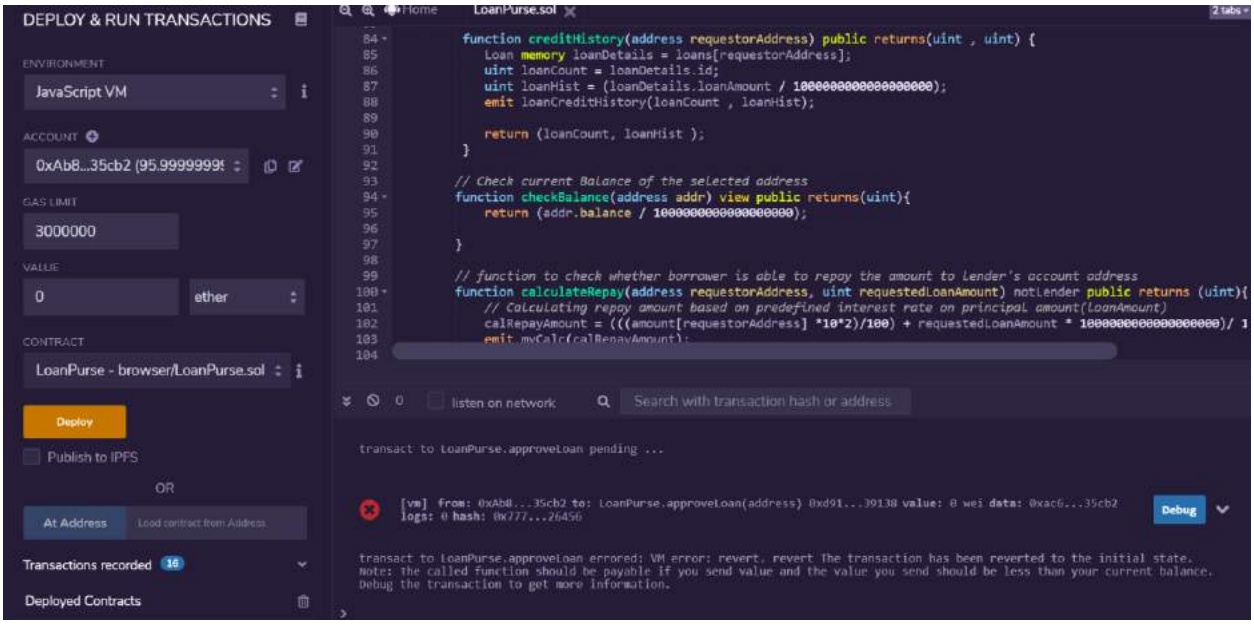
- **Requestor pays back the same calculated repay amount to Lender. The account balance of Requestor and Lender get updated after paying the repay amount:**

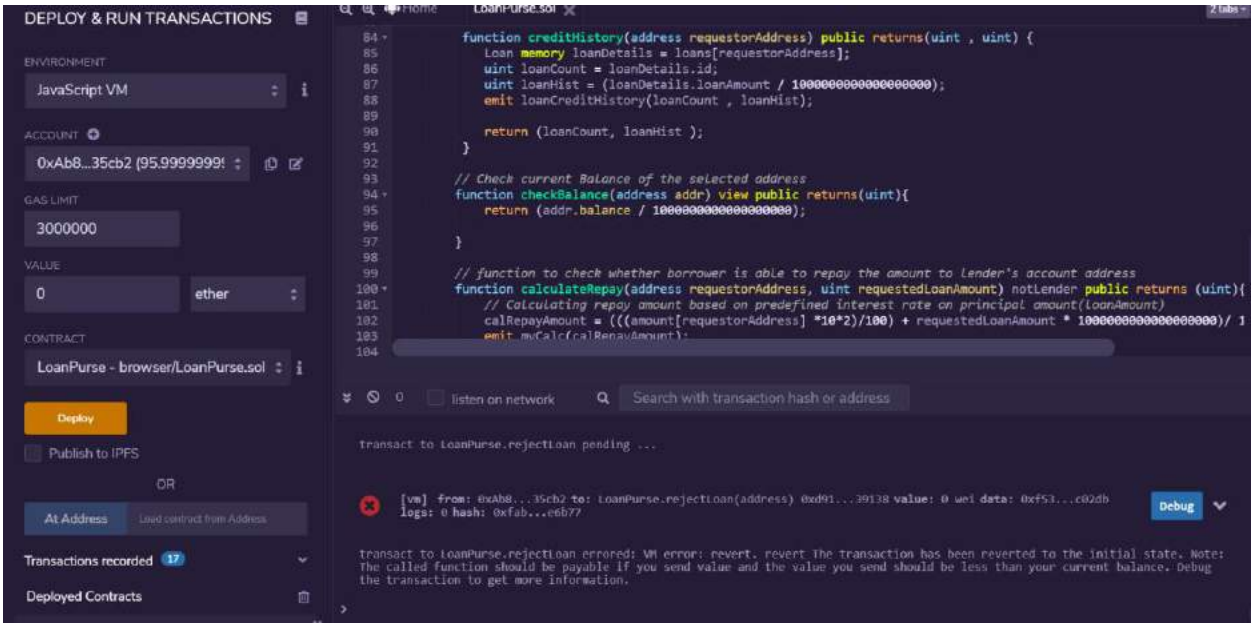


- **Lender receives 24 ethers in his account and the balance is deducted from borrower account:**

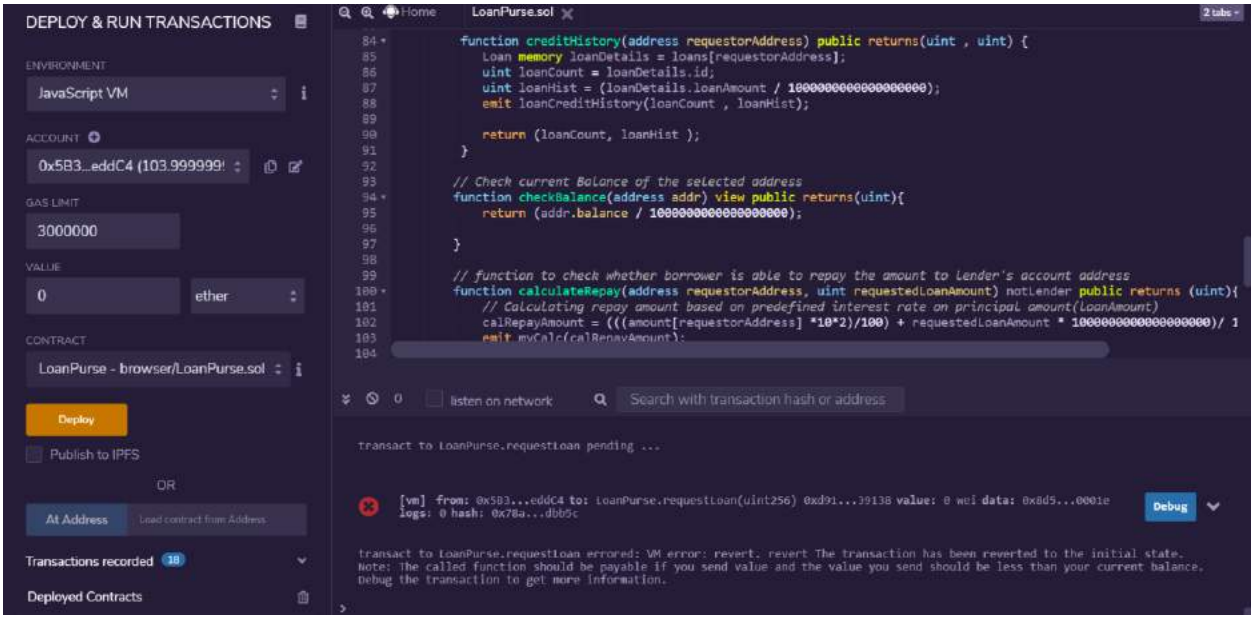


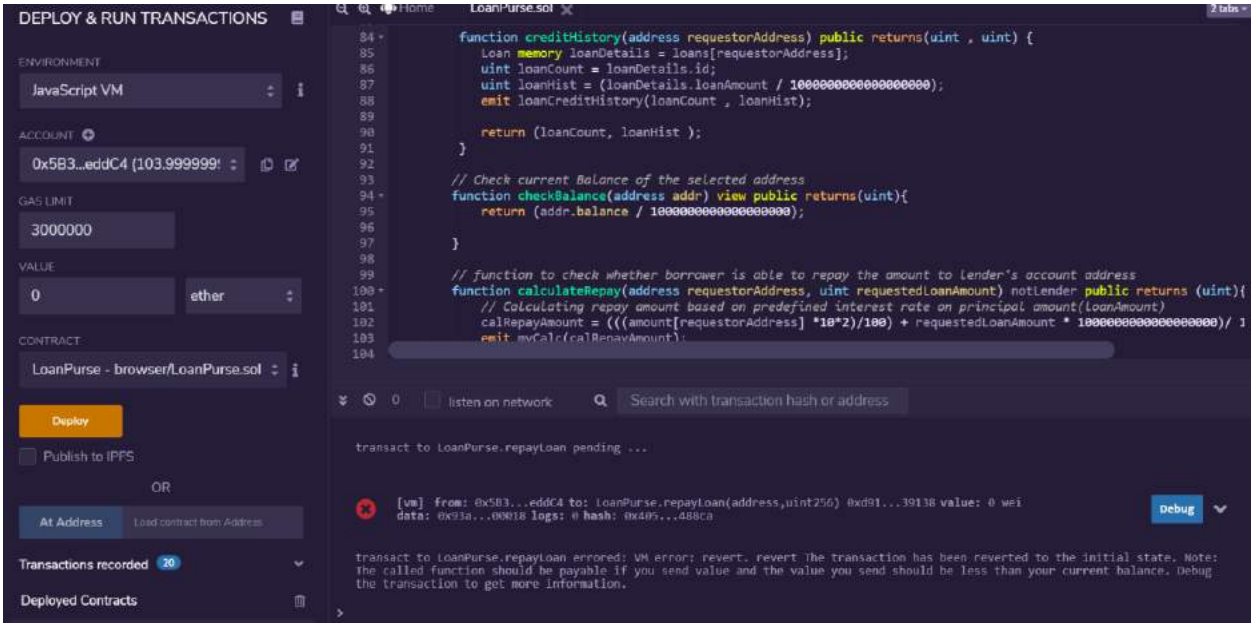
- **When the borrower tries to approve/reject the loan, SC gives an error(as expected based on function modifier):**





- **When the lender tries to request loan or repay the loan, SC gives an error(as expected based on function modifier):**

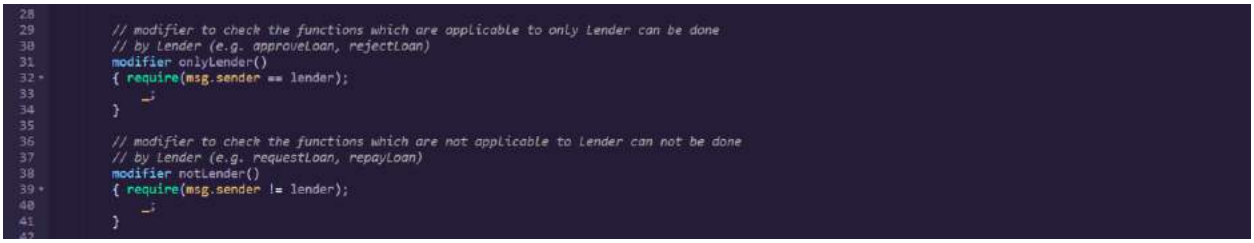




Code snippets of the LoanPurse.sol

- Variables, mapping, struct, and modifiers:

```
pragma solidity >=0.4.22 <=0.6.0;

contract  LoanPurse {
        address payable public lender;  // owner of the contract who deploy the contract
        uint public calRepayAmount; // calculate the Repay Amount based on predefined interest rate
        uint noOfLoan=0;  // count of total loans
        uint amt;
        mapping(address => Loan) loans; // mapping address to loan
        mapping(address => uint) public amount;

         // Loan structure
        struct Loan {
           uint id;
           address borrower;
           uint loanAmount;
           string loanStatus;
        }

        // Events are defined here
        event Initialized();
        event LoanRequested();
        event LoanApproved();
        event LoanRejected();
        event LoanRepaid();
        event myCalc(uint myRepay);
        event loanCreditHistory(uint loanCount1, uint loanAmt1 );


        // modifier to check the functions which are applicable to only Lender can be done
        // by Lender (e.g. approveLoan, rejectLoan)
        modifier onlyLender()
        { require(msg.sender == lender);
            _;
        }
```



```
        // modifier to check the functions which are applicable to only Lender can be done
        // by Lender (e.g. approveLoan, rejectLoan)
        modifier onlyLender()
        { require(msg.sender == lender);
            _;
        }

        // modifier to check the functions which are not applicable to Lender can not be done
        // by Lender (e.g. requestLoan, repayLoan)
        modifier notLender()
        { require(msg.sender != lender);
            _;
        }
```

- Constructor and other functions:

```
43      // constructor() is executed when the smart contract is deployed into the network
44      // and `msg.sender` which is the lender/owner, is the account creating this contract.
45      constructor () public {
46          lender = msg.sender;
47
48          emit Initialized();
49      }
50
51      // function to request Loan by borrower to Lender by giving few details about the borrower and reason for loan
52      function requestLoan(uint loanAmount) notLender public {
53          // add loan object to mapping
54          amount[msg.sender] = loanAmount * 10000000000000000000;
55          loans[msg.sender] = Loan(noOfLoan, msg.sender, loanAmount, "Loan Requested");
56
57          emit LoanRequested();
58      }
59
60      // function to approve the loan request by Lender and transfer the loan amount to borrower account
61      function approveLoan(address payable requestorAddress) onlyLender payable public {...}
75
76      // function to reject the loan request by Lender
77      function rejectLoan(address requestorAddress) onlyLender public {...}
83
84       function creditHistory(address requestorAddress) public returns(uint , uint) {...}
92
93      // Check current Balance of the selected address
94      function checkBalance(address addr) view public returns(uint){...}
98
99      // function to check whether borrower is able to repay the amount to Lender's account address
100     function calculateRepay(address requestorAddress, uint requestedLoanAmount) notLender public returns (uint){...}
107
108
109     function repayLoan(address payable lenderAddress, uint repayAmt) notLender public payable {...}
125 }
```

Phase 3 Conclusion and Outcome:

- Successfully designed and developed the LoanDapp smart contract with all the functionalities and based on the design diagram
- Verified and validated the deployment of smart contract and end-to-end transactions involved in the Loan process on Remix and Ganache
- LoanPurse-contract folder has been deployed using truffle and required files are created

Phase 4
Security and Events

Security:

Security and privacy are important aspects of any application. Thus, to maintain the privacy of transaction values, I am doing some validations which are required to confirm the valid transactions happening between borrower and lender and required amounts are available to carry out the transactions. Also, I am validating that functionalities to performed by certain user like approve and reject the loan requests can only be done by lender. In the same manner, request loan and repay loan can only be done by borrower/requestor. This provides the security to the Dapp and the ownership of each transactions supposed for the assigned users.

```
60          require(msg.sender == lender);
61          require(msg.value > amt, "Insufficient Balance.");
```

```
103         require(msg.sender.balance >= repayAmt);
104         require(msg.value > repayAmt+1000000000000000000, "Insufficient Balance.");
105
```

Events:

Events are notifications that can be emitted from functions to indicate the presence of a condition or flag during a smart contract function's execution. Solidity provides feature to define and emit an event with and without parameters. Events are logged on-chain, in the receipt tree and can be accessed using their names. Events are used to notify the application the various changes made to the contract.

I have defined below events in my Dapp and where they are emitted in the smart contract:

```
// Events are defined here
event Initialized();
event LoanRequested();
event LoanApproved();
event LoanRejected();
event LoanRepaid();
event myCalc(uint myRepay);
event loanCreditHistory(uint loanCount1, uint loanAmt1);
```

```
        // constructor() is executed when the smart contract is deployed into the network
        // and `msg.sender` which is the lender/owner, is the account creating this contract.
        constructor () public {
            lender = msg.sender;

            emit Initialized();
        }

        // function to request Loan by borrower to Lender by giving few details about the borrower and reason for Loan
        function requestLoan(uint loanAmount) notLender public {
            // add Loan object to mapping
            amount[msg.sender] = loanAmount * 1000000000000000000;
            loans[msg.sender] = Loan(noOfLoan, msg.sender, loanAmount, "Loan Requested");

            emit LoanRequested();
        }
```

```
        // function to approve the loan request by Lender and transfer the loan amount to borrower account
        function approveLoan(address payable requestorAddress) onlyLender payable public {
            require(msg.sender == lender);
            require(msg.value > amt, "Insufficient Balance.");
            amt = amount[requestorAddress];
            requestorAddress.transfer(amt);

            uint tempBalance = 0;
            tempBalance = msg.value - amt;
            msg.sender.transfer(tempBalance);

            noOfLoan++;
            loans[requestorAddress] = Loan(noOfLoan, requestorAddress, amt, "Loan Request Approved");
            emit LoanApproved();
        }

        // function to reject the loan request by Lender
        function rejectLoan(address requestorAddress) onlyLender public {
            require(msg.sender == lender);
            loans[requestorAddress].loanStatus="Loan Request Rejected";

            emit LoanRejected();
        }
```

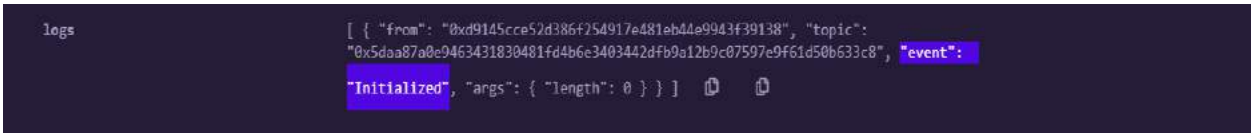
```
         function creditHistory(address requestorAddress) public returns(uint , uint) {
            Loan memory loanDetails = loans[requestorAddress];
            uint loanCount = loanDetails.id;
            uint loanHist = (loanDetails.loanAmount / 1000000000000000000);
            emit loanCreditHistory(loanCount , loanHist);

            return (loanCount, loanHist );
         }

        // Check current Balance of the selected address
        function checkBalance(address addr) view public returns(uint){
            return (addr.balance / 1000000000000000000);

        }

        // function to check whether borrower is able to repay the amount to Lender's account address
        function calculateRepay(address requestorAddress, uint requestedLoanAmount) notLender public returns (uint){
            // Calculating repay amount based on predefined interest rate on principal amount(loanAmount)
            calRepayAmount = (((amount[requestorAddress] *10*2)/100) + requestedLoanAmount * 1000000000000000000)/ 1000000000000000000 ;
            emit myCalc(calRepayAmount);

            return calRepayAmount;
        }
```

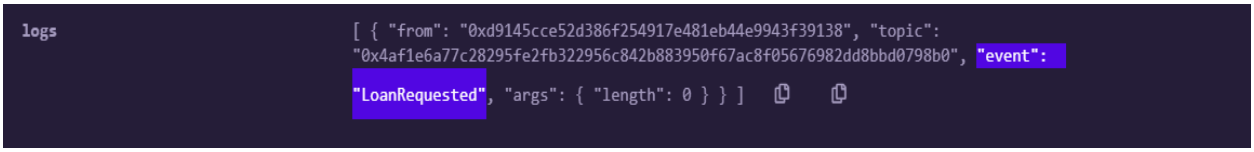
```
         function repayLoan(address payable lenderAddress, uint repayAmt) notLender public payable {
             require(msg.sender.balance >= repayAmt);
             require(msg.value > repayAmt+1000000000000000000, "Insufficient Balance.");
             require(repayAmt == calRepayAmount);

             // transfer the calculated Repay Amount to Lender's account
             lenderAddress.transfer(calRepayAmount * 1000000000000000000);

             // Transfer the remaining amount to borrower's account after deducting the repay amount from msg.value amount
             uint tempBalance = 0;
             tempBalance = msg.value - (calRepayAmount * 1000000000000000000);
             msg.sender.transfer(tempBalance);
             loans[msg.sender].loanStatus = "Loan Amount Paid";

             emit LoanRepaid();
         }
}
```

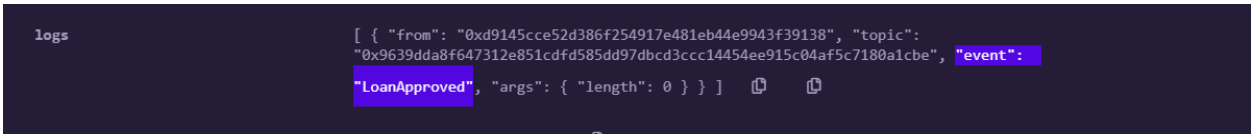
The following are the events used in the smart contract and what they indicate:

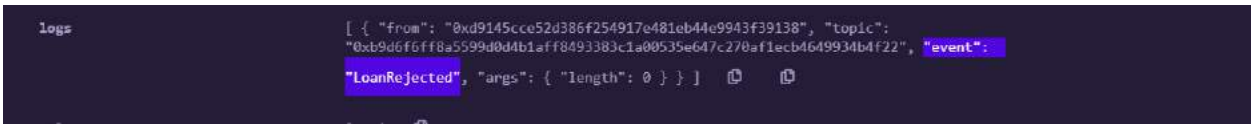
- **event Initialized():** This indicates the loan Dapp process has been initialized and started
- **event LoanRequested():** this indicates the phase where the loan amount has been requested by borrower
- **event LoanApproved():** This indicates the phase where the loan amount request from borrower has been approved by lender and the loan amount transferred
- **event LoanRejected():** This indicates the phase where the loan amount request from borrower has been rejected by lender
- **event LoanRepaid():** This indicates the phase where the loan amount has been paid back by borrower to the lender
- **event myCalc(uint myRepay):** This indicates the phase where the repayLoan amount is calculated and returned to show the calculated repay amount to the Requestor
- **event loanCreditHistory(uint loanCount1, uint loanAmt1):** This indicates the phase where the loan credit history is shown for the requestor like the number of previous loan counts and the last loan amount approved for the Requestor

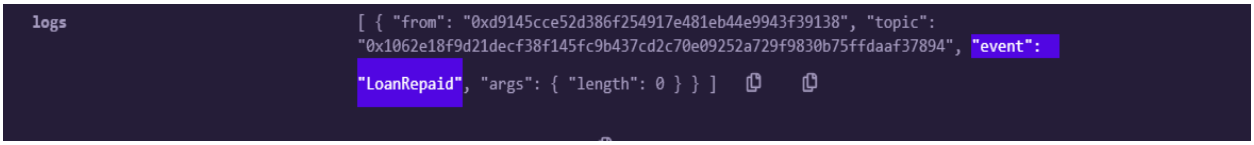
Screenshots of different event logs are shown below:

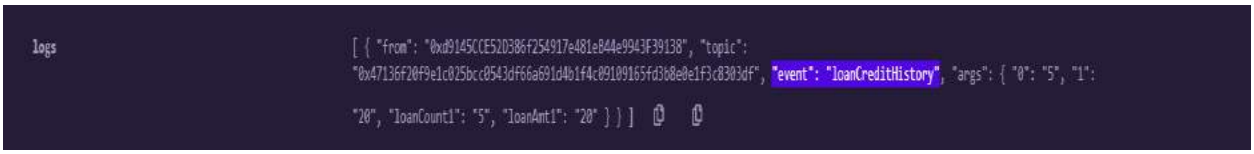










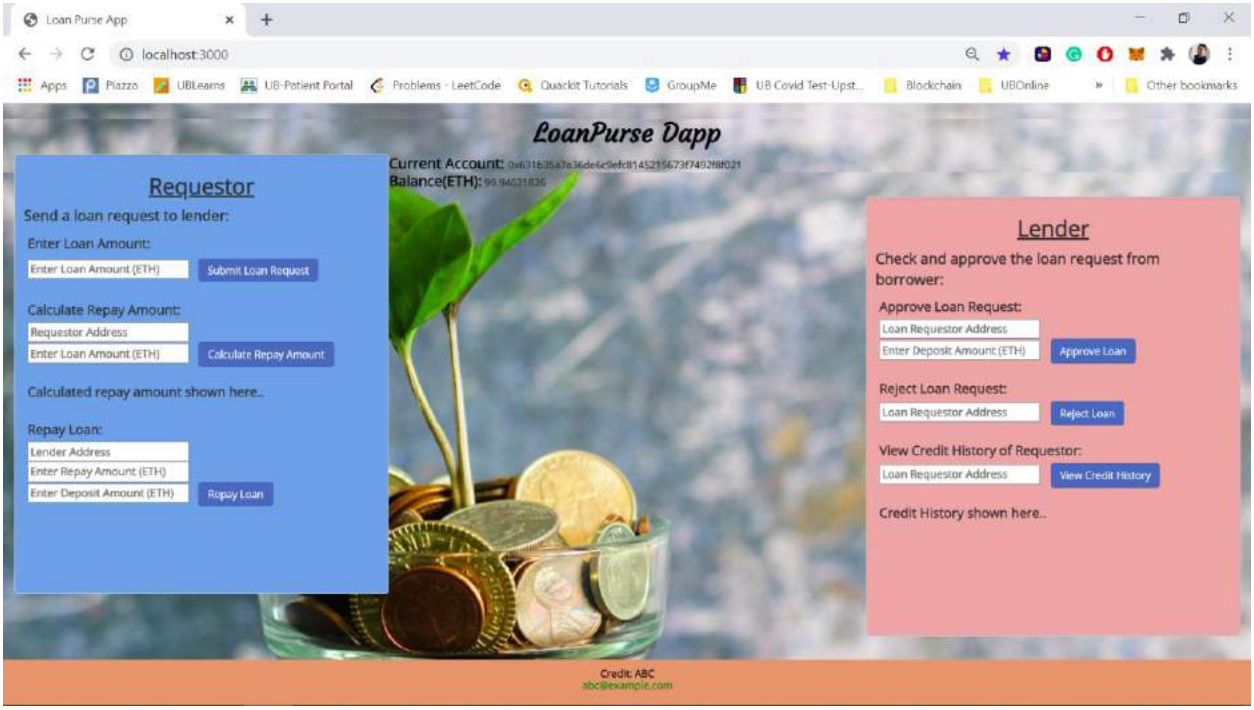


Phase 4 Conclusion and Outcome:

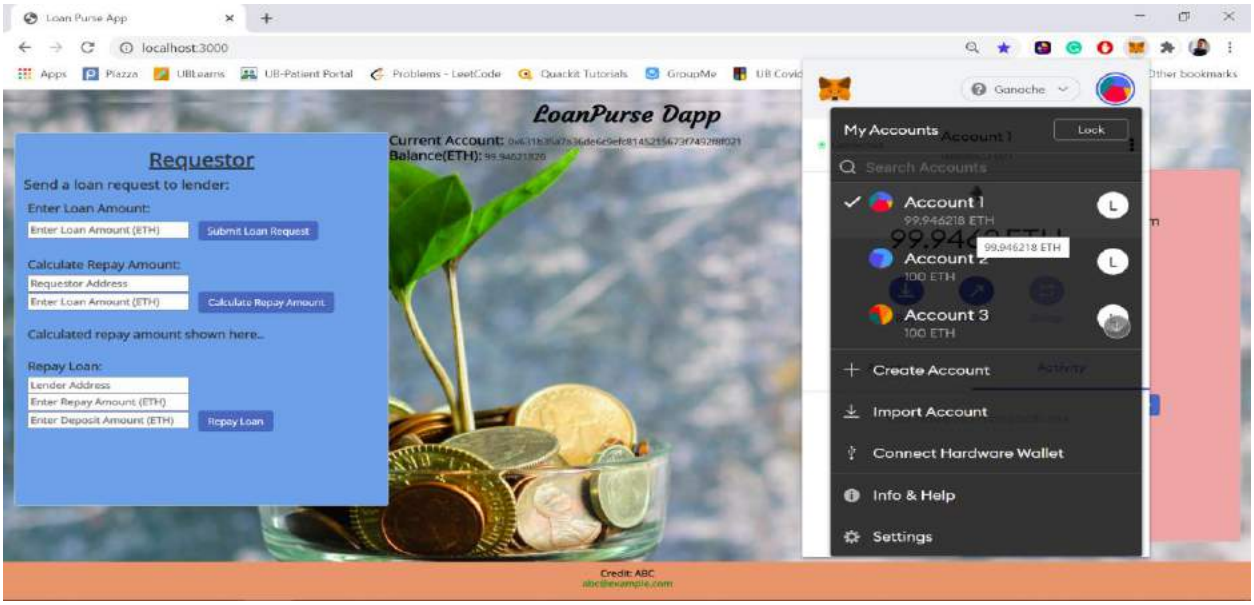
- Successfully implemented security, privacy and events in the smart contract to make the Dapp more secured.
- Verified and validated the deployment of smart contract and end-to-end transactions involved in the Loan process on Remix and Ganache.
- LoanPurse-contract folder has been deployed using truffle and required files are created.

Phase 5
Web App Development (Dapp)

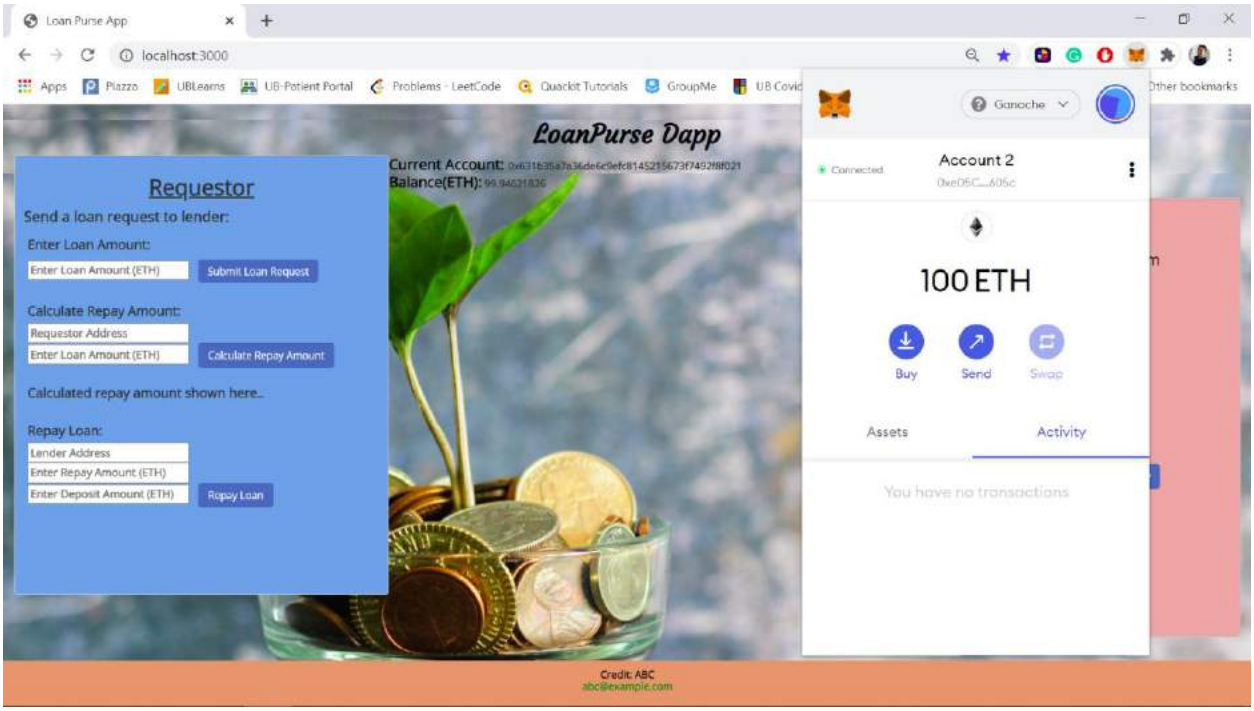
The below screenshot shows the web part of the LoanPurse Dapp deployed on localhost server using Ganache. The web application consists of all the required fields as part of loan request and approval process. The corresponding fields to each of the stakeholders like Requestor and Lender are shown categorically separated as per their requirements.

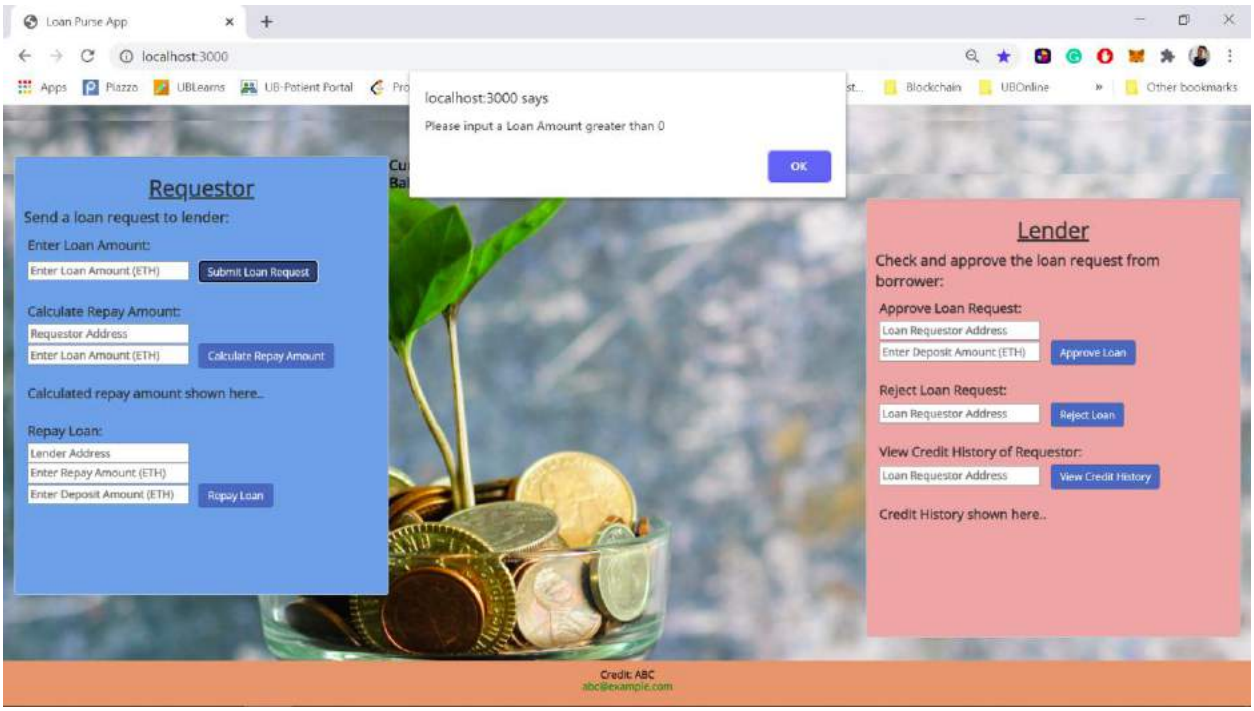


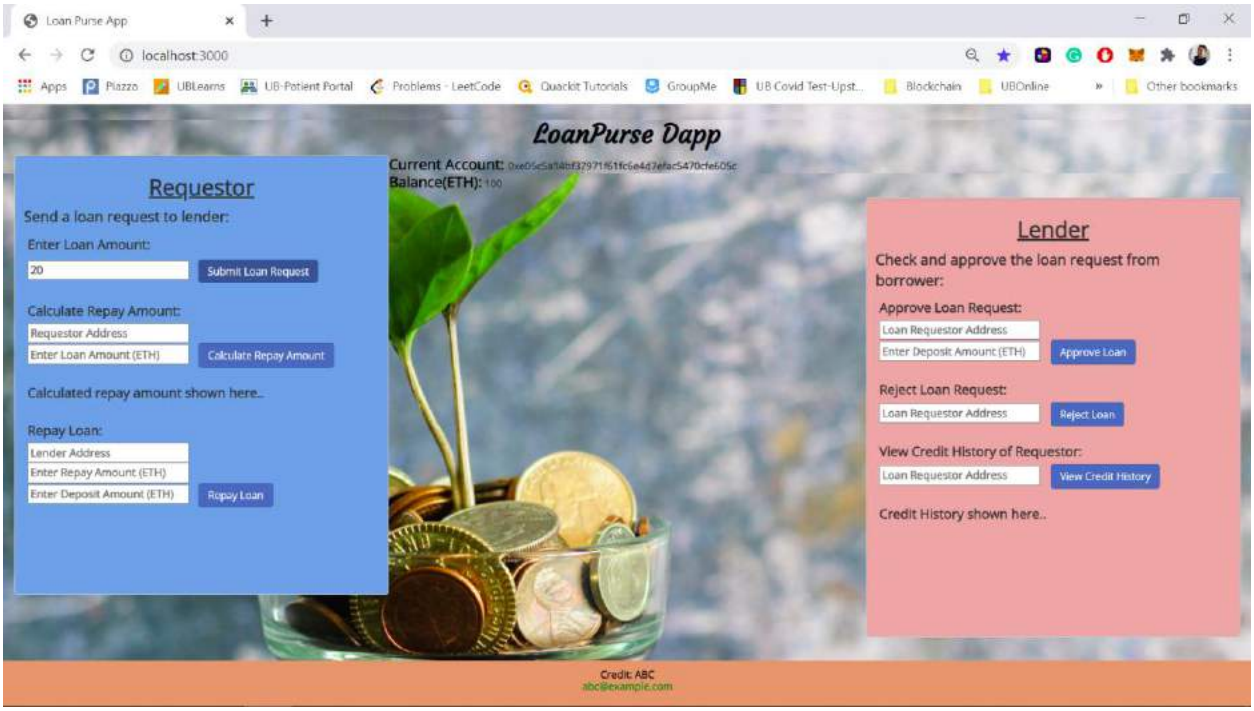
This screenshot shows that all the account has been connected with the web app, and the smart contract on web app was deployed by the 1st account(Smart Contract Owner) in Ganache for which small transaction charges are deducted.

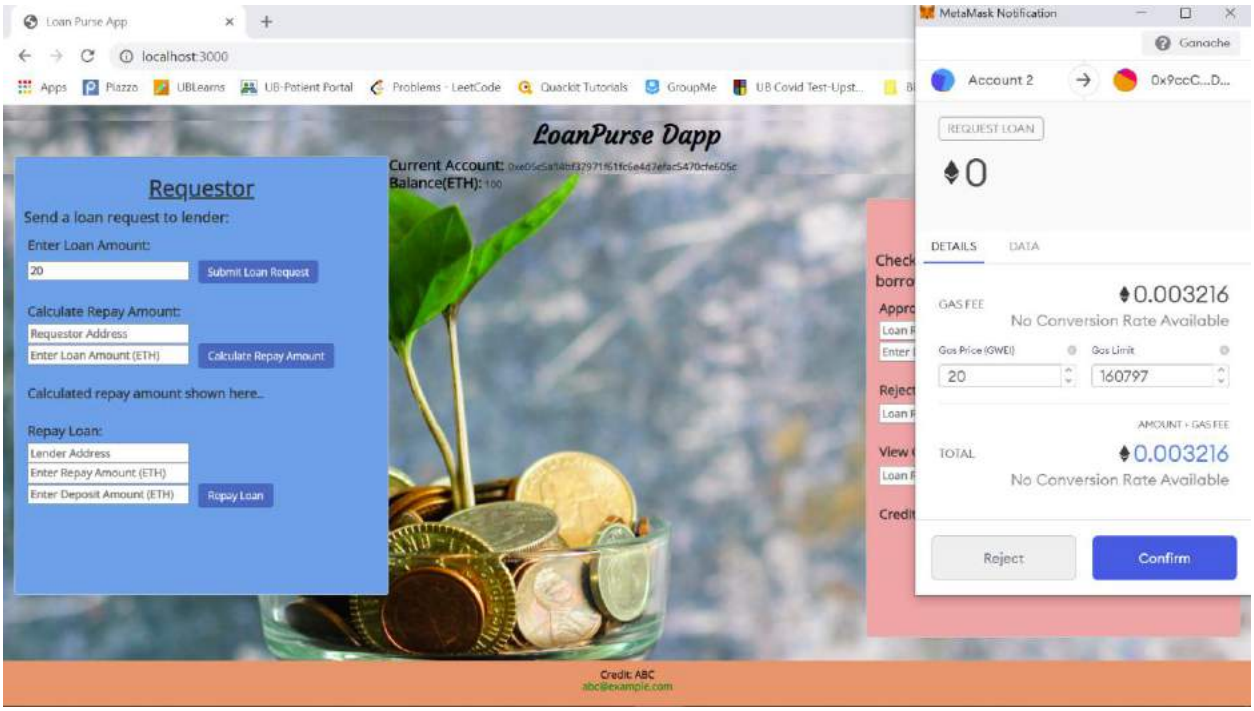


Account 2(Requestor) requests for the loan amount to the Lender by entering the required loan amount in the field and submitting the loan request.

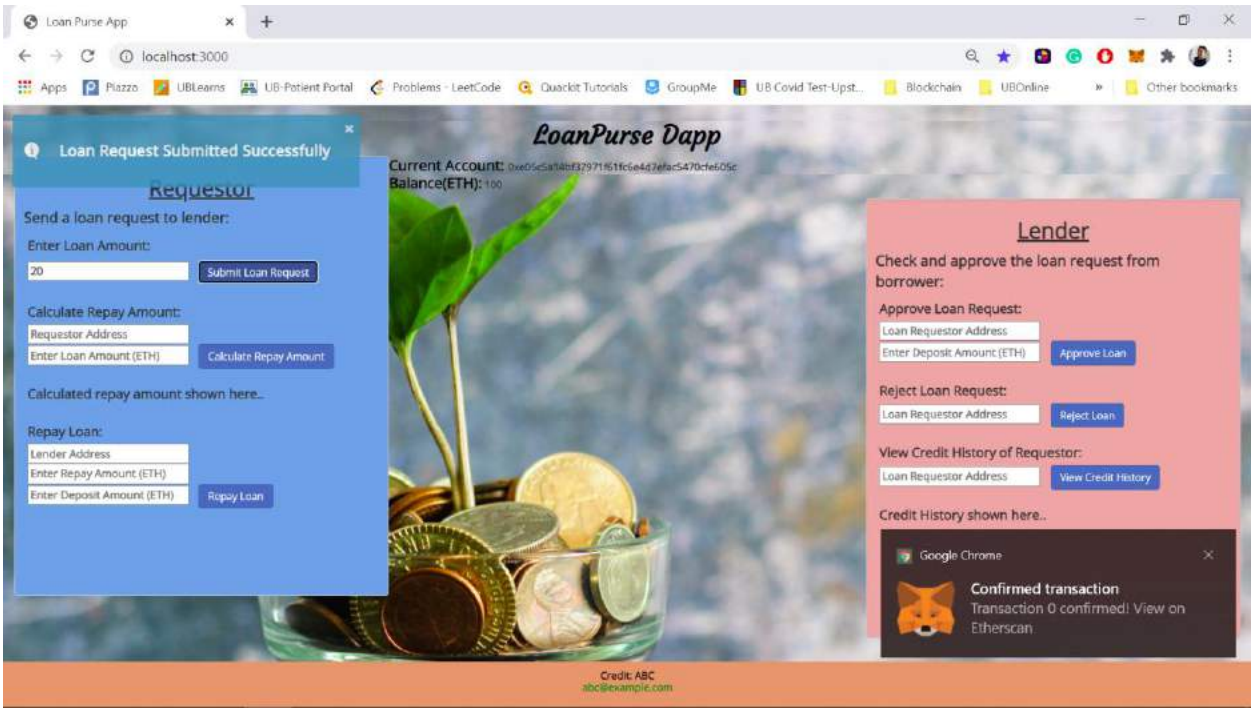



Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Blockchain
UBOnline
Other bookmarks
localhost:3000 says
Please input a Loan Amount greater than 0
OK
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
Requestor Address
Enter Loan Amount (ETH)
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

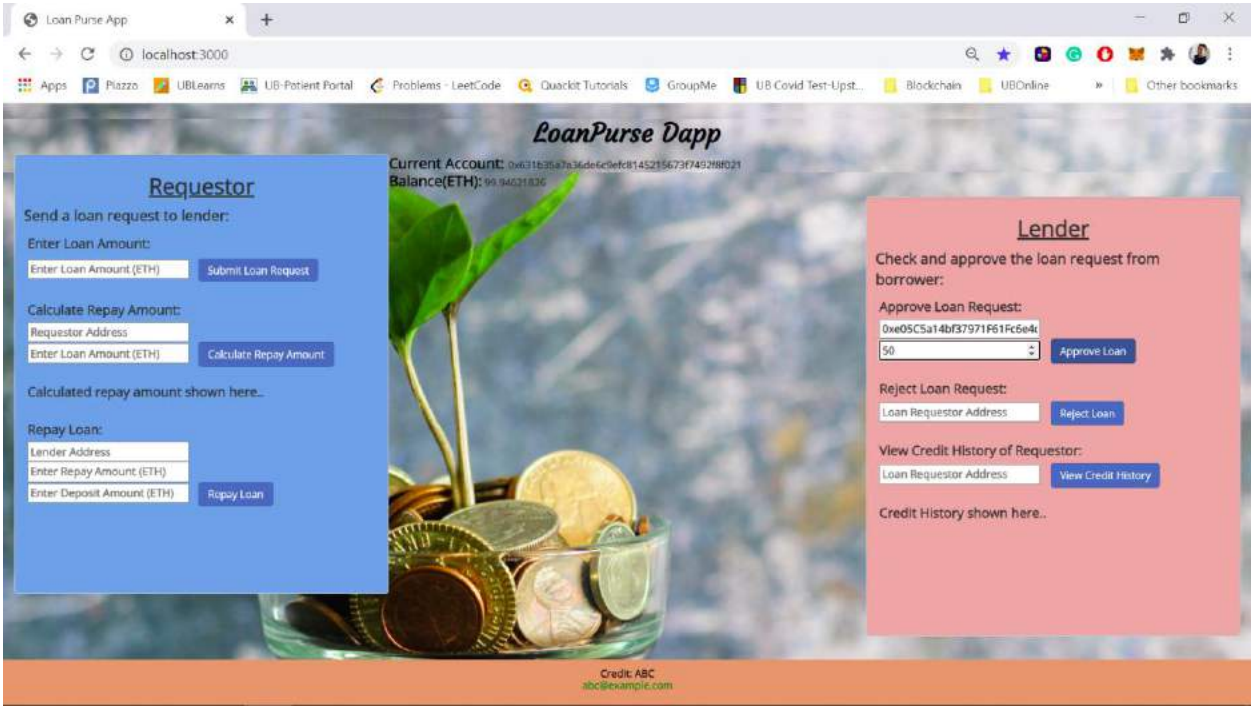

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quackit Tutorials
GroupMe
UB Covid Test-Upst...
Blockchain
UBOnline
Other bookmarks
LoanPurse Dapp
Current Account:
Balance(ETH): 100
Requestor
Send a loan request to lender:
Enter Loan Amount:
20
Submit Loan Request
Calculate Repay Amount:
Requestor Address
Enter Loan Amount (ETH)
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

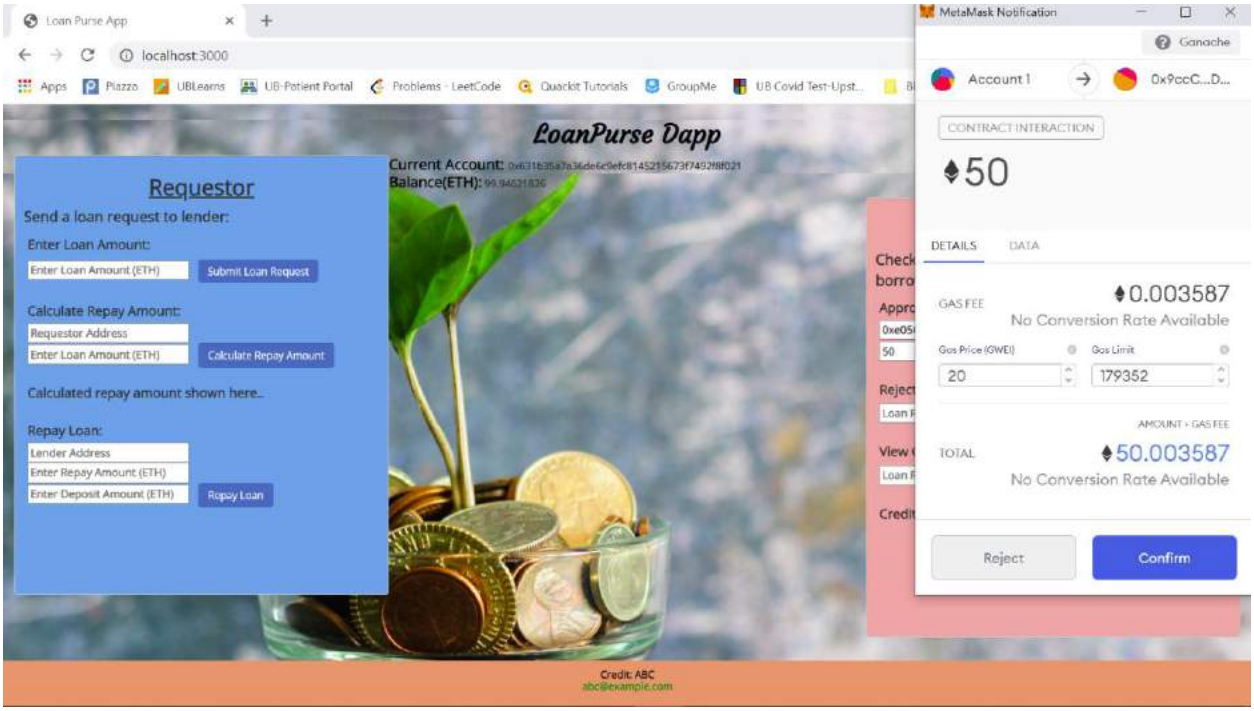


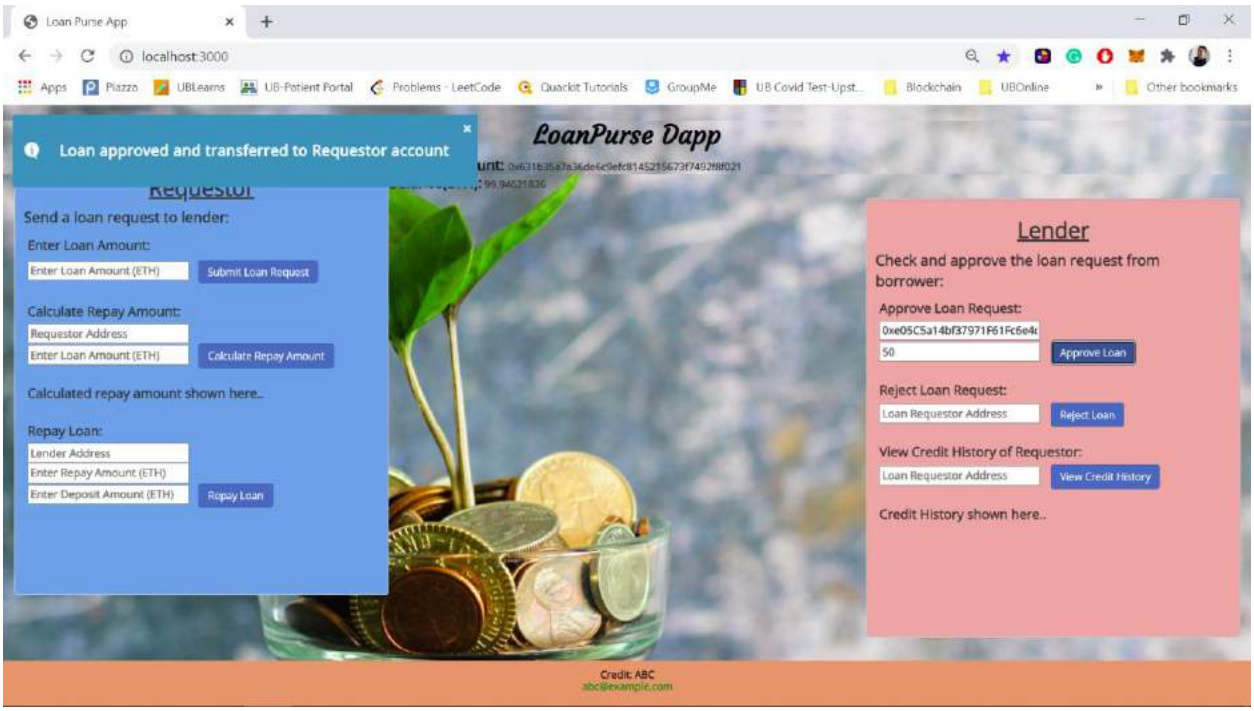
Toast message shows that the loan request was successful.

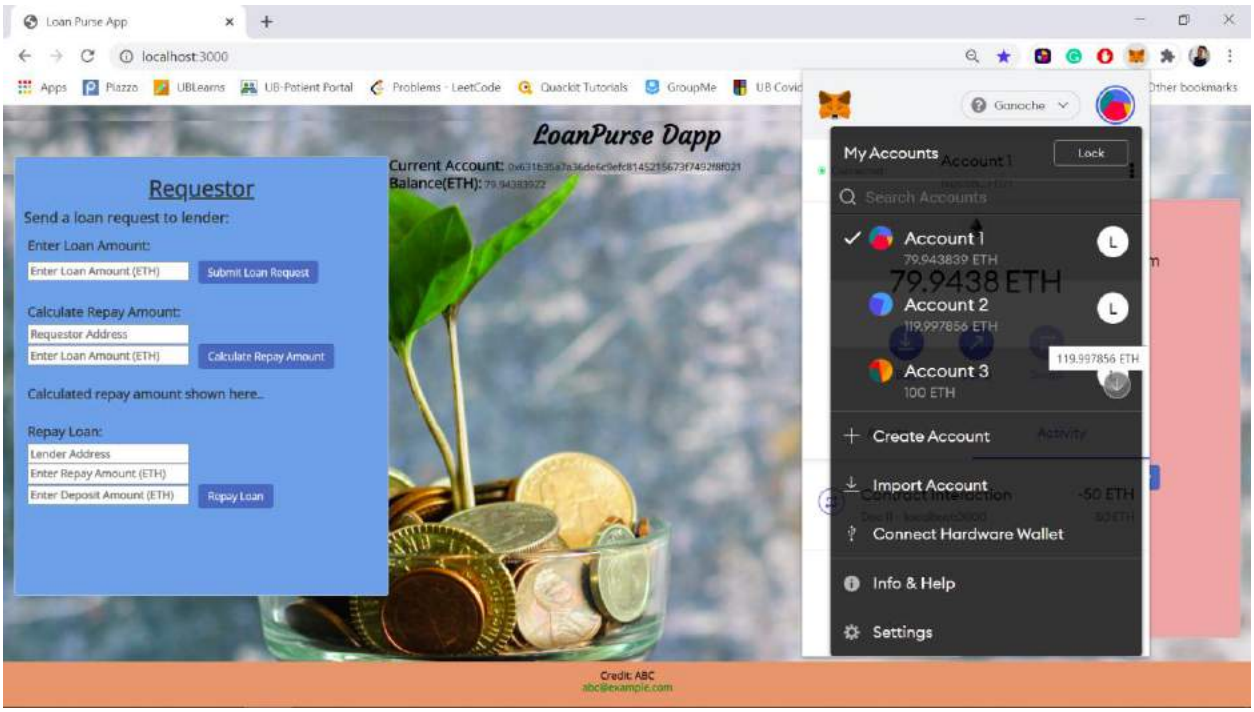


Account 1 (Lender) approves the loan request from the Requestor. Once the Loan request is approved, the loan amount gets transferred to the Requestors' account as shown below.

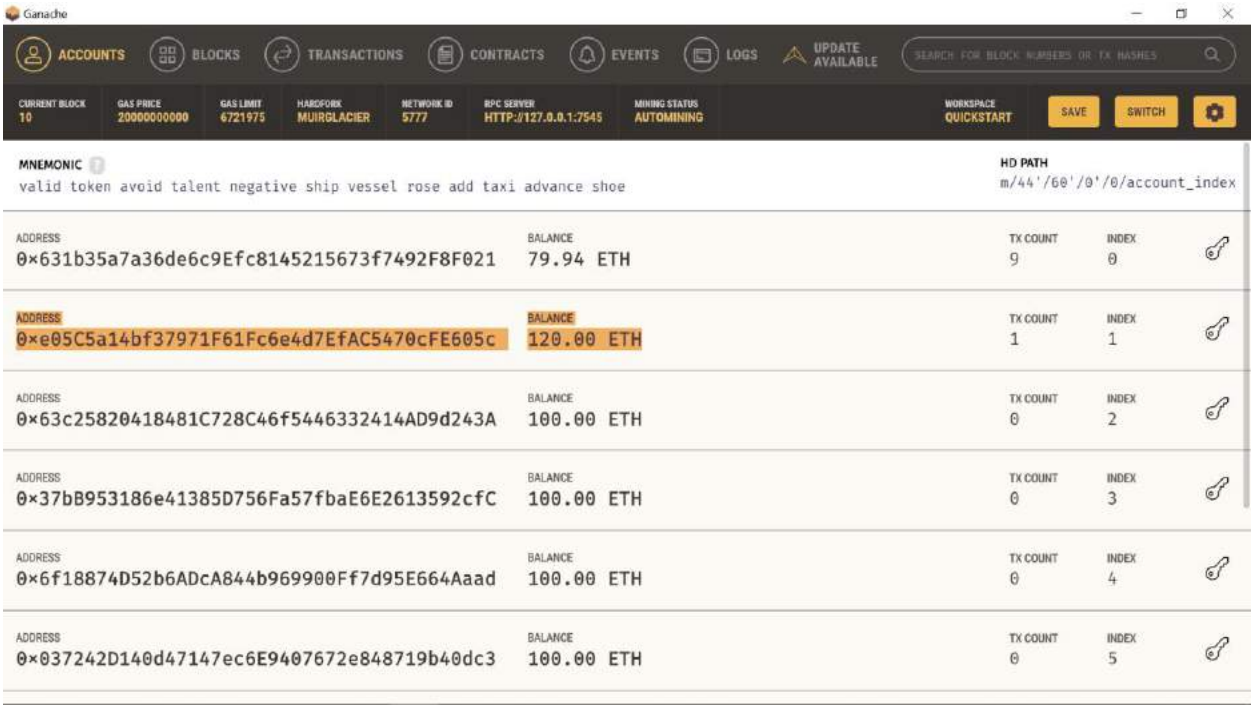




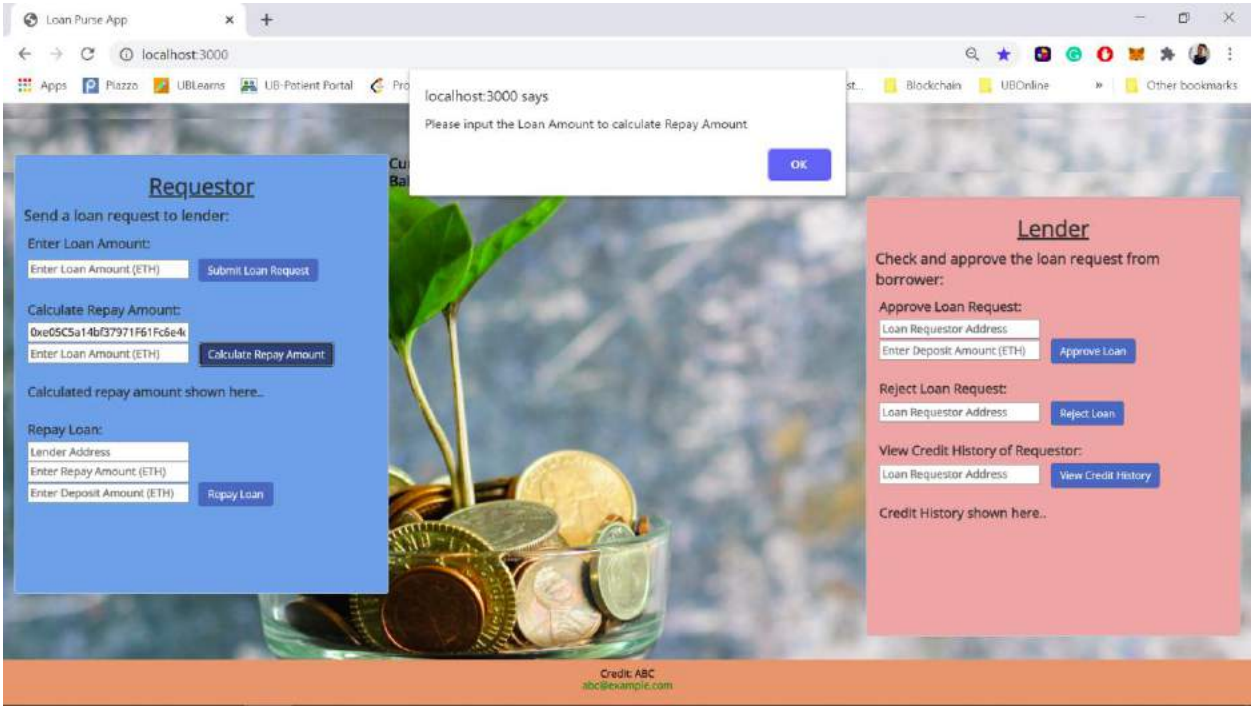

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quacklt Tutorials
GroupMe
UB Covid Test-Upst...
Blockchain
UBOnline
Other bookmarks
Loan approved and transferred to Requestor account
LoanPurse Dapp
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
Requestor Address
Enter Loan Amount (ETH)
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
0xe05C5a14bf37971F61Fc6e4
50
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

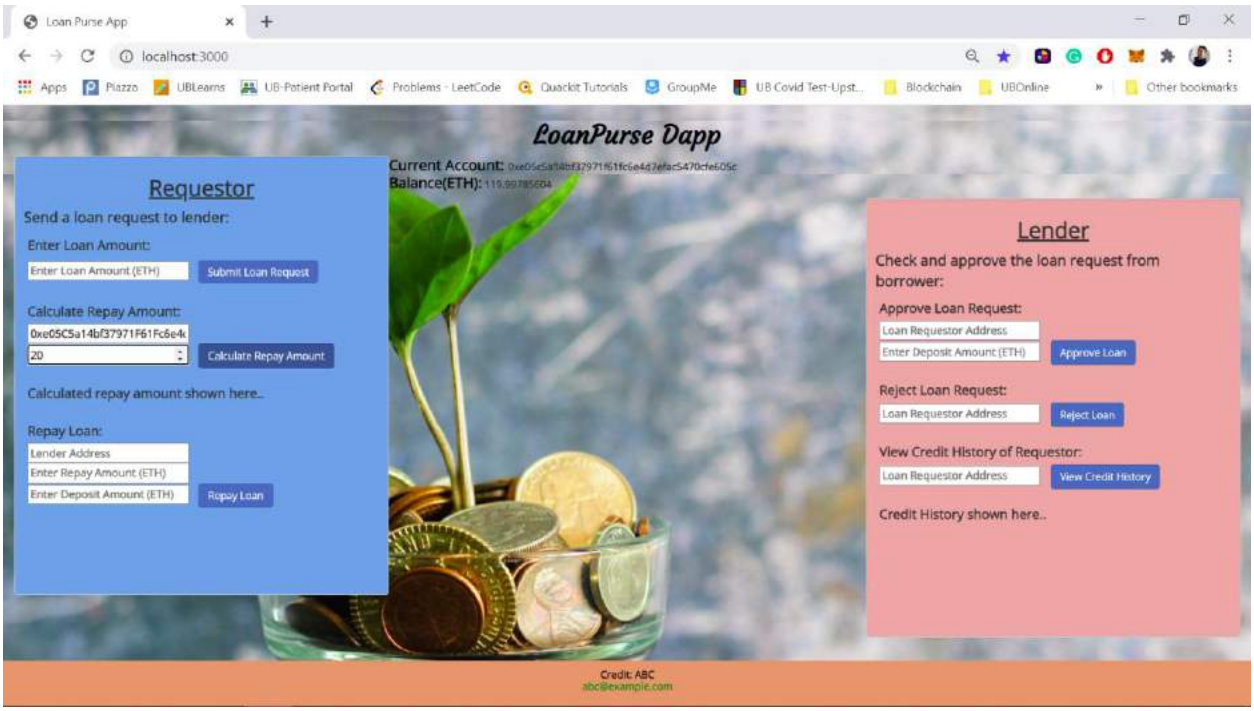

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quacklt Tutorials
GroupMe
Other bookmarks
LoanPurse Dapp
Current Account:
Balance(ETH):
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
Requestor Address
Enter Loan Amount (ETH)
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Ganache
My Accounts
Lock
Search Accounts
Account 1
79.943839 ETH
79.9438 ETH
Account 2
119.997856 ETH
119.997856 ETH
Account 3
100 ETH
Create Account
Import Account
Connect Hardware Wallet
Info & Help
Settings
Credit: ABC
abc@example.com

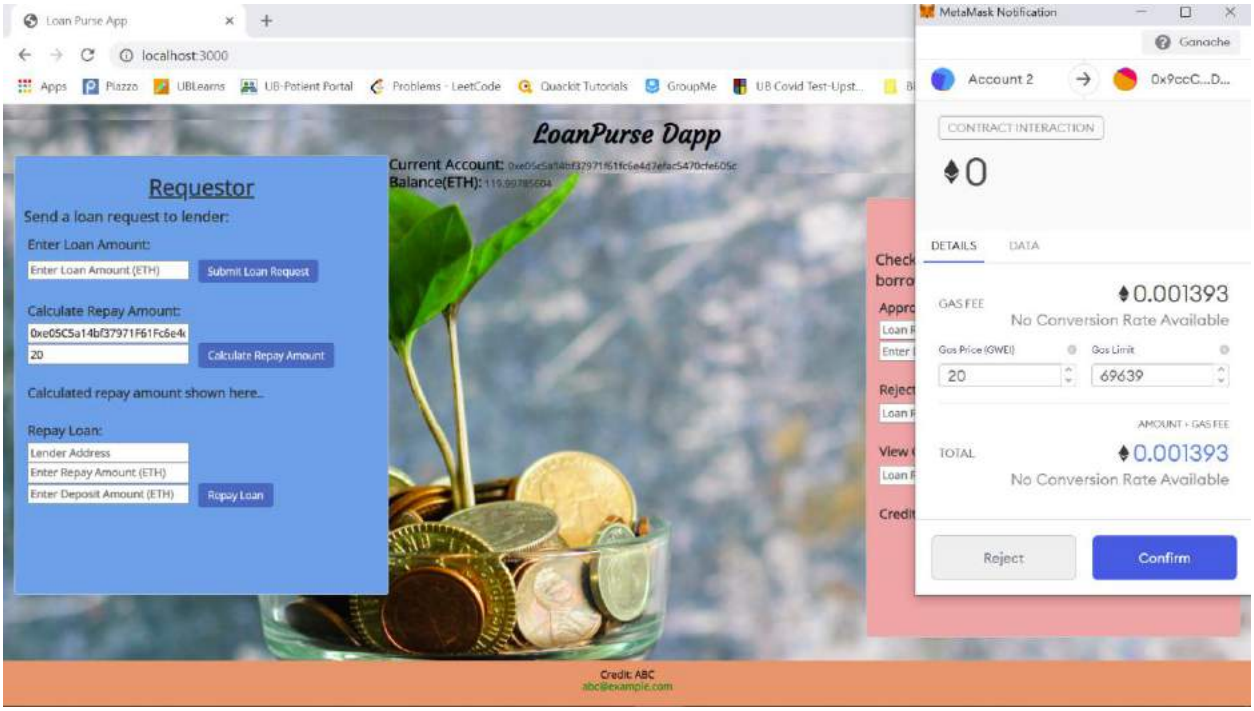
Account 2(Requestor) receives the loan amount to its account.

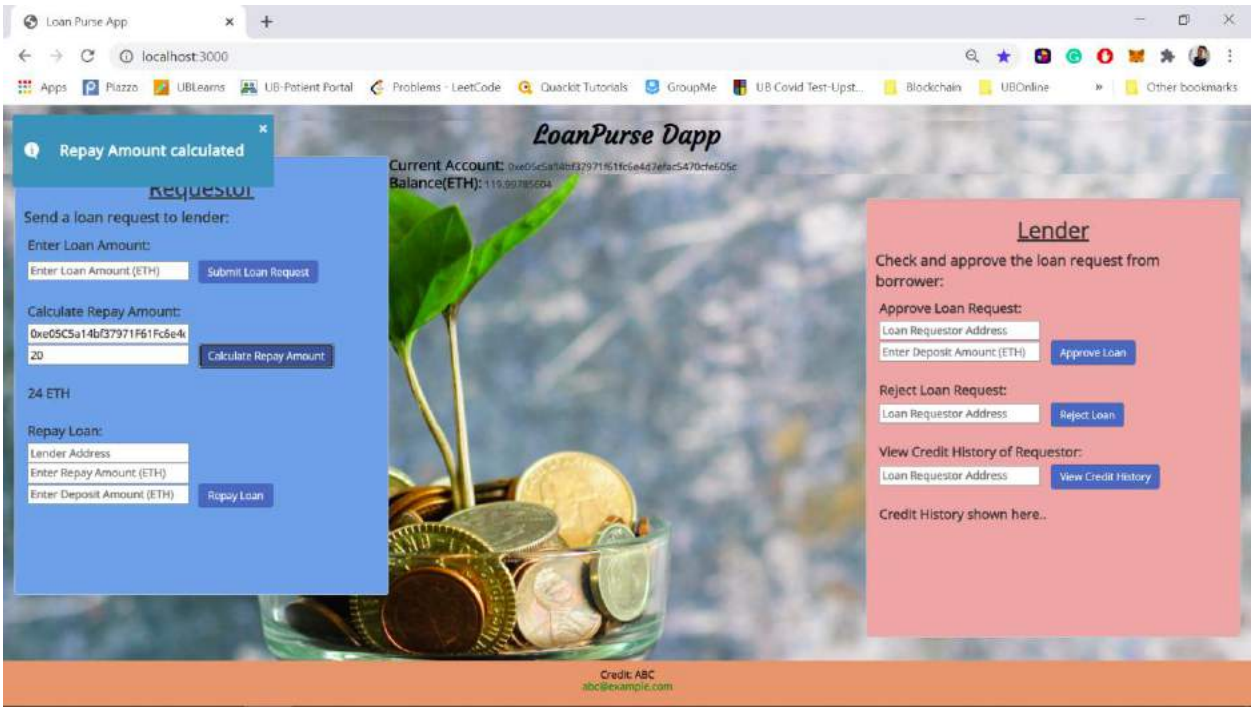


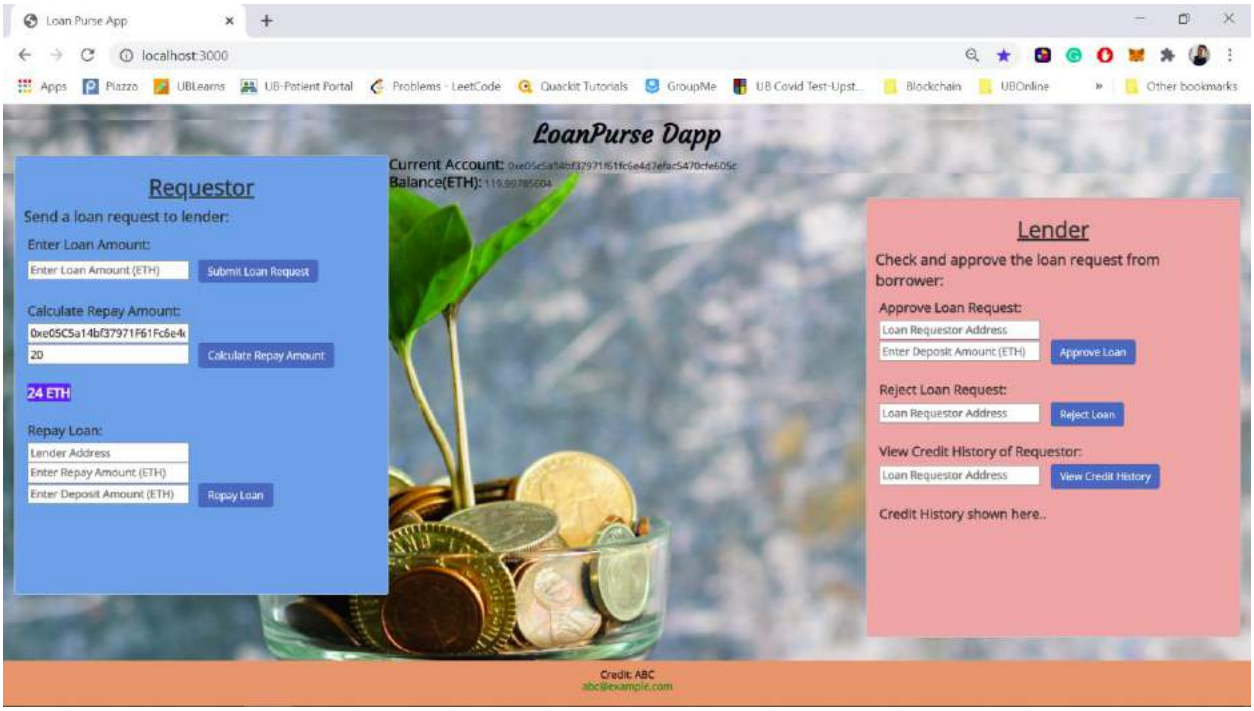
Account 2(Requestor) calculates the repay amount to repay the taken loan.



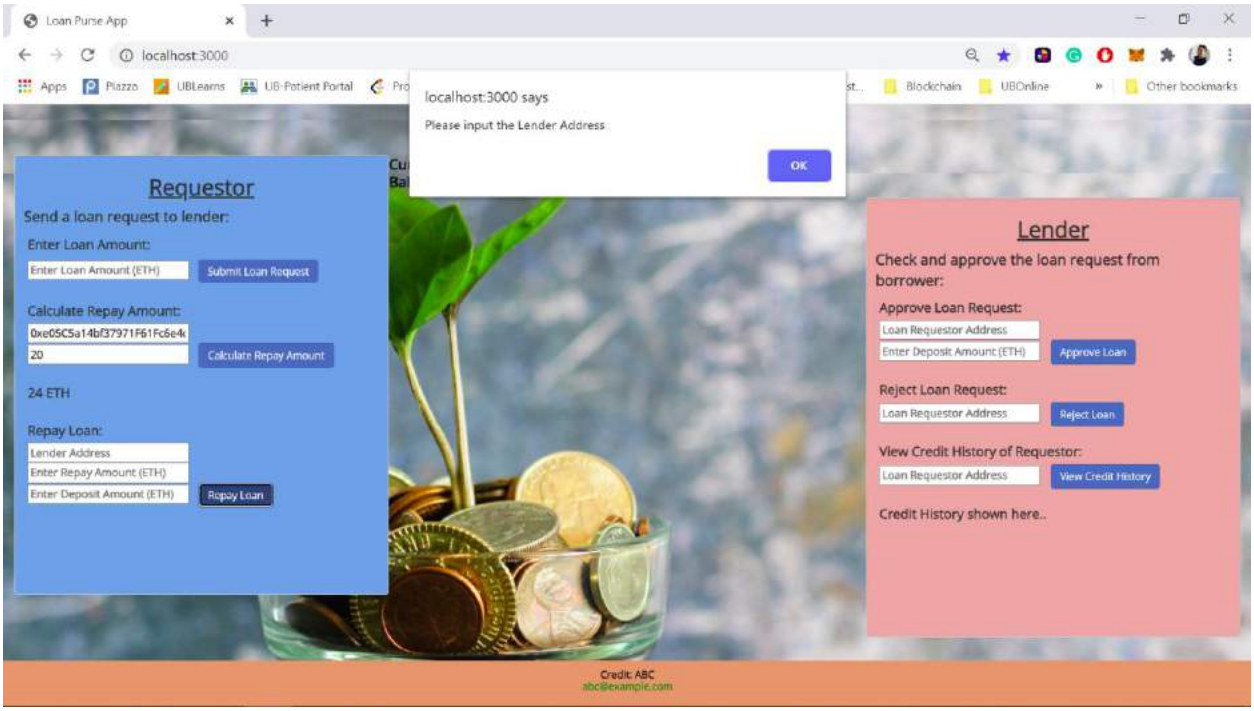

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quacklt Tutorials
GroupMe
UB Covid Test-Upst...
Blockchain
UBOnline
Other bookmarks
LoanPurse Dapp
Current Account:
Balance(ETH):
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0xe05C5a14bf37971F61Fc6e4
20
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

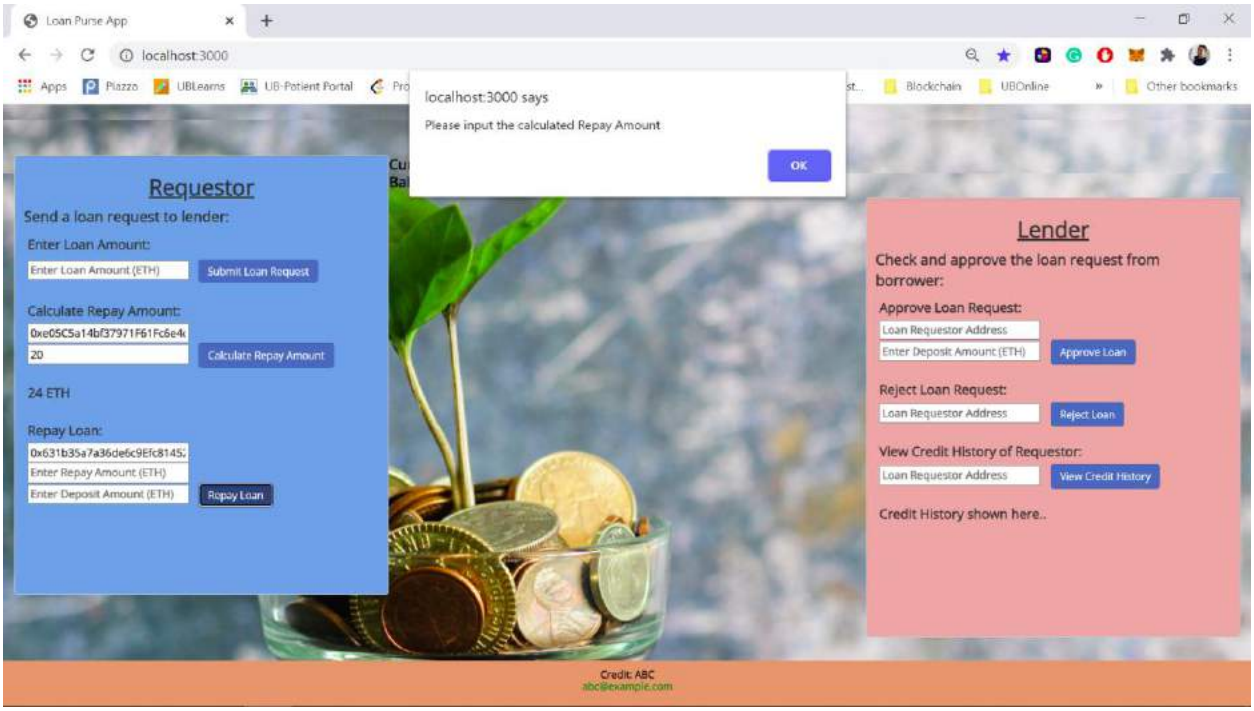

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quacklt Tutorials
GroupMe
UB Covid Test-Upst...
LoanPurse Dapp
Current Account:
Balance(ETH):
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0xe05C5a14bf37971F61Fc6e4
20
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
MetaMask Notification
Ganache
Account 2
0x9ccC...D...
CONTRACT INTERACTION
0
DETAILS
DATA
GAS FEE
0.001393
No Conversion Rate Available
Gas Price (GWEI)
Gas Limit
20
69639
AMOUNT + GAS FEE
TOTAL
0.001393
No Conversion Rate Available
Reject
Confirm
Credit: ABC
abc@example.com

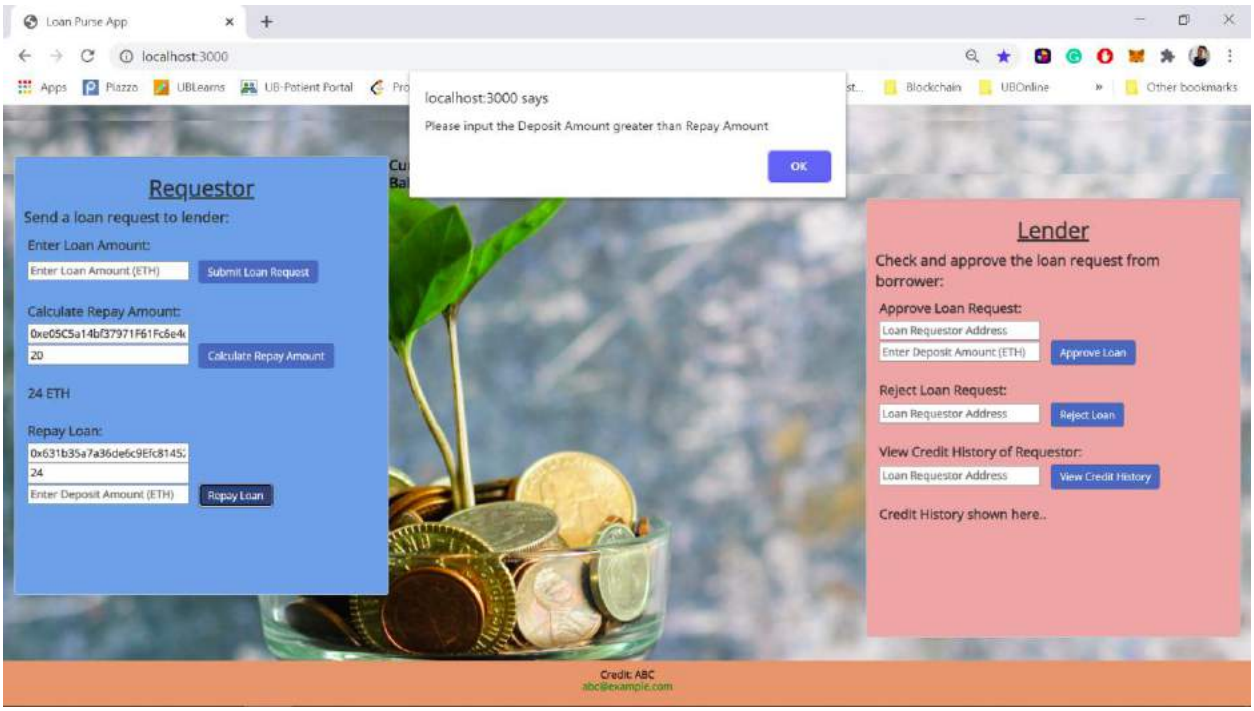

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quacklt Tutorials
GroupMe
UB Covid Test-Upst...
Blockchain
UBOnline
Other bookmarks
Repay Amount calculated
LoanPurse Dapp
Current Account:
Balance(ETH):
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0xe05C5a14bf37971F61Fc6e4
20
Calculate Repay Amount
24 ETH
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

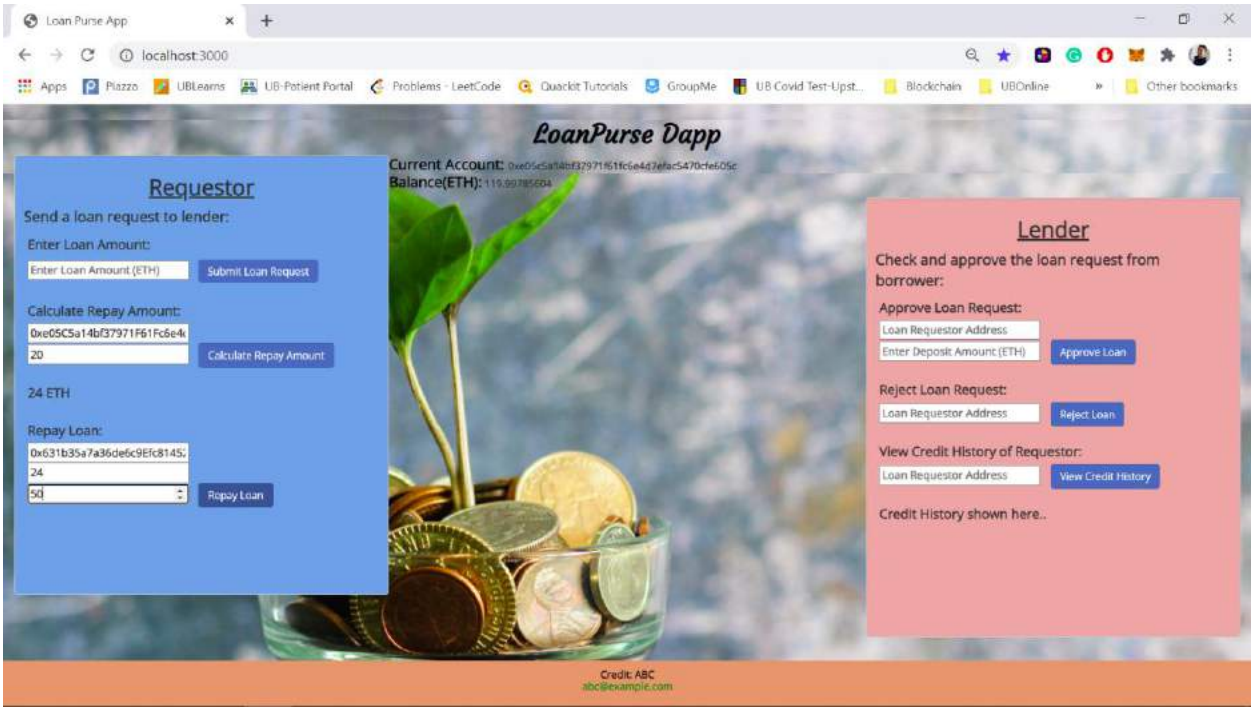

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quacklt Tutorials
GroupMe
UB Covid Test-Upst...
Blockchain
UBOnline
Other bookmarks
LoanPurse Dapp
Current Account:
Balance(ETH):
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0xe05C5a14bf37971F61Fc6e4
20
Calculate Repay Amount
24 ETH
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

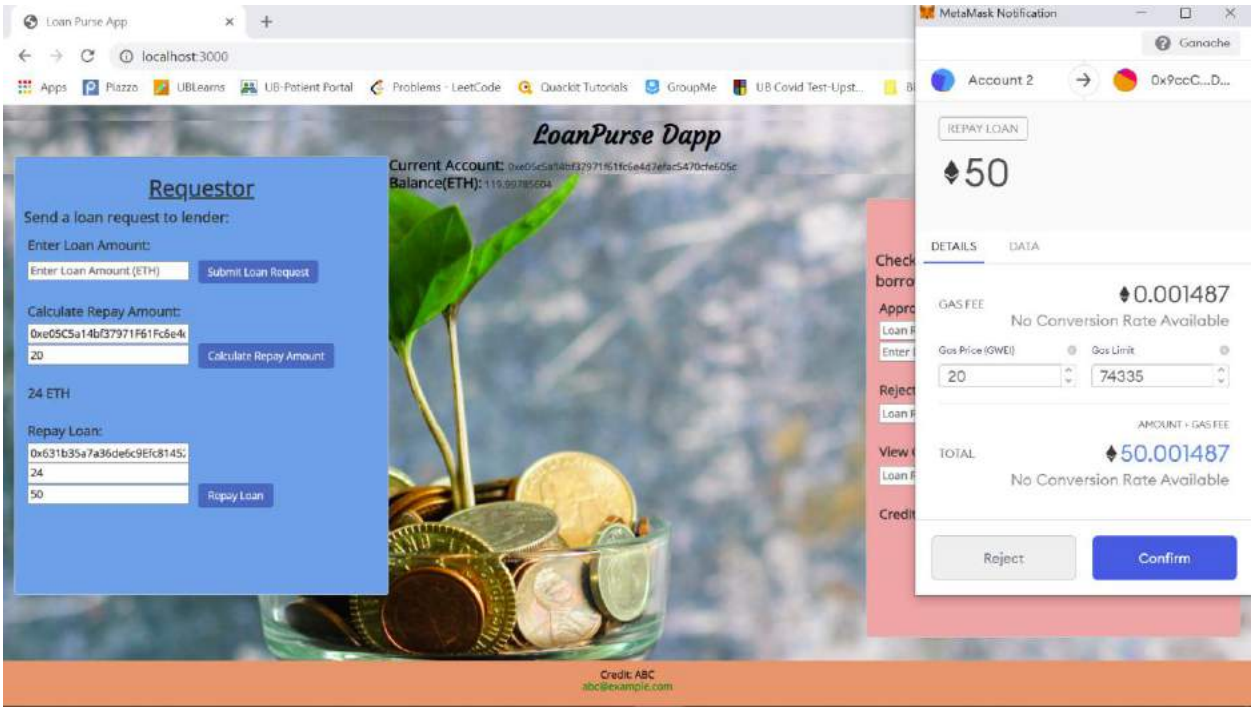
Account 2(Requestor) repays the loan amount to Account 1(Lender).

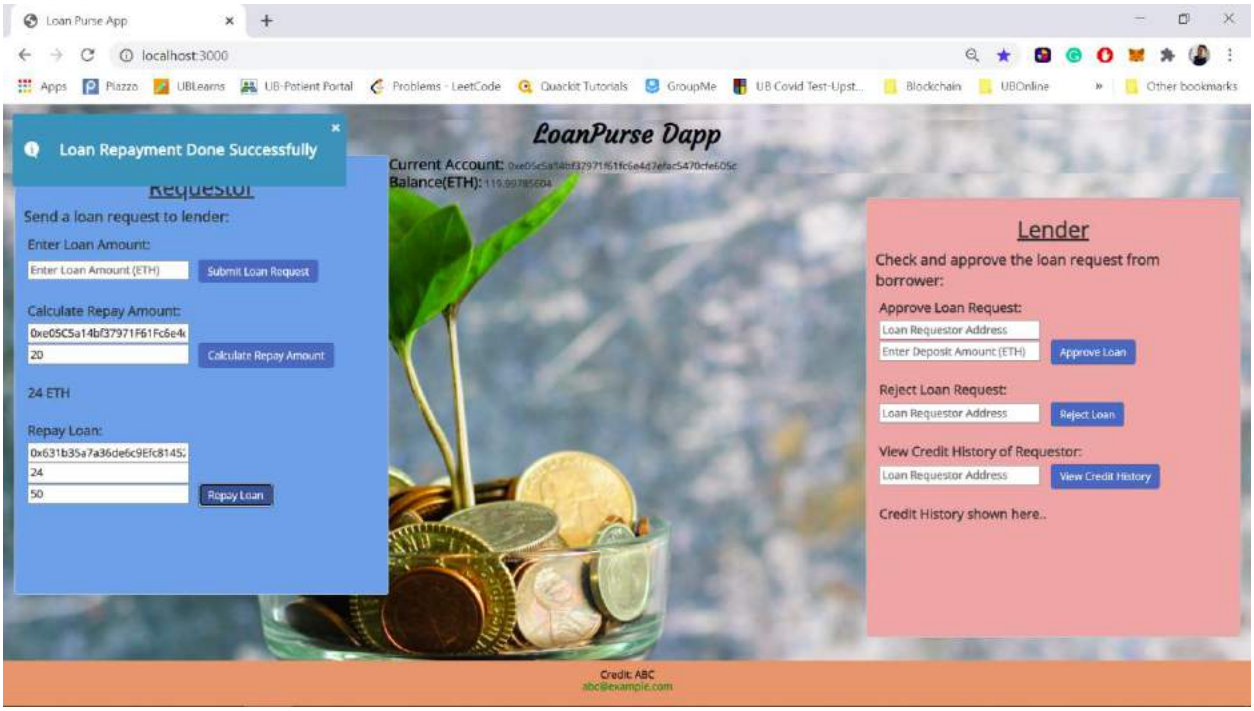




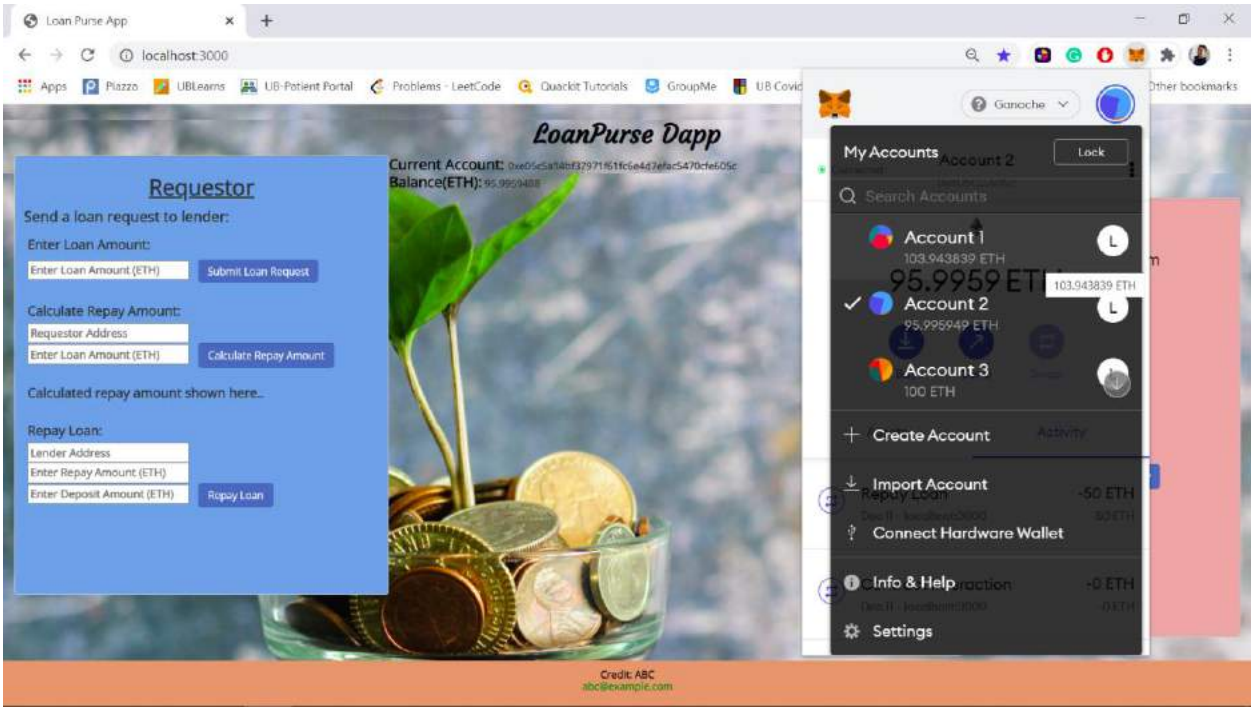

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Blockchain
UBOnline
Other bookmarks
localhost:3000 says
Please input the Deposit Amount greater than Repay Amount
OK
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0xe05C5a14bf37971F61Fc6e4
20
Calculate Repay Amount
24 ETH
Repay Loan:
0x631b35a7a36de6c9Efc8145
24
Enter Deposit Amount (ETH)
Repay Loan
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

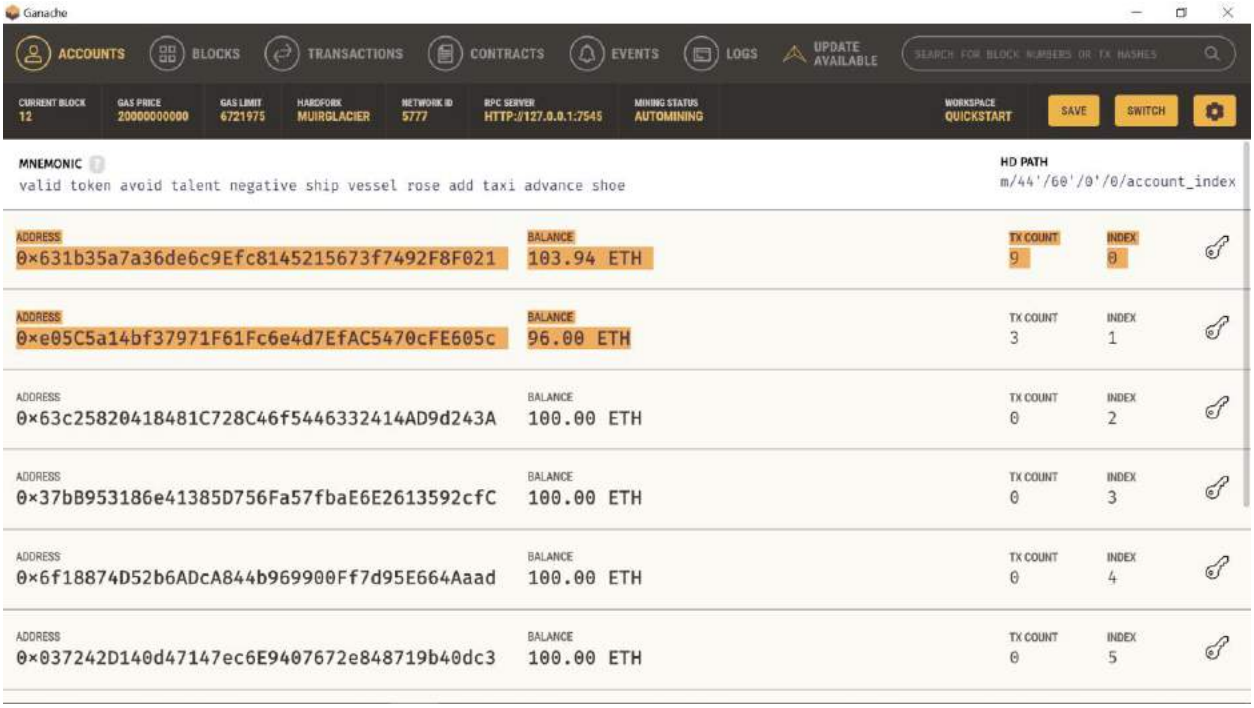

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quackit Tutorials
GroupMe
UB Covid Test-Upst...
Blockchain
UBOnline
Other bookmarks
LoanPurse Dapp
Current Account:
Balance(ETH):
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0xe05C5a14bf37971F61Fc6e4
20
Calculate Repay Amount
24 ETH
Repay Loan:
0x631b35a7a36de6c9Efc8145
24
50
Repay Loan
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com


Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quacklt Tutorials
GroupMe
UB Covid Test-Upst...
LoanPurse Dapp
Current Account:
Balance(ETH):
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0xe05C5a14bf37971F61Fc6e4
20
Calculate Repay Amount
24 ETH
Repay Loan:
0x631b35a7a36de6c9Efc8145
24
50
Repay Loan
MetaMask Notification
Ganache
Account 2
0x9ccC...D...
REPAY LOAN
50
DETAILS
DATA
GAS FEE
0.001487
No Conversion Rate Available
Gas Price (GWEI)
Gas Limit
20
74335
AMOUNT + GAS FEE
TOTAL
50.001487
No Conversion Rate Available
Reject
Confirm
Credit: ABC
abc@example.com

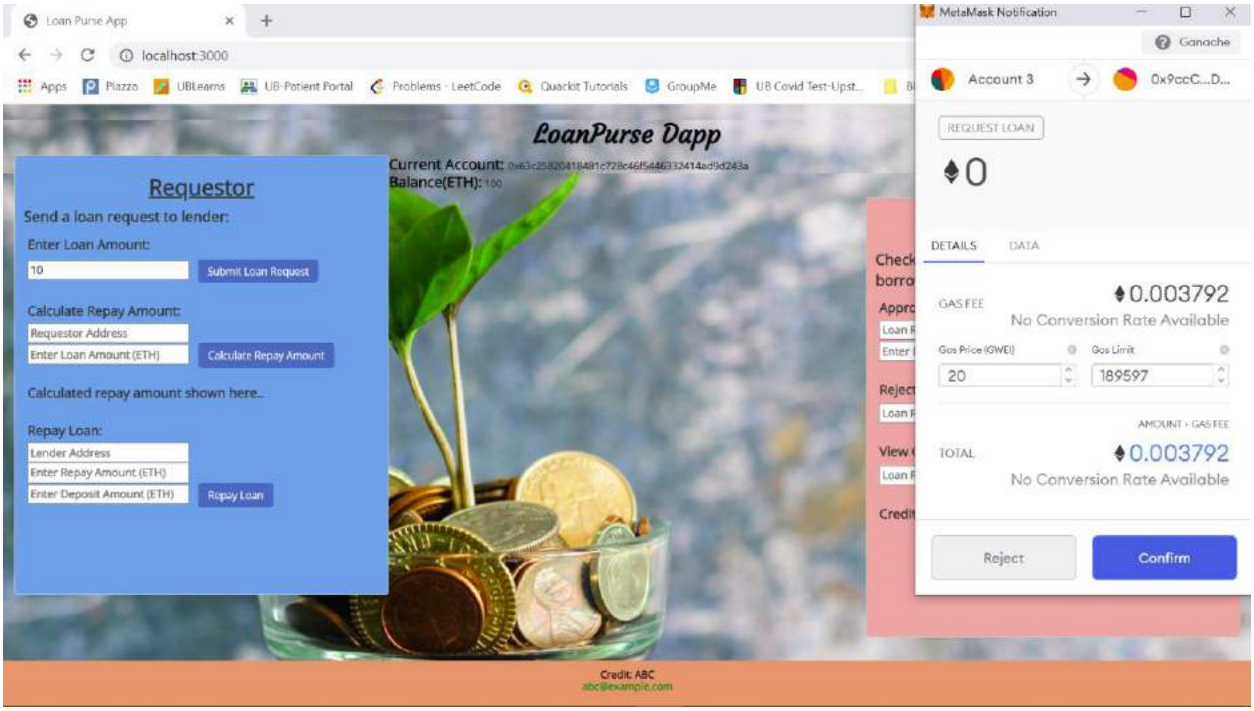

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
UB-Patient Portal
Problems - LeetCode
Quacklt Tutorials
GroupMe
UB Covid Test-Upst...
Blockchain
UBOnline
Other bookmarks
Loan Repayment Done Successfully
LoanPurse Dapp
Current Account:
Balance(ETH):
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0xe05C5a14bf37971F61Fc6e4
20
Calculate Repay Amount
24 ETH
Repay Loan:
0x631b35a7a36de6c9Efc8145
24
50
Repay Loan
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

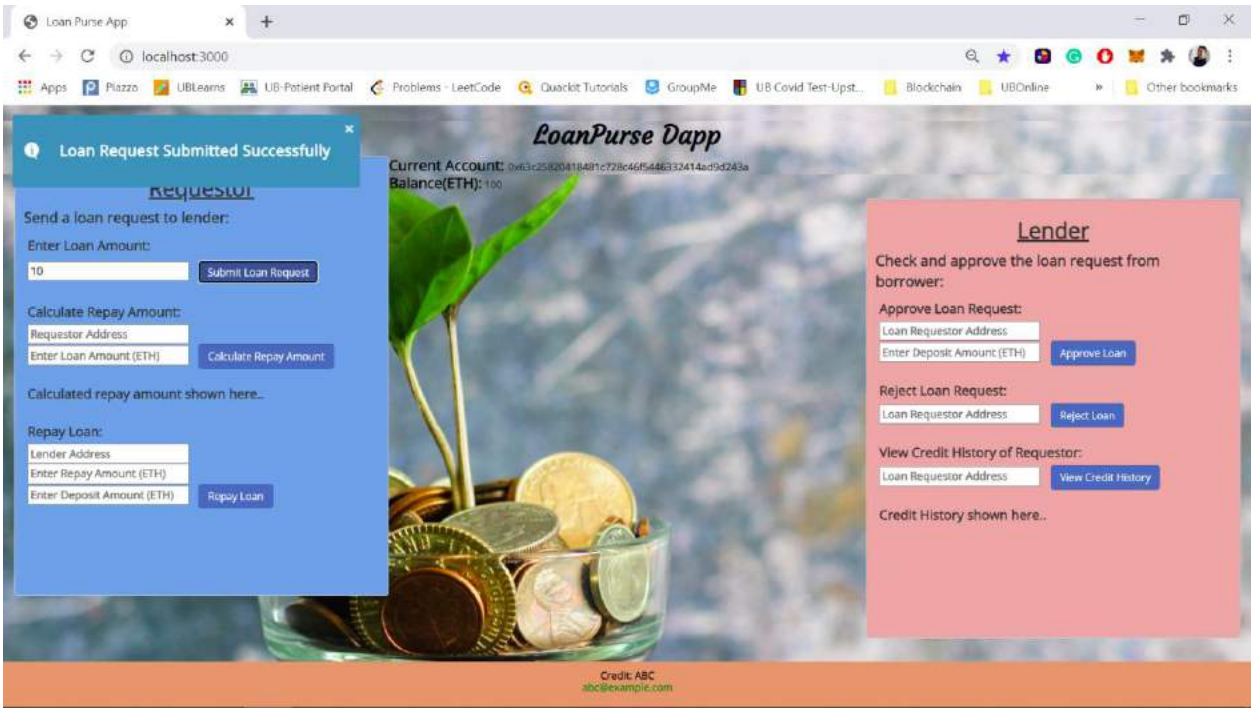
Account balances of Requestor and Lender after Loan Repayment is done successfully (24 ETH is deducted from the Requestors' account and transferred to Lenders' account as Loan repayment).



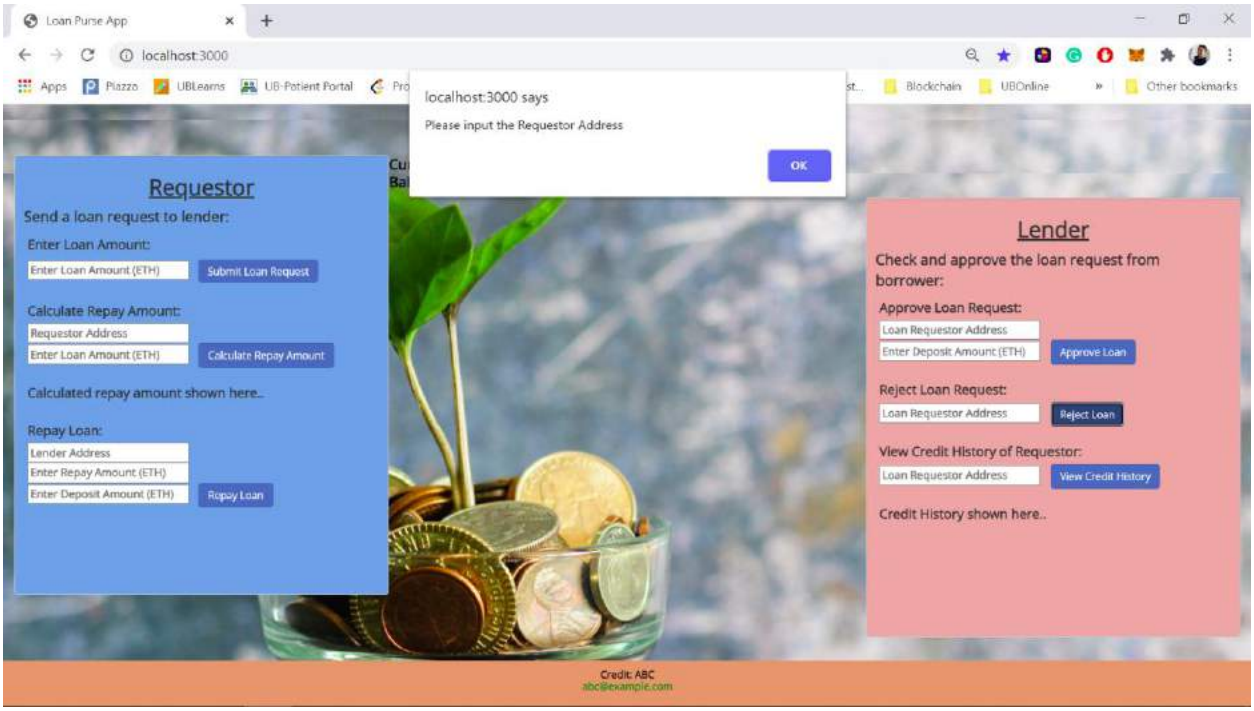


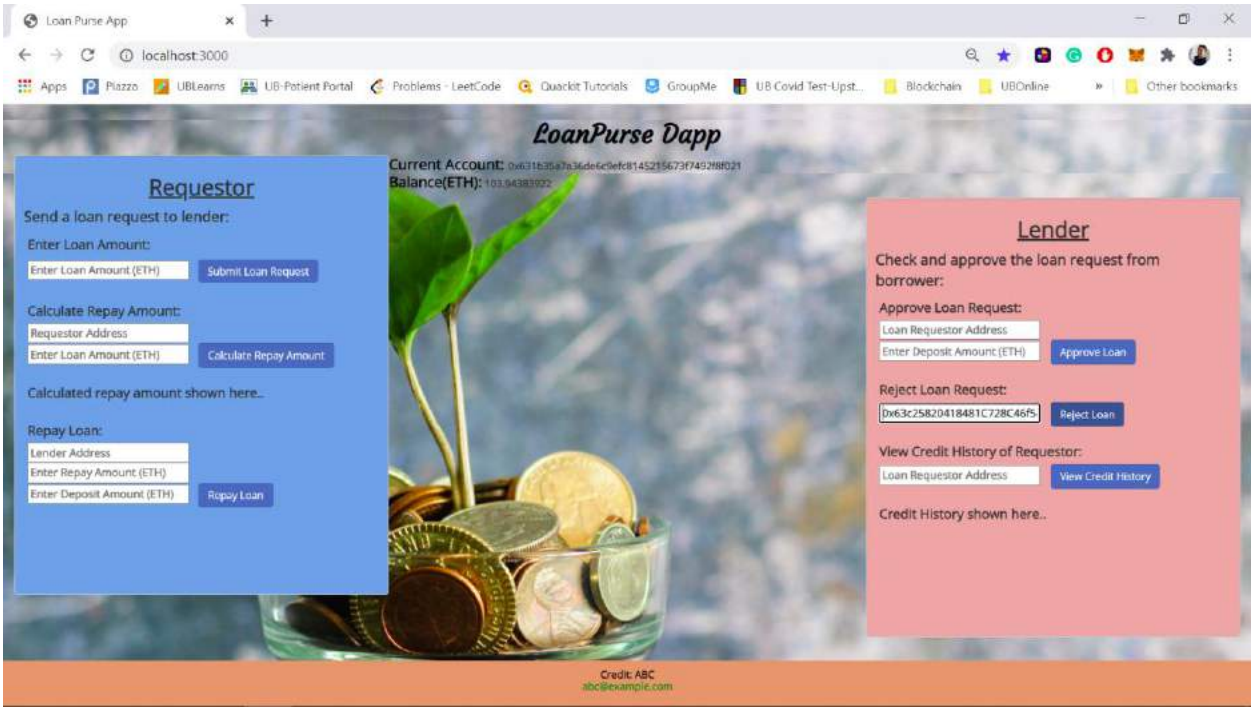
Account 3(Requestor) requests for the loan amount to the lender by entering the required loan amount in the field and submitting the loan request.

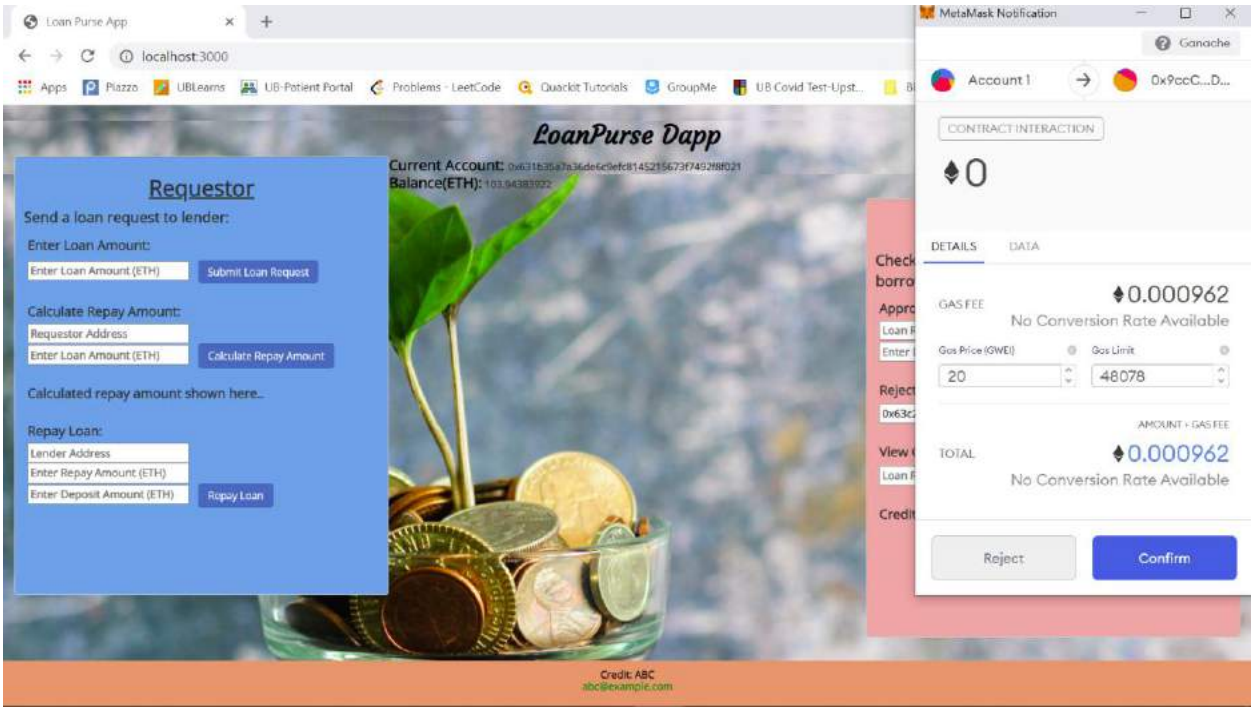




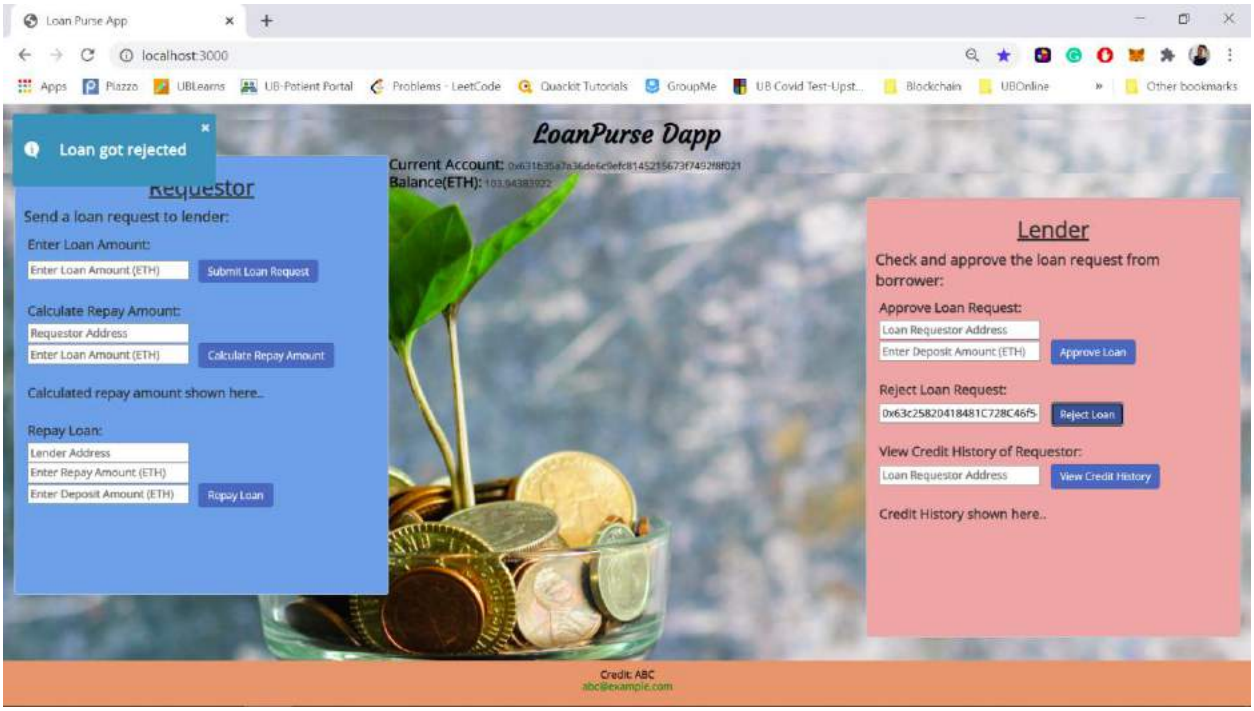
Account 1(Lender) rejects the loan request for the Account 3(Requestor).

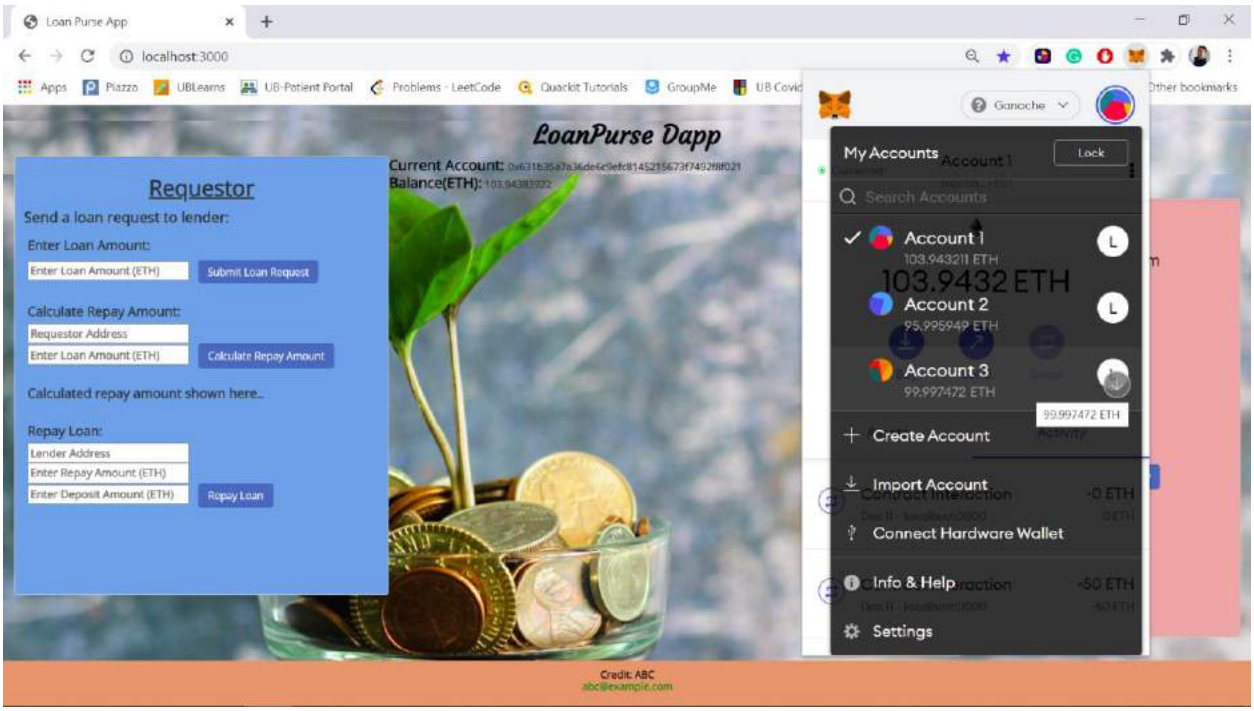


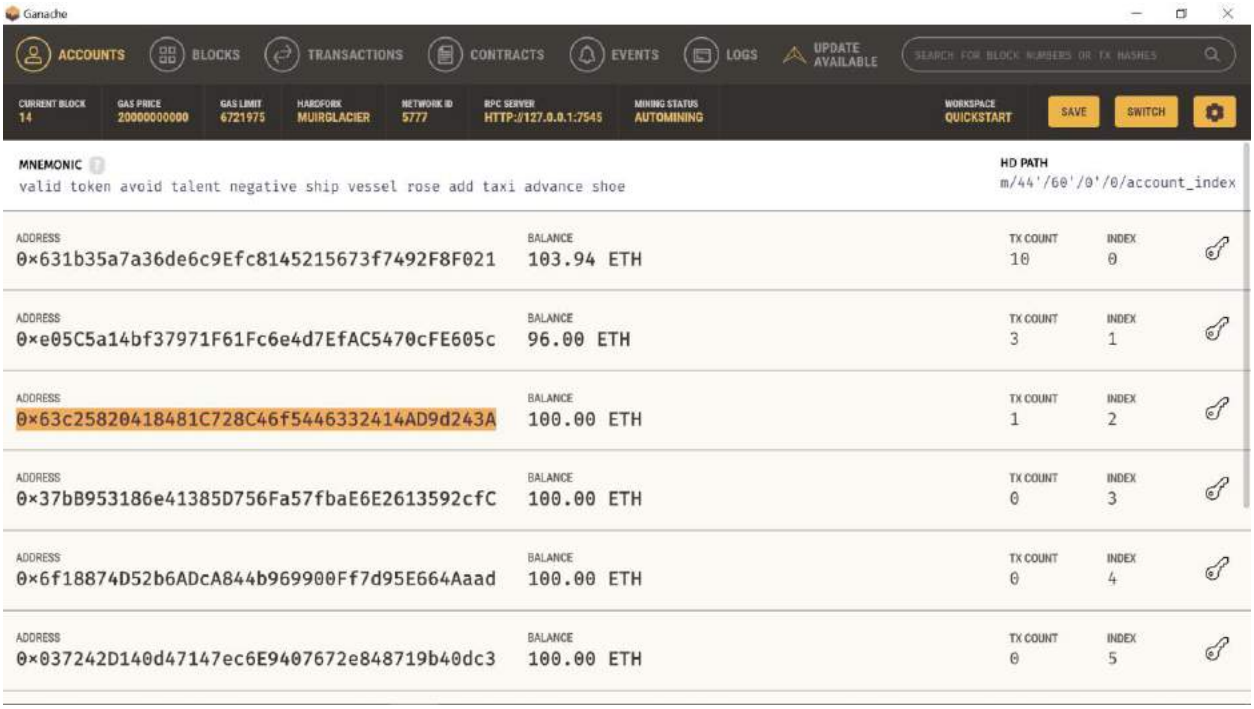




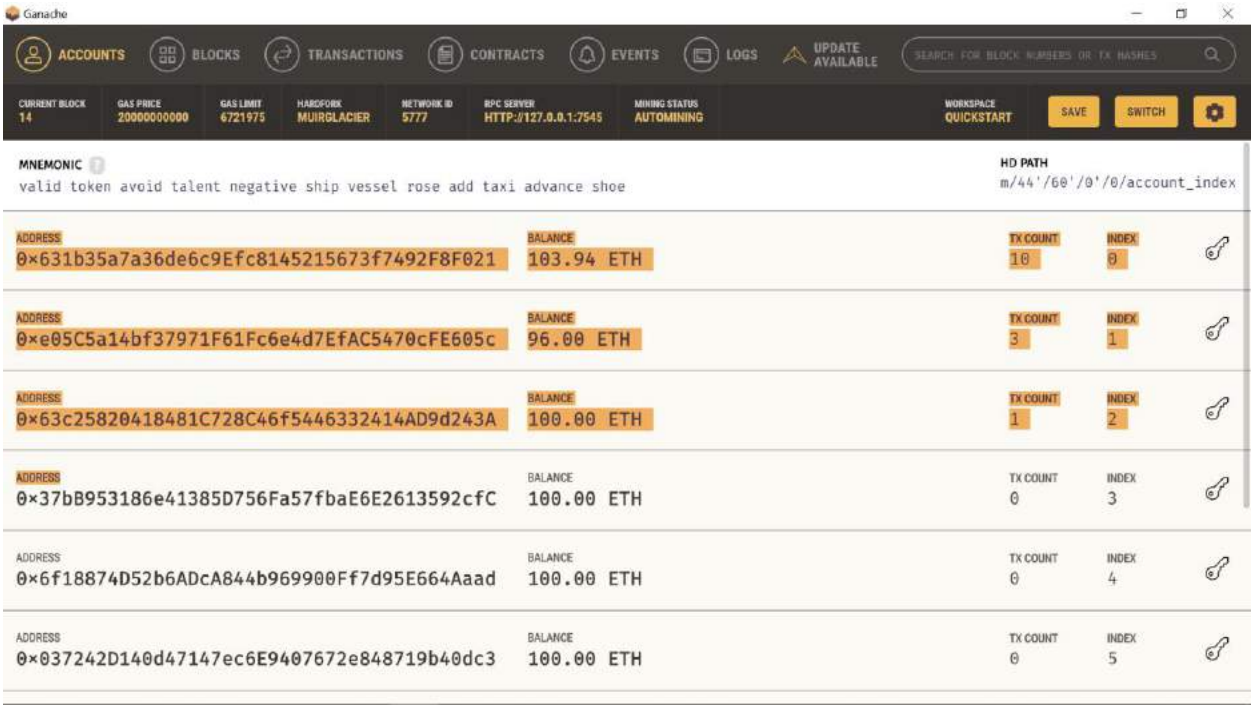
A toast message is displayed once the loan is rejected and no loan amount is transferred to the Requestors' account (Account 3 in this case).

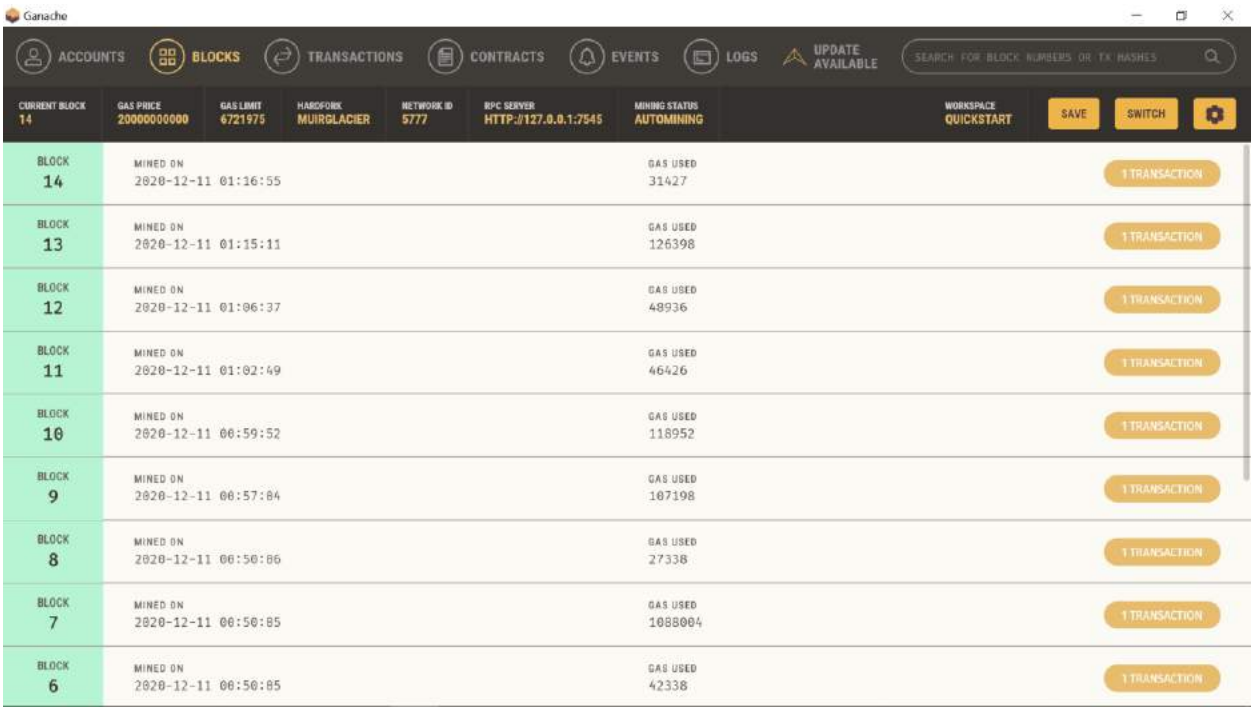


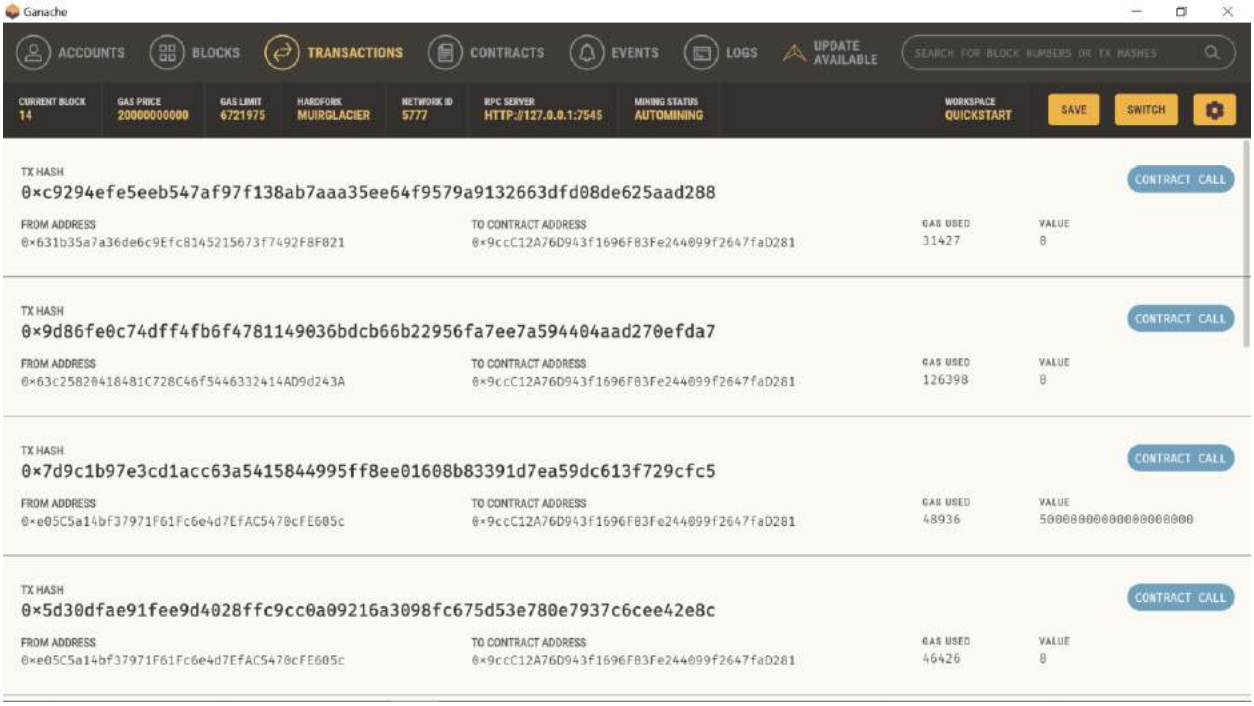

Loan Purse App
localhost:3000
LoanPurse Dapp
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
Requestor Address
Enter Loan Amount (ETH)
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Current Account:
Balance(ETH):
Ganache
My Accounts
Lock
Search Accounts
Account 1
103.943211 ETH
103.9432 ETH
Account 2
95.995949 ETH
Account 3
99.997472 ETH
Create Account
Import Account
Connect Hardware Wallet
Info & Help
Settings
Credit: ABC
abc@example.com

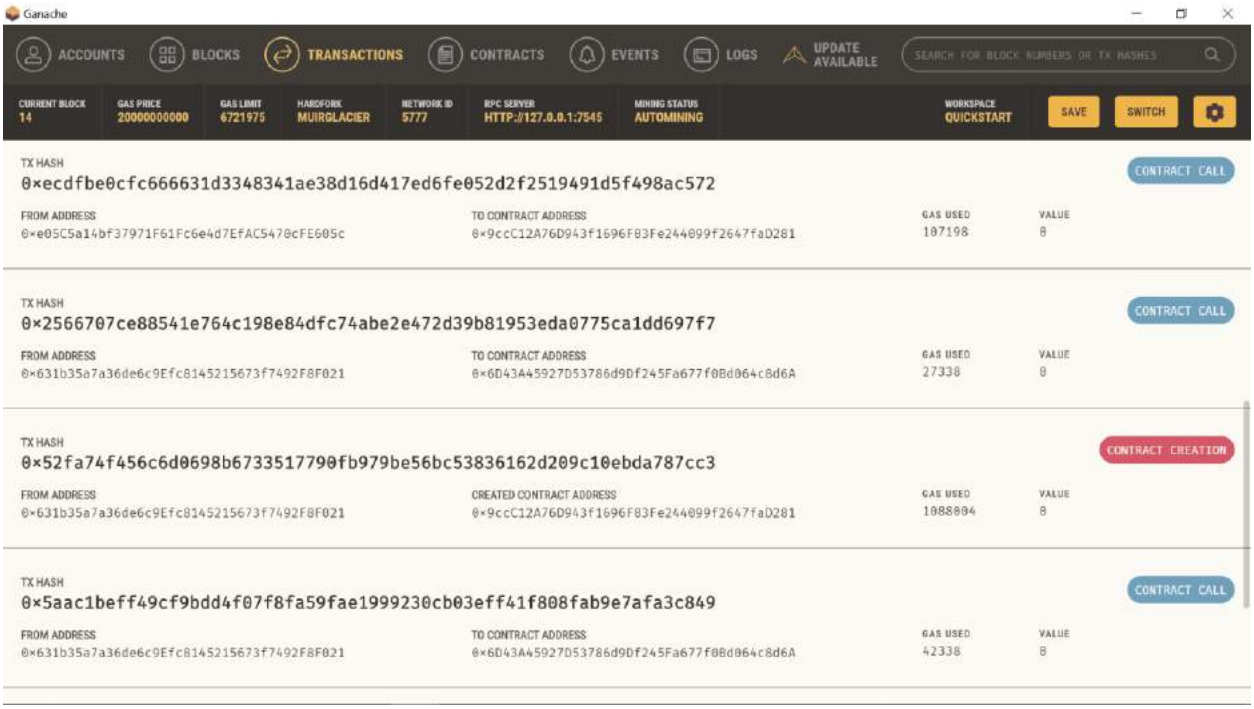

Ganache
ACCOUNTS
BLOCKS
TRANSACTIONS
CONTRACTS
EVENTS
LOGS
UPDATE AVAILABLE
SEARCH FOR BLOCK NUMBERS OR TX HASHES
CURRENT BLOCK 14
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MUIRGLACIER
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:7545
MINING STATUS AUTOMINING
WORKSPACE QUICKSTART
SAVE
SWITCH
MNEMONIC
valid token avoid talent negative ship vessel rose add taxi advance shoe
HD PATH
m/44'/60'/0'/0/account_index
ADDRESS 0x631b35a7a36de6c9Efc8145215673f7492F8F021 BALANCE 103.94 ETH TX COUNT 10 INDEX 0
ADDRESS 0xe05C5a14bf37971F61Fc6e4d7EfAC5470cFE605c BALANCE 96.00 ETH TX COUNT 3 INDEX 1
ADDRESS 0x63c25820418481C728C46f5446332414AD9d243A BALANCE 100.00 ETH TX COUNT 1 INDEX 2
ADDRESS 0x37bB953186e41385D756Fa57fbaE6E2613592cfC BALANCE 100.00 ETH TX COUNT 0 INDEX 3
ADDRESS 0x6f18874D52b6ADcA844b969900Ff7d95E664Aaad BALANCE 100.00 ETH TX COUNT 0 INDEX 4
ADDRESS 0x037242D140d47147ec6E9407672e848719b40dc3 BALANCE 100.00 ETH TX COUNT 0 INDEX 5

Showing the transactions on the Ganache server.






Ganache
ACCOUNTS
BLOCKS
TRANSACTIONS
CONTRACTS
EVENTS
LOGS
UPDATE AVAILABLE
SEARCH FOR BLOCK NUMBERS OR TX HASHES
CURRENT BLOCK 14
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MUIRGLACIER
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:7545
MINING STATUS AUTOMINING
WORKSPACE QUICKSTART
SAVE
SWITCH
TX HASH
0×c9294efe5eeb547af97f138ab7aaa35ee64f9579a9132663dfd08de625aad288
CONTRACT CALL
FROM ADDRESS
0×631b35a7a36de6c9Efc8145215673f7492F8F021
TO CONTRACT ADDRESS
0×9ccC12A76D943f1696F83Fe244099f2647faD281
GAS USED
31427
VALUE
0
TX HASH
0×9d86fe0c74dff4fb6f4781149036bdcb66b22956fa7ee7a594404aad270efda7
CONTRACT CALL
FROM ADDRESS
0×63c25820418481C728C46f5446332414AD9d243A
TO CONTRACT ADDRESS
0×9ccC12A76D943f1696F83Fe244099f2647faD281
GAS USED
126398
VALUE
0
TX HASH
0×7d9c1b97e3cd1acc63a5415844995ff8ee01608b83391d7ea59dc613f729cfc5
CONTRACT CALL
FROM ADDRESS
0×e05C5a14bf37971F61Fc6e4d7EfAC5470cFE605c
TO CONTRACT ADDRESS
0×9ccC12A76D943f1696F83Fe244099f2647faD281
GAS USED
48936
VALUE
5000000000000000000
TX HASH
0×5d30dfae91fee9d4028ffc9cc0a09216a3098fc675d53e780e7937c6cee42e8c
CONTRACT CALL
FROM ADDRESS
0×e05C5a14bf37971F61Fc6e4d7EfAC5470cFE605c
TO CONTRACT ADDRESS
0×9ccC12A76D943f1696F83Fe244099f2647faD281
GAS USED
46426
VALUE
0

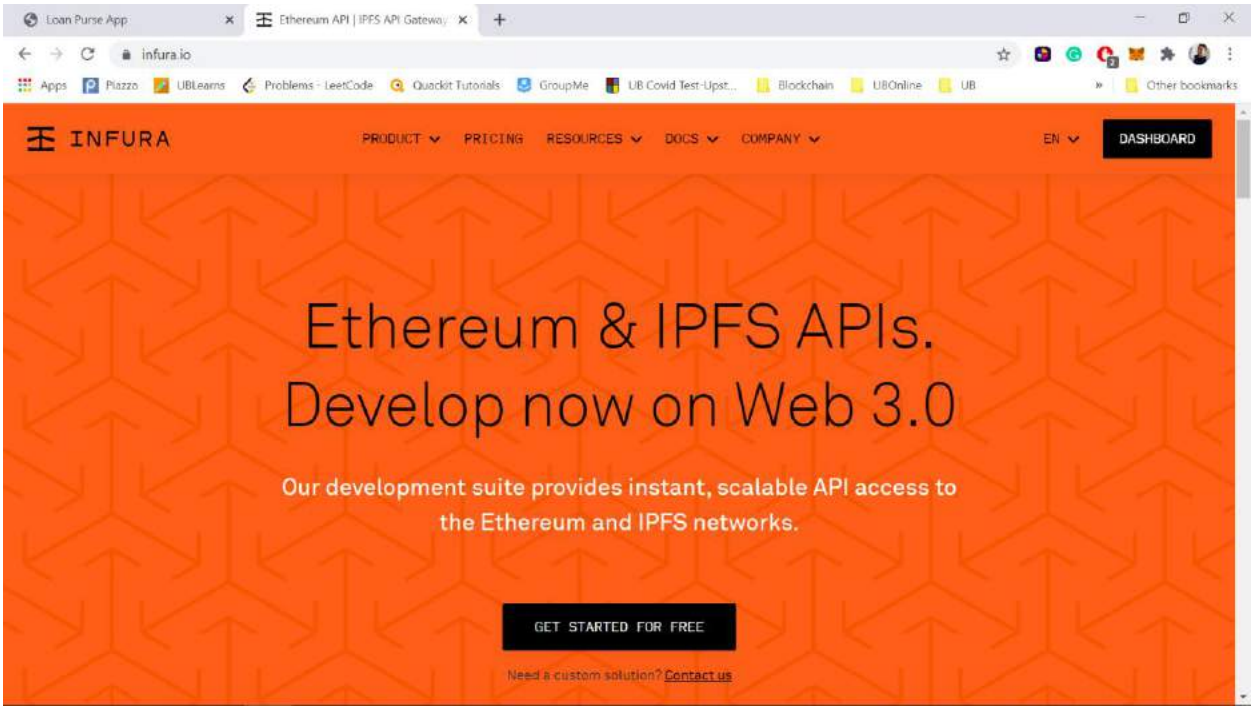

Ganache
ACCOUNTS
BLOCKS
TRANSACTIONS
CONTRACTS
EVENTS
LOGS
UPDATE AVAILABLE
SEARCH FOR BLOCK NUMBERS OR TX HASHES
CURRENT BLOCK 14
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MUIRGLACIER
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:7545
MINING STATUS AUTOMINING
WORKSPACE QUICKSTART
SAVE
SWITCH
TX HASH
0×ecdfbe0cfc666631d3348341ae38d16d417ed6fe052d2f2519491d5f498ac572
CONTRACT CALL
FROM ADDRESS
0×e05C5a14bf37971F61Fc6e4d7EfAC5470cFE605c
TO CONTRACT ADDRESS
0×9ccC12A76D943f1696F83Fe244099f2647faD281
GAS USED
107198
VALUE
0
TX HASH
0×2566707ce88541e764c198e84dfc74abe2e472d39b81953eda0775ca1dd697f7
CONTRACT CALL
FROM ADDRESS
0×631b35a7a36de6c9Efc8145215673f7492F8F021
TO CONTRACT ADDRESS
0×6D43A45927D53786d9Df245Fa677f0Bd064c8d6A
GAS USED
27338
VALUE
0
TX HASH
0×52fa74f456c6d0698b6733517790fb979be56bc53836162d209c10ebda787cc3
CONTRACT CREATION
FROM ADDRESS
0×631b35a7a36de6c9Efc8145215673f7492F8F021
CREATED CONTRACT ADDRESS
0×9ccC12A76D943f1696F83Fe244099f2647faD281
GAS USED
1088804
VALUE
0
TX HASH
0×5aac1beff49cf9bdd4f07f8fa59fae1999230cb03eff41f808fab9e7afa3c849
CONTRACT CALL
FROM ADDRESS
0×631b35a7a36de6c9Efc8145215673f7492F8F021
TO CONTRACT ADDRESS
0×6D43A45927D53786d9Df245Fa677f0Bd064c8d6A
GAS USED
42338
VALUE
0

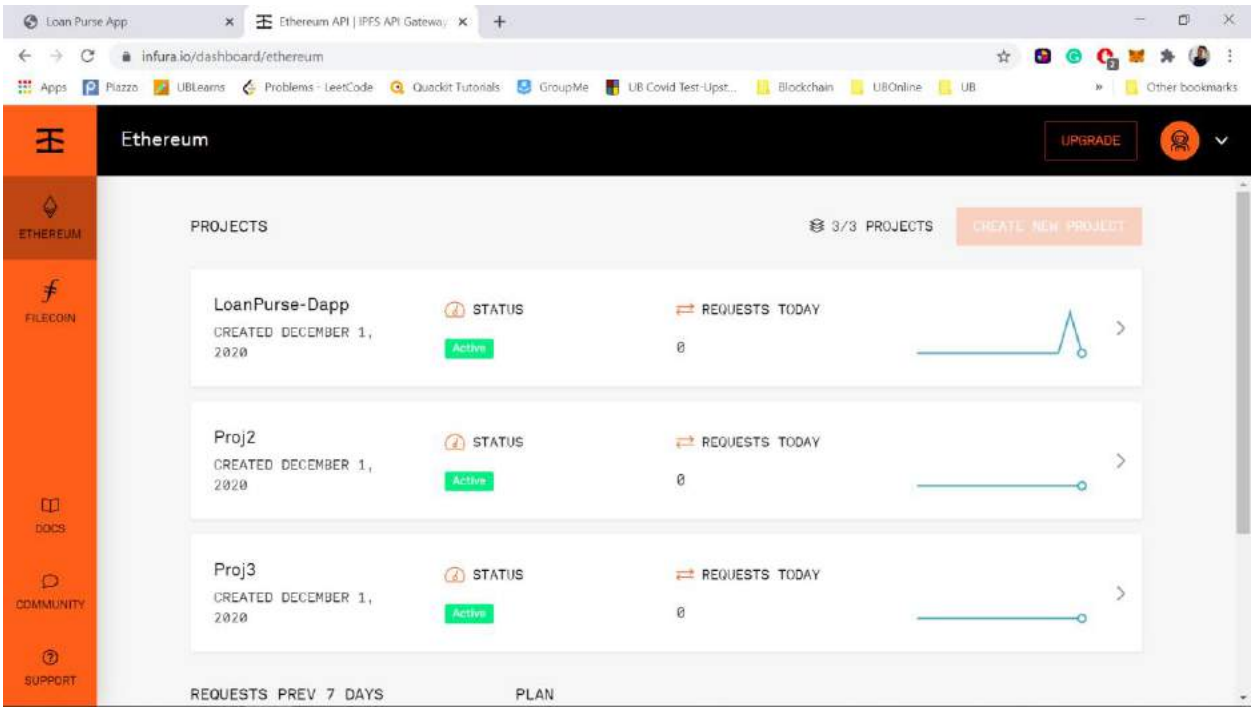
Phase 6
Dapp Deployment on Infura

This phase includes the deployment of the designed Dapp on Infura cloud network for the public use where any decentralized users can interact with the Dapp.

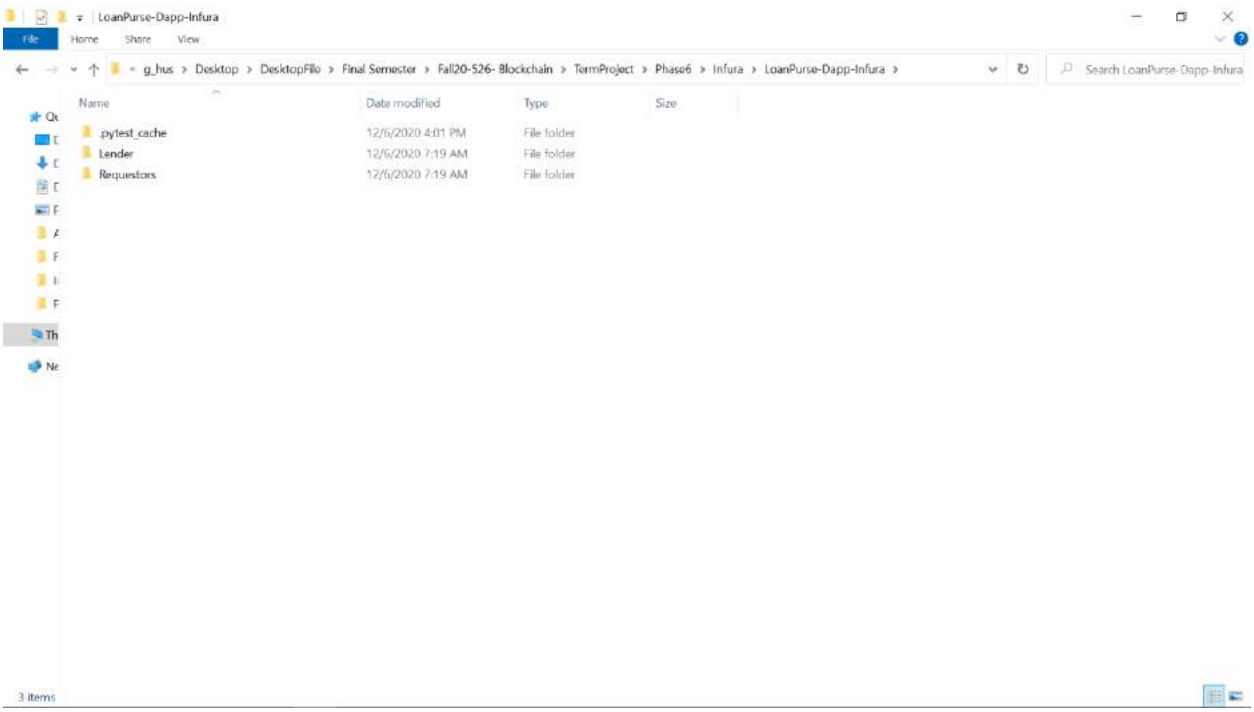
I have created a project called (LoanPurse -Dapp) on the Infura to host my Dapp as shown below.

Infura details:

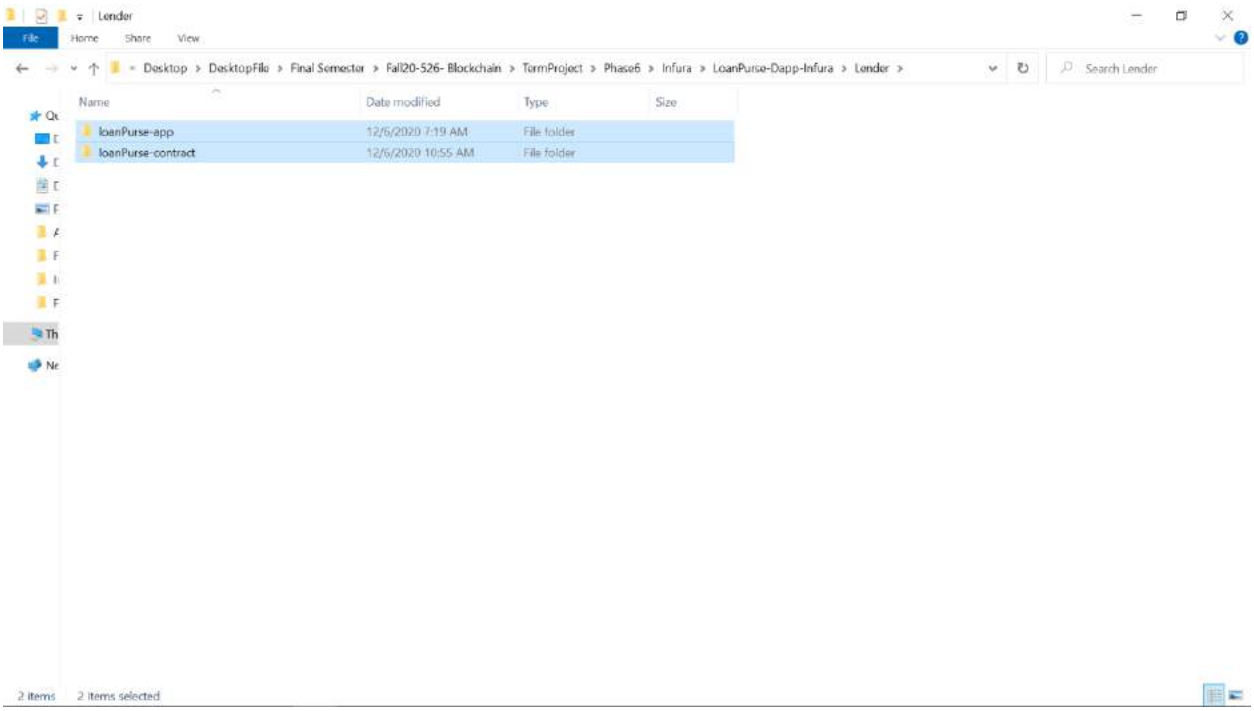




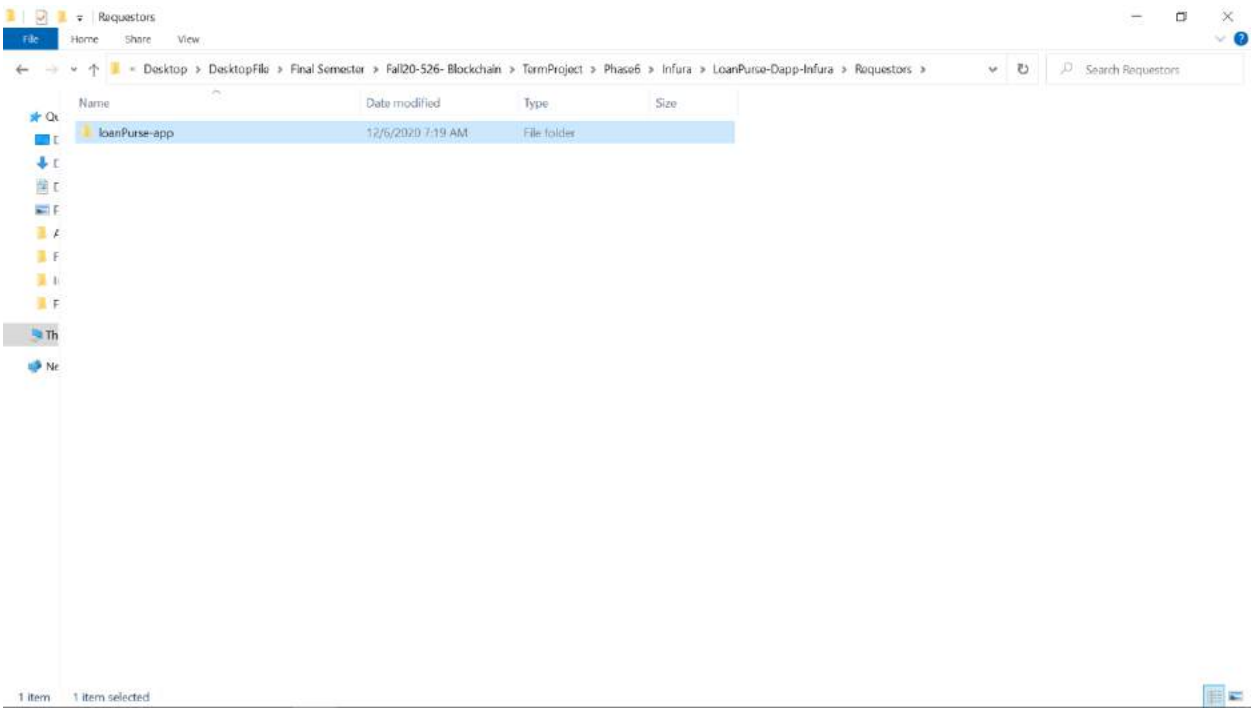
Dapp folder structure:



Lender has both contract and app files as shown below



Requestor (Decentralized end-users) has only app file with smart contract address which can be accessed.

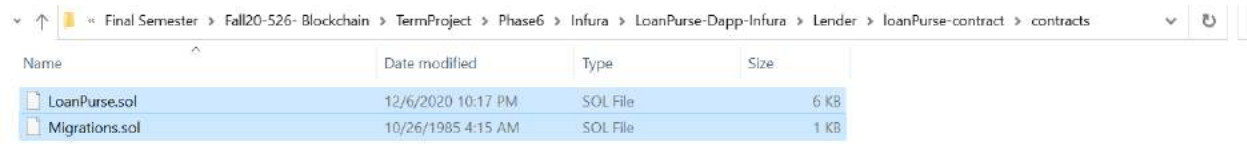


Truffle-config.js file:

```
const HDWalletProvider = require('truffle-hdwallet-provider');
lenders'';

module.exports = {
  // See <http://truffleframework.com/docs/advanced/configuration>
  // to customize your Truffle configuration!
  networks: {
    ropsten: {
      provider: () => new HDWalletProvider(
      network_id: 3,
      gas: 6000000,
      skipDryRun: false
    },
    development: {
      host: "localhost",
      port: 7545,
      network_id: "*" //"5777",
    }
  },

  compilers: {
    solc: {
      version: "0.5.8"
    }
  }
}
```

Package.json

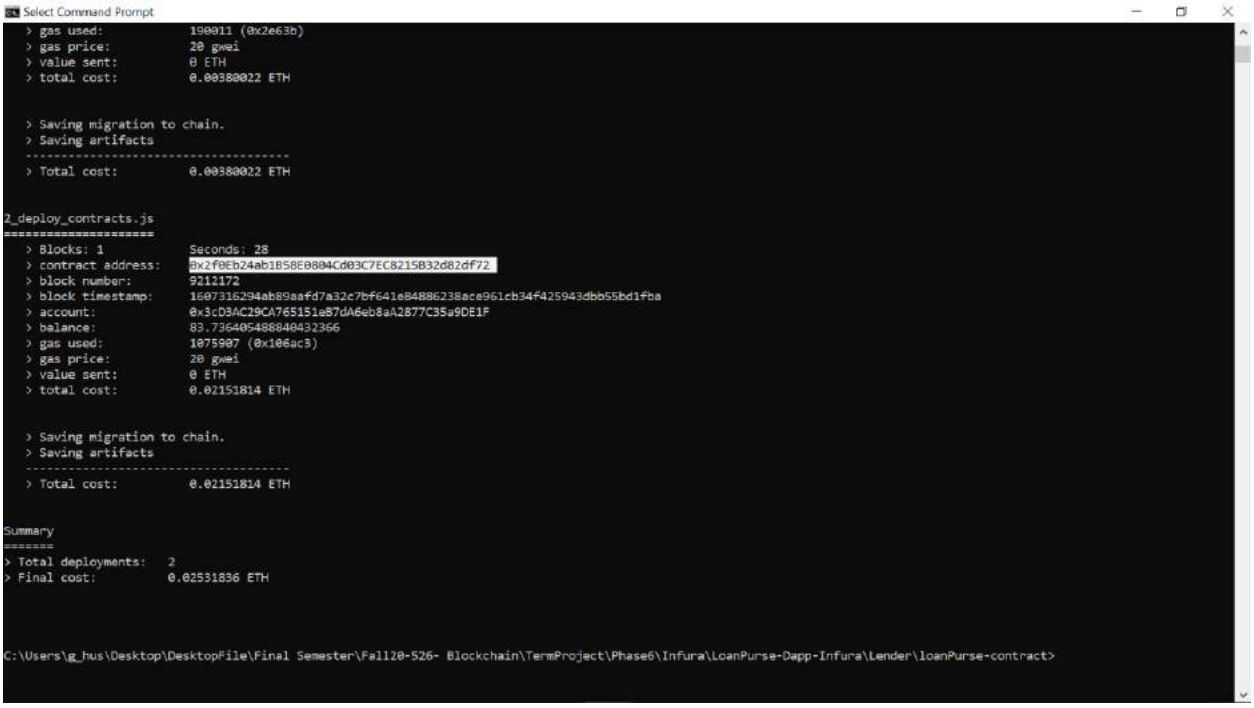
```
{
  "name": "loanPurse-app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "devDependencies": {
    "truffle-hdwallet-provider": "^1.0.17"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.4"
  },
  "description": ""
}
```



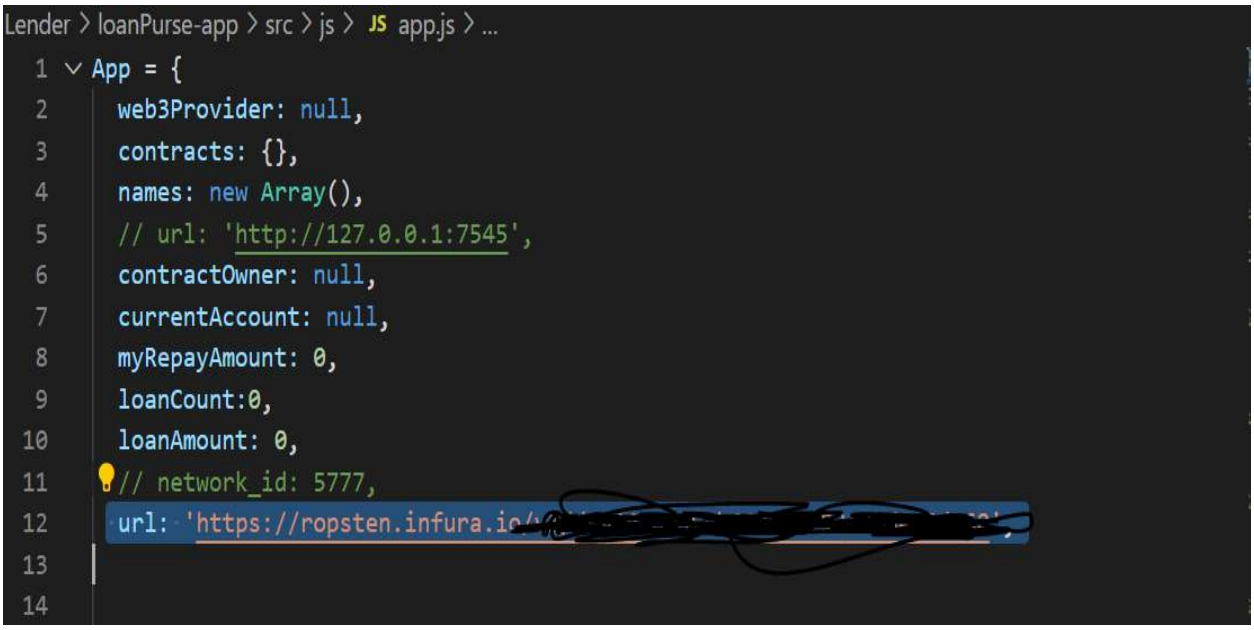
Deployment of smart contract using truffle with Ropsten account:

```
truffle migrate --network ropsten
```

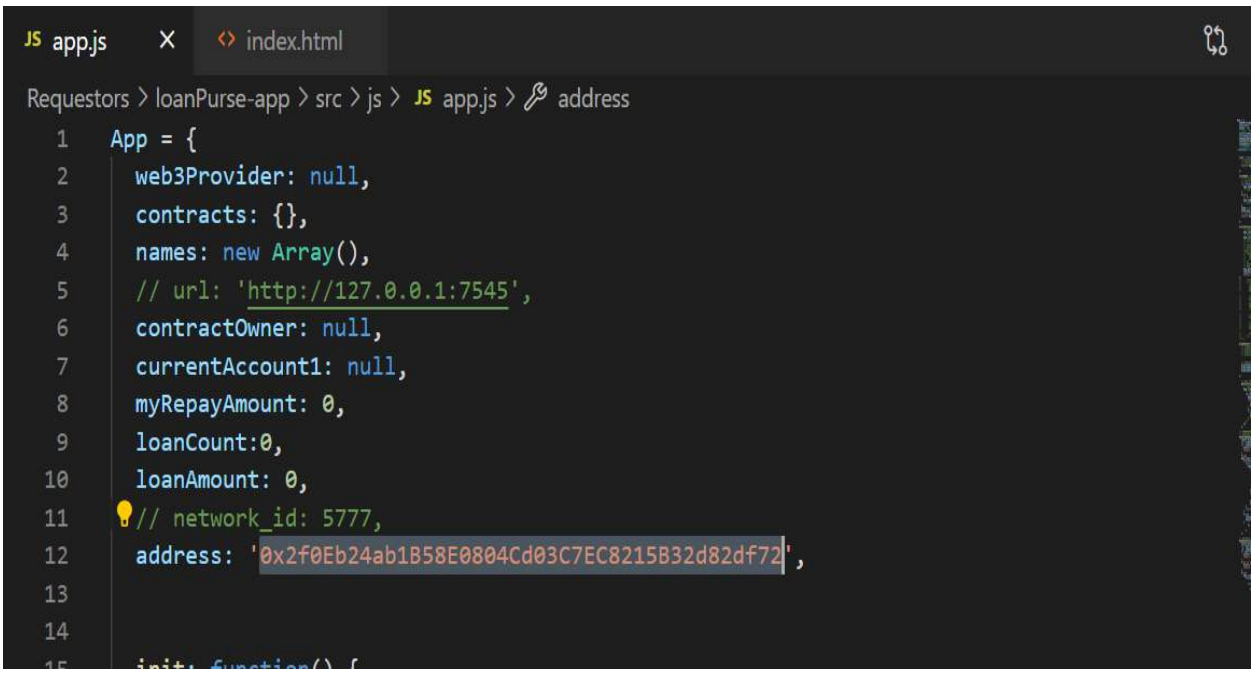
Smart Contract Address:



Infura URL where Lender has deployed the Dapp:

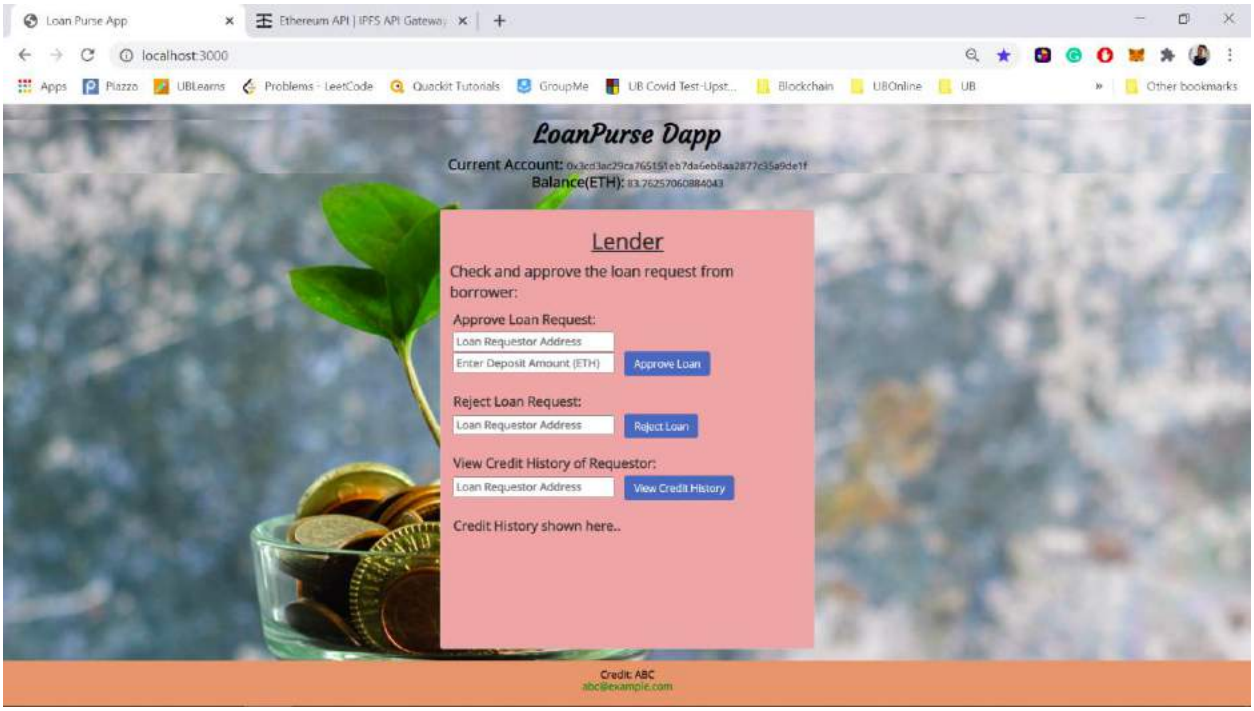


Smart contract address is available for Decentralized users(Requestors) to use the Dapp:

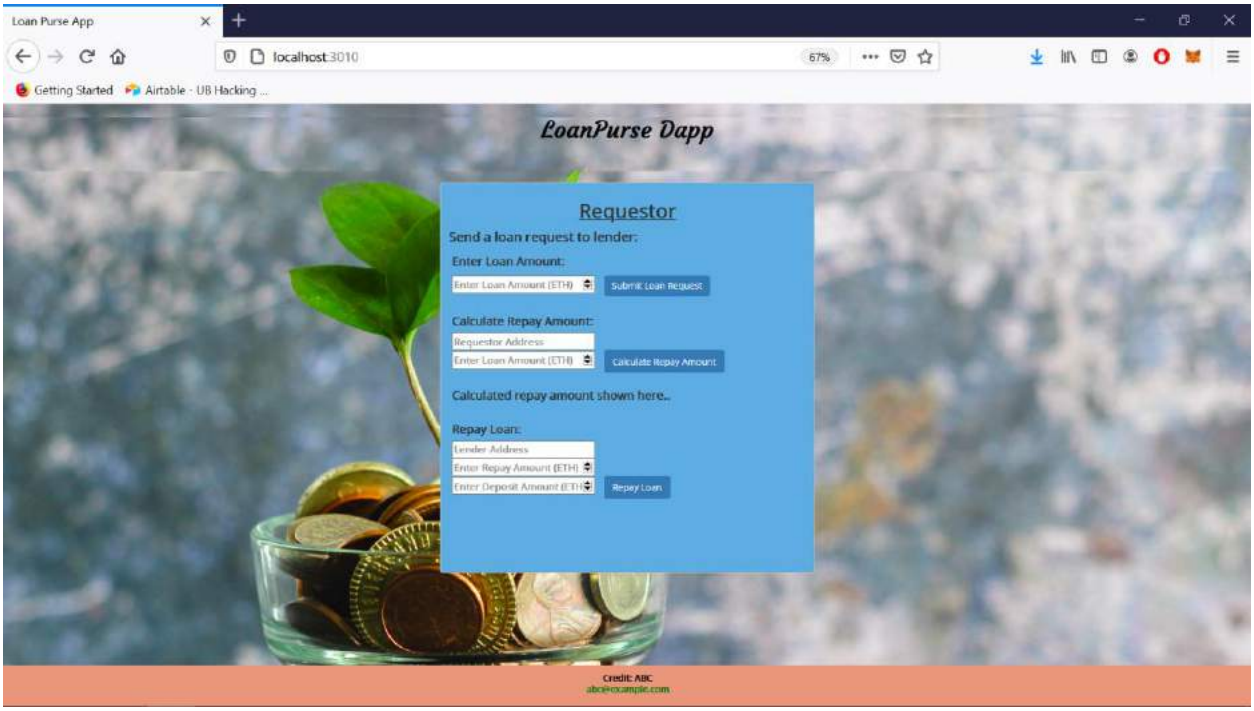


Dapp web part screenshots:

Lenders' screen:



Requestors' screen (Decentralized end-users):



- **Steps to execute application:**
  - Server side need to install the node modules and compile and deploy the contract using below commands.

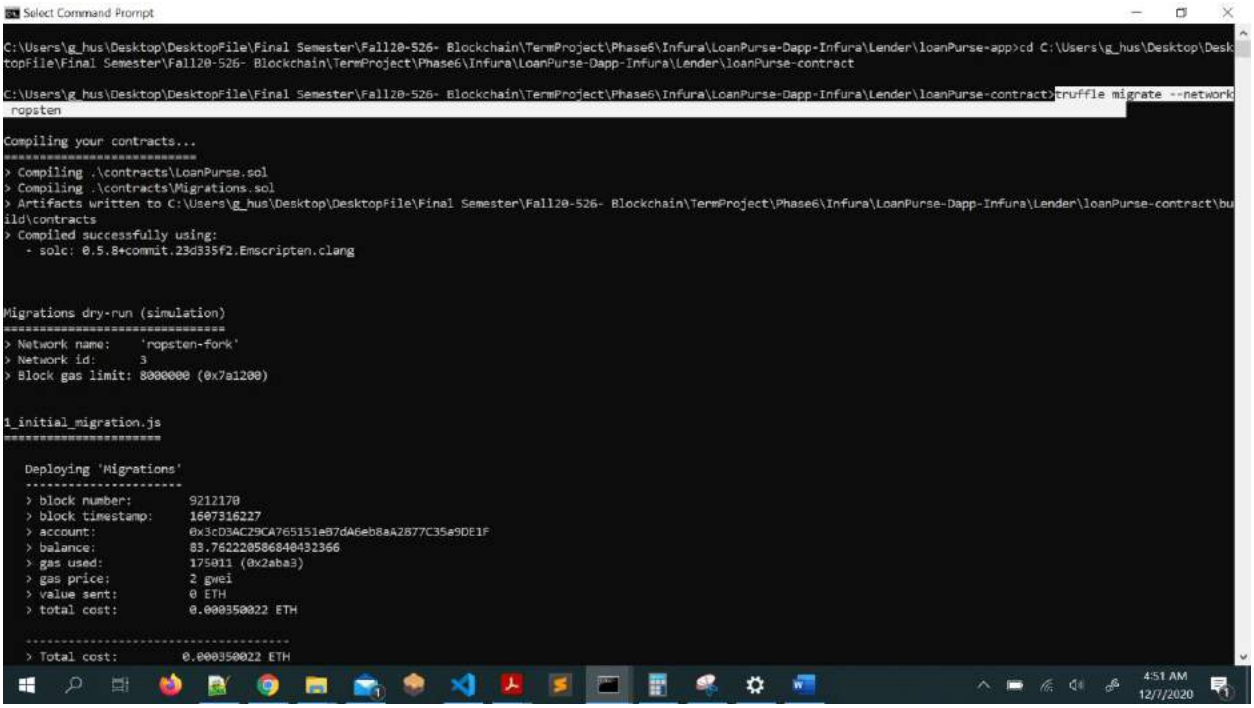
```
npm install
truffle migrate --network Ropsten
```

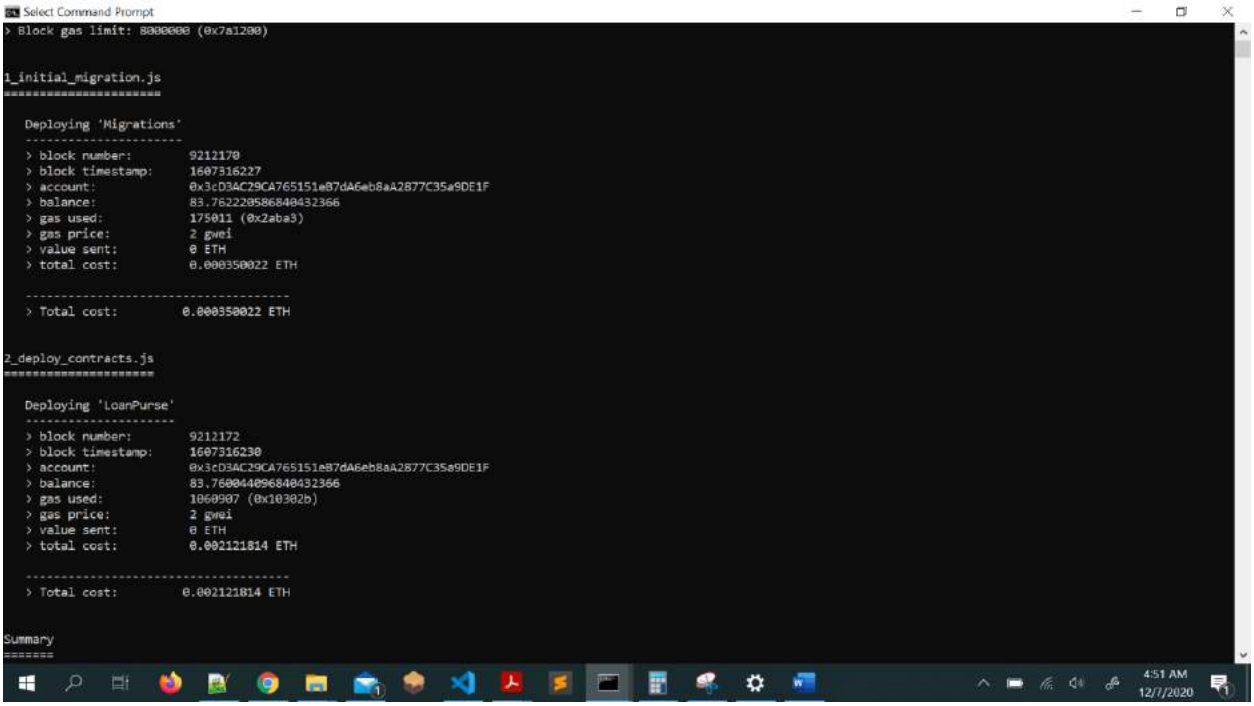
  - The application will start listening on port 3000 for server after executing the following command and available for use by the Lender(Smart contract owner) and process any of the requests coming from the Requestors' side

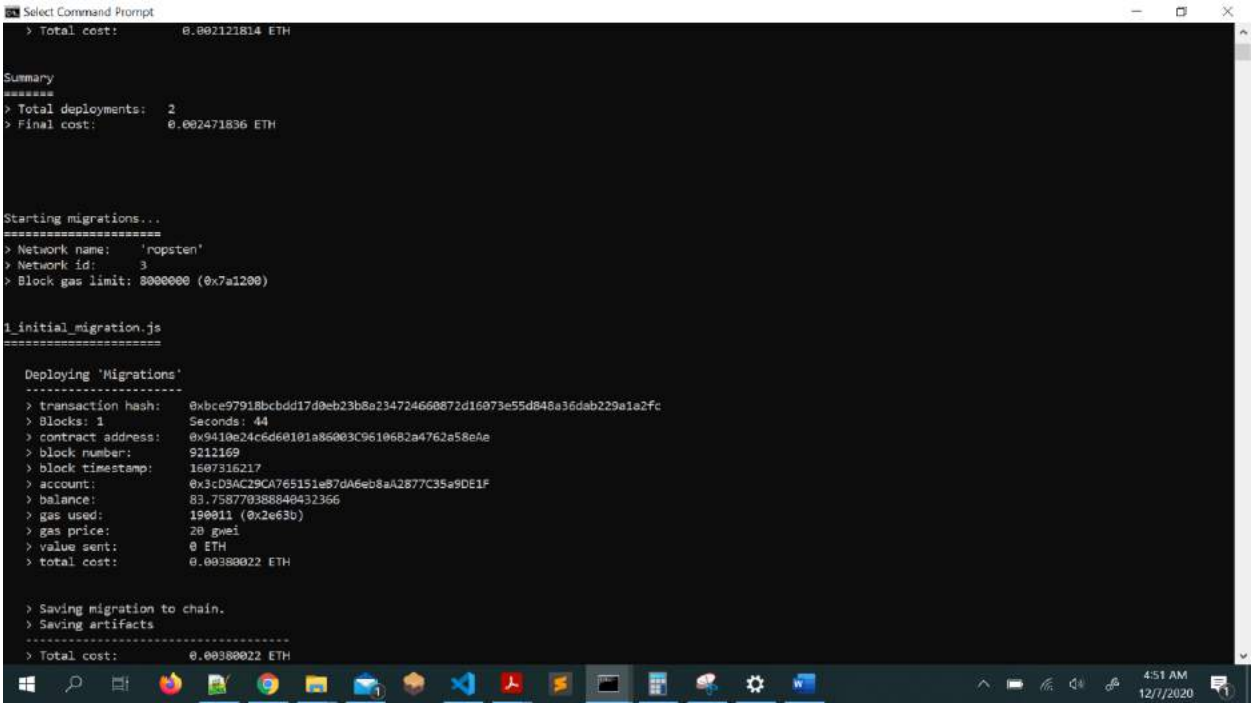
```
npm install
npm start
```

  - Client side need to install the node modules. The application will start listening at port 3010 after executing the following commands. The Requestors can use the web application to start the loan request process and send the details to Lender who is responsible for approving the loan request and transfer the loan amount to requestor.

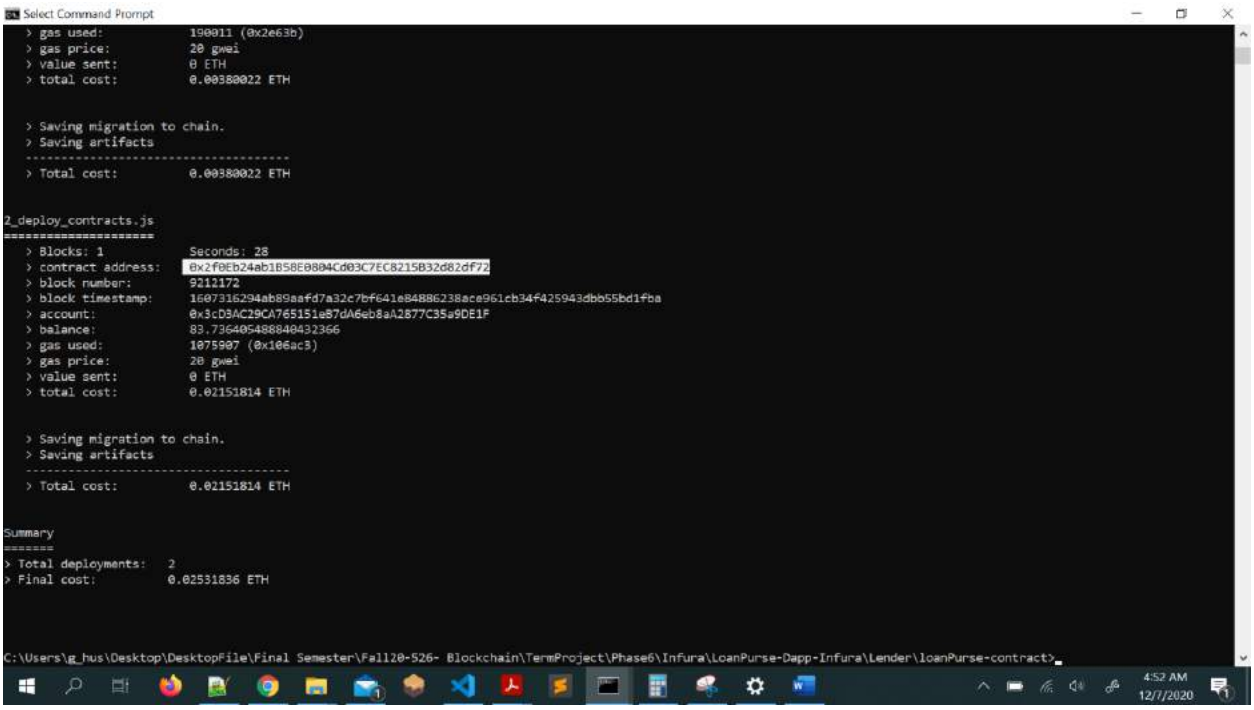
```
npm install
npm start
```




Select Command Prompt
> Block gas limit: 8000000 (0x7a1200)
1_initial_migration.js
Deploying 'Migrations'
> block number: 9212170
> block timestamp: 1607316227
> account: 0x3cD3AC29CA765151e87dA6eb8aA2877C35a9DE1F
> balance: 83.76222058684043236
> gas used: 175011 (0x2aba3)
> gas price: 2 gwei
> value sent: 0 ETH
> total cost: 0.000350022 ETH
> Total cost: 0.000350022 ETH
2_deploy_contracts.js
Deploying 'LoanPurse'
> block number: 9212172
> block timestamp: 1607316230
> account: 0x3cD3AC29CA765151e87dA6eb8aA2877C35a9DE1F
> balance: 83.760044096840432366
> gas used: 1060907 (0x1030b)
> gas price: 2 gwei
> value sent: 0 ETH
> total cost: 0.002121814 ETH
> Total cost: 0.002121814 ETH
Summary
4:51 AM
12/7/2020


Select Command Prompt
> Total cost: 0.002121814 ETH
Summary
> Total deployments: 2
> Final cost: 0.002471836 ETH
Starting migrations...
> Network name: 'ropsten'
> Network id: 3
> Block gas limit: 8000000 (0x7a1200)
1_initial_migration.js
Deploying 'Migrations'
> transaction hash: 0xbce97918bcbdd17d0eb23b8a23472460872d16073e55d848a36dab229a1a2fc
> Blocks: 1 Seconds: 44
> contract address: 0x9410e24c6d60101a86003C9610682a4762a58eAe
> block number: 9212169
> block timestamp: 1607316217
> account: 0x3cD3AC29CA765151e87dA6eb8aA2877C35a9DE1F
> balance: 83.75877038884043236
> gas used: 190011 (0x2e63b)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.00380022 ETH
> Saving migration to chain.
> Saving artifacts
> Total cost: 0.00380022 ETH
4:51 AM
12/7/2020

Smart Contract Address:



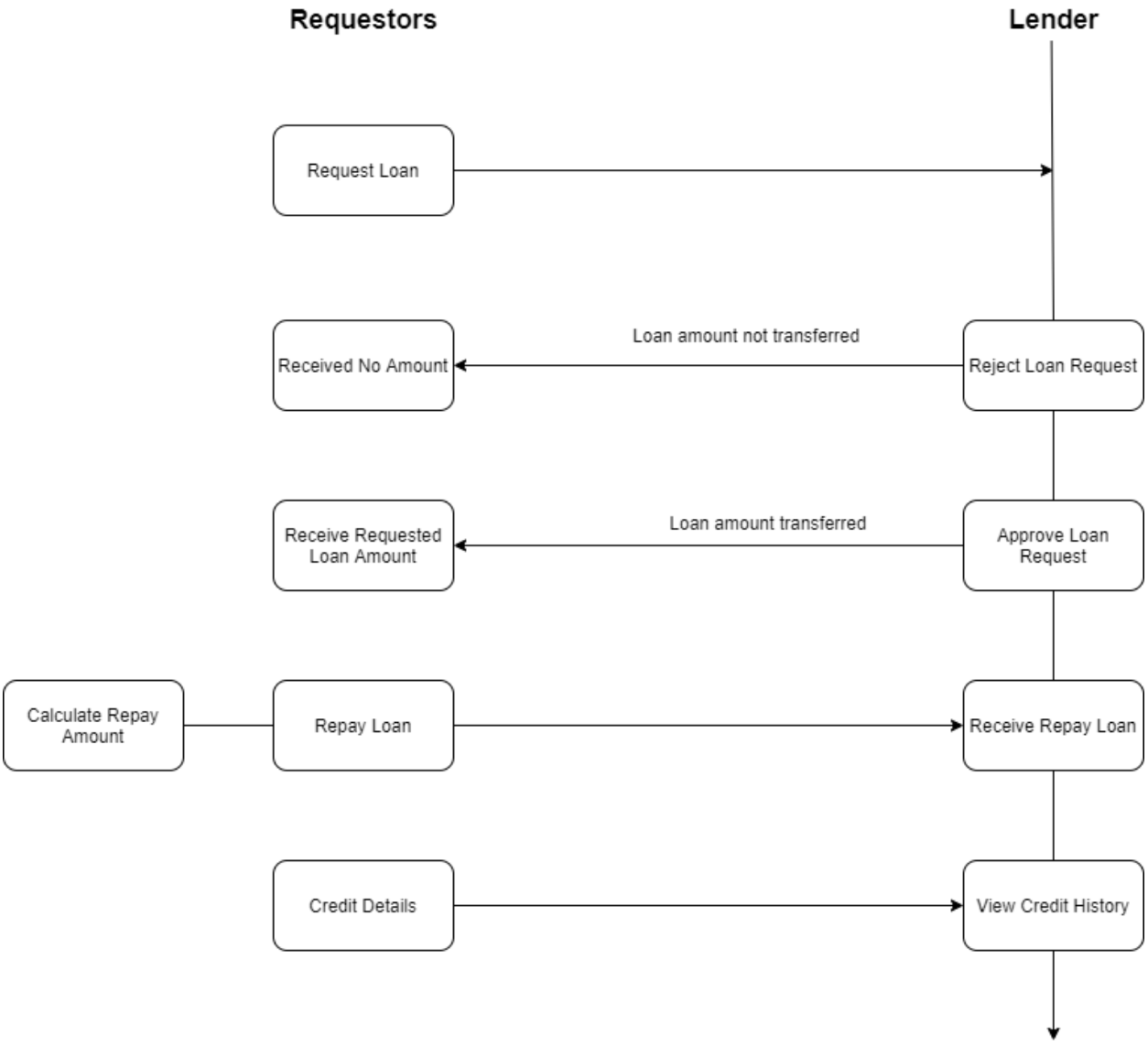
Phase 6 - Conclusion:

The Loan Purse Dapp has been successfully deployed on Infura cloud-like platform and available for the decentralized end-users to start using the application and interact with it for further requests.

Phase 7
Bundle up and Project wrap up

This is the final phase for the project which includes the updated project documentation including the design phase till deployment phase.

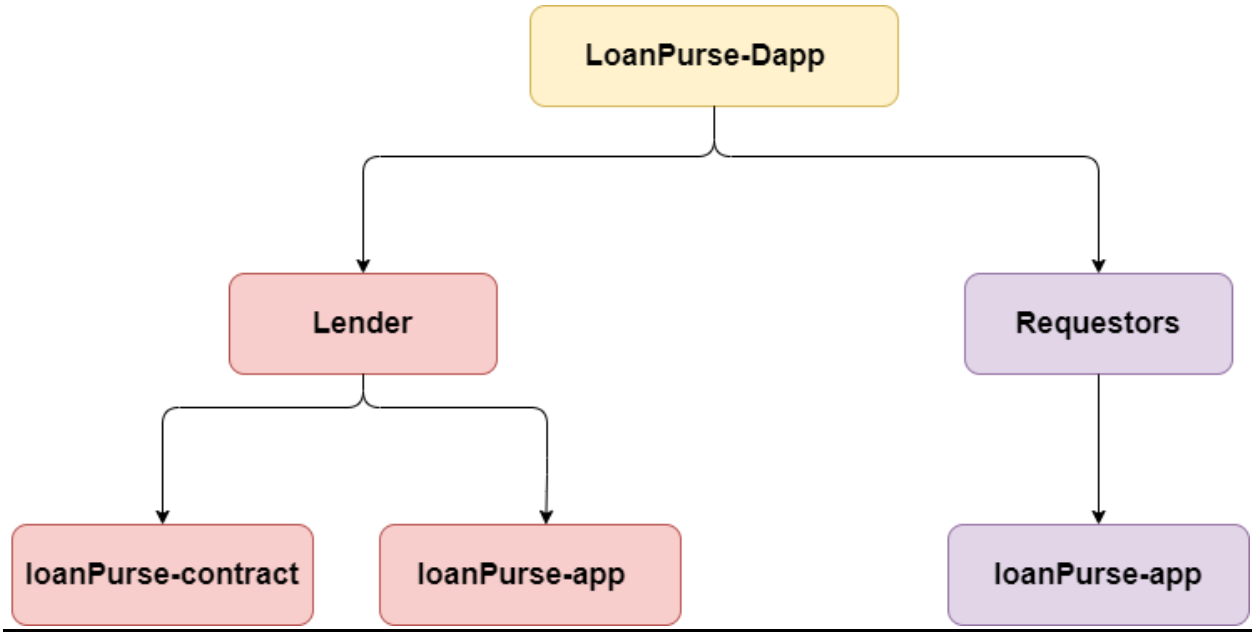
Interaction Diagram:



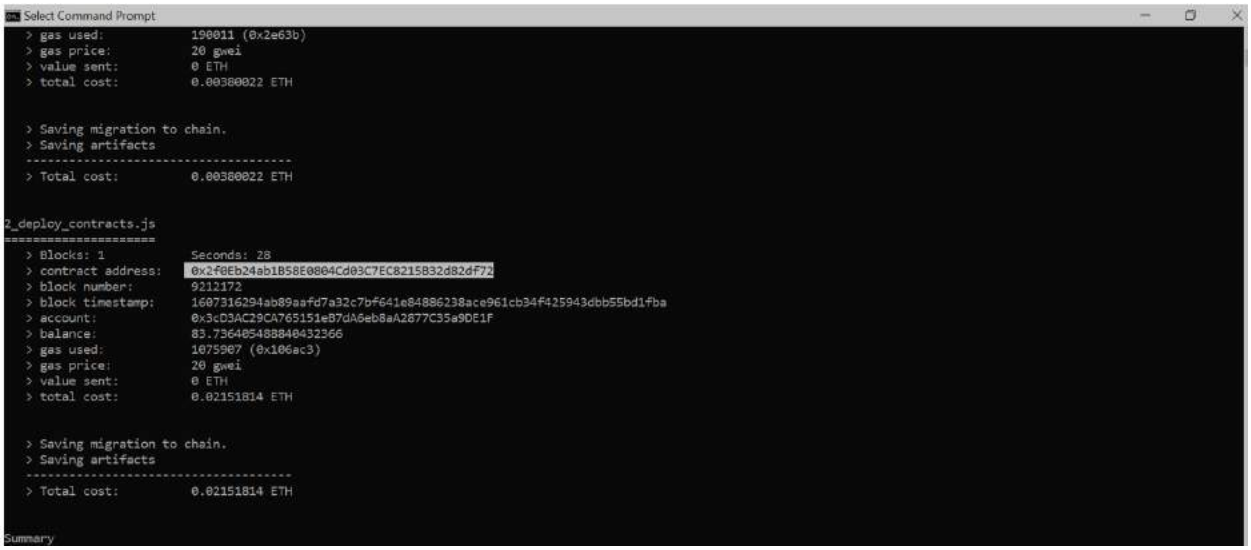
The above flow diagram shows the working flow of the LoanPurse Dapp including all the processes being followed by the stakeholders of the Dapp like Requestors and Lender. Requestor requests for the loan amount to the Lender. Lender then either rejects the loan request or

approves the loan request. In case, Lender rejects the loan request from the Requestor, the loan amount is not transferred to the Requestors' account. If the loan request is approved by the Lender, the requested loan amount is transferred to the Requestors' account. While repaying the loan, the Requestor calculates the Repay Loan Amount and repays the loan to the Lender. Once, the Requestor repays the loan, the repaid amount gets transferred to the Lenders' account. Lender can also view the Credit History of the Requestors.

Directory Structure:



Smart Contract Address:



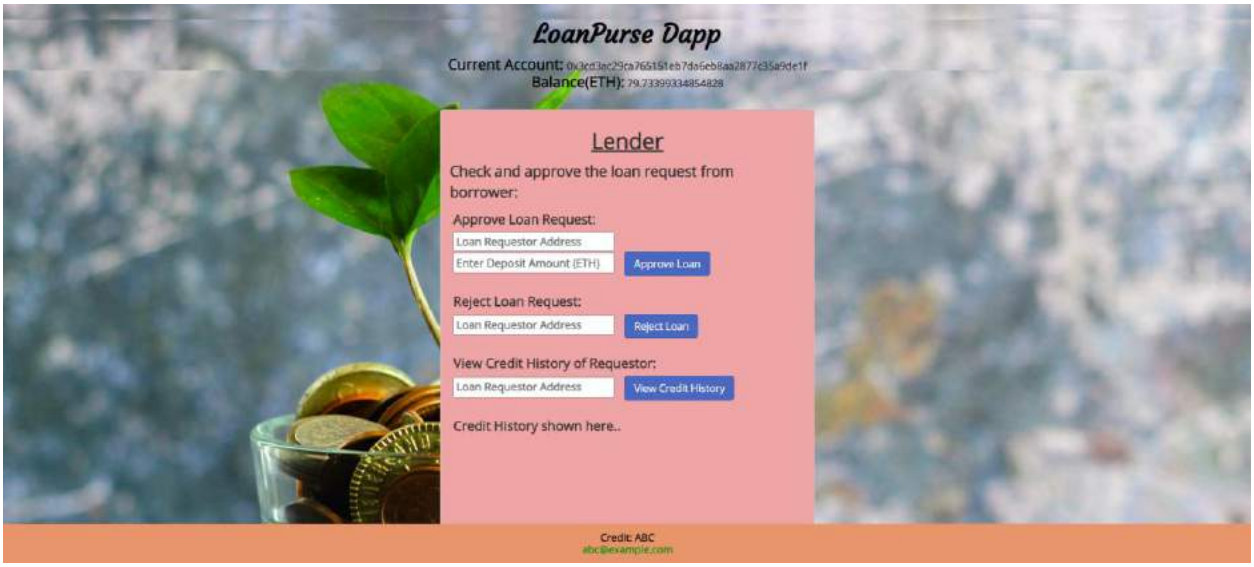
Smart Contract Code Snippets:

```
pragma solidity >=0.4.22 <=0.6.0;

contract  LoanPurse {
        address payable public lender;  // owner of the contract who deploy the contract
        uint public calRepayAmount; // calculate the Repay Amount based on predefined interest rate
        uint noOfLoan=0;  // count of total loans
        uint amt;
        mapping(address => Loan) loans; // mapping address to loan
        mapping(address => uint) public amount;

         // loan structure
        struct Loan {
           uint id;
           address borrower;
           uint loanAmount;
           string loanStatus;
       }

        // Events are defined here
        event Initialized();
        event LoanRequested();
        event LoanApproved();
        event LoanRejected();
        event LoanRepaid();
        event myCalc(uint myRepay);
        event loanCreditHistory(uint loanCount1, uint loanAmt1);


        // modifier to check the functions which are applicable to only Lender can be done
        // by Lender (e.g. approveLoan, rejectLoan)
        modifier onlyLender()
        { require(msg.sender == lender);
            _;
        }

         // modifier to check the functions which are not applicable to Lender can not be done
         // by Lender (e.g. requestLoan, repayLoan)
         modifier notLender()
         { require(msg.sender != lender);
             _;
         }

         // constructor() is executed when the smart contract is deployed into the network
         // and `msg.sender` which is the Lender/owner, is the account creating this contract.
         constructor () public {
             lender = msg.sender;

             emit Initialized();
         }

         // function to request Loan by borrower to Lender by giving few details about the borrower and reason for loan
         function requestLoan(uint loanAmount) notLender public {
             // add Loan object to mapping
             amount[msg.sender] = loanAmount * 10000000000000000000;
             loans[msg.sender] = Loan(noOfLoan, msg.sender, loanAmount, "Loan Requested");

             emit LoanRequested();
         }

         // function to approve the Loan request by Lender and transfer the Loan amount to borrower account
         function approveLoan(address payable requestorAddress) onlyLender payable public {
             require(msg.sender == lender);
             require(msg.value > amt, "Insufficient Balance.");
             amt = amount[requestorAddress];
             requestorAddress.transfer(amt);

             // Transfer the remaining amount to Lender's account after transferring the requested Loan amount from msg.value amount
             uint tempBalance = 0;
               tempBalance = msg.value - amt;

               // transfer the requested Loan Amount to Requestor's account
               msg.sender.transfer(tempBalance);

               noOfLoan++;
               loans[requestorAddress] = Loan(noOfLoan, requestorAddress, amt, "Loan Request Approved");
               emit LoanApproved();
           }

          // function to reject the Loan request by Lender
          function rejectLoan(address requestorAddress) onlyLender public {
              require(msg.sender == lender);
              loans[requestorAddress].loanStatus="Loan Request Rejected";

              emit LoanRejected();
          }

          // function to view the Credit History of the Requestor
           function creditHistory(address requestorAddress) public returns(uint , uint) {
              Loan memory loanDetails = loans[requestorAddress];
              uint loanCount = loanDetails.id;
              uint loanHist = (loanDetails.loanAmount / 10000000000000000000);
              emit loanCreditHistory(loanCount , loanHist);

              return (loanCount, loanHist );
           }

          // Check current Balance of the selected address
          function checkBalance(address addr) view public returns(uint){
              return (addr.balance / 10000000000000000000);

          }
```

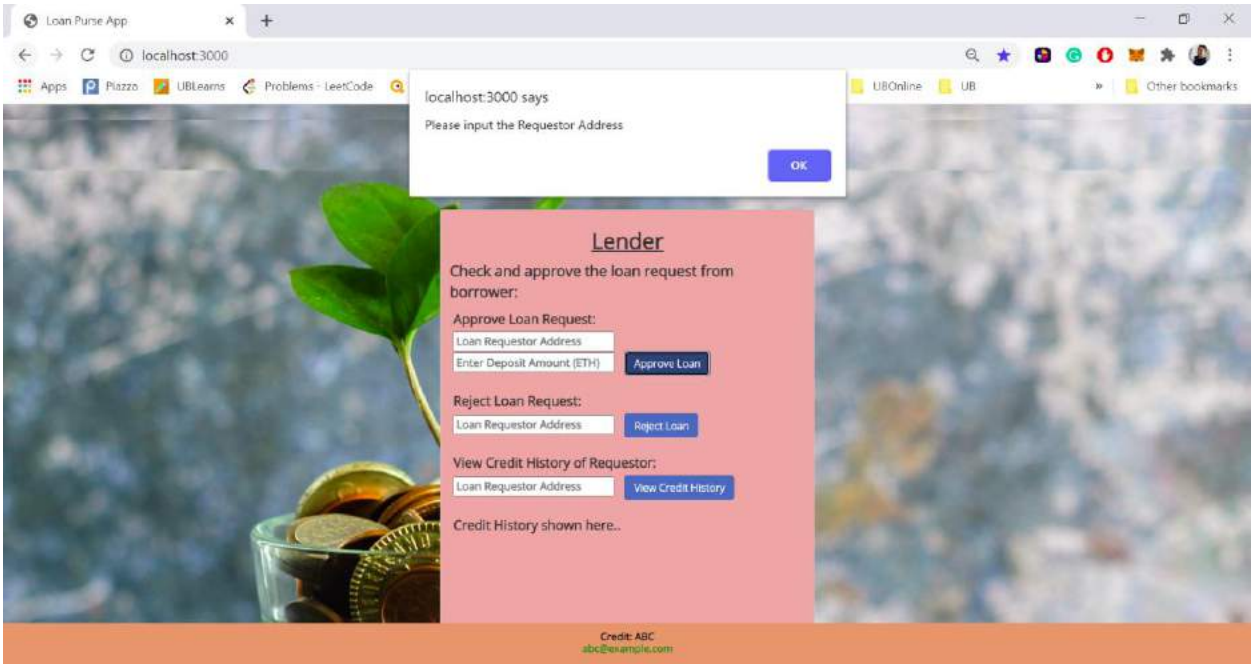
```
102
103         // function to check whether borrower is able to repay the amount to lender's account address
104         function calculateRepay(address requestorAddress, uint requestedLoanAmount) notLender public returns (uint){
105             // Calculating repay amount based on predefined interest rate on principal amount(loanAmount)
106             calRepayAmount = (((amount[requestorAddress] *10*2)/100) + requestedLoanAmount * 1000000000000000000)/ 1000000000000000000 ;
107             emit myCalc(calRepayAmount);
108
109             return calRepayAmount;
110         }
111
112         // function to repay the loan to the Lender
113         function repayLoan(address payable lenderAddress, uint repayAmt) notLender public payable {
114             require(msg.sender.balance >= repayAmt);
115             require(msg.value > repayAmt+1000000000000000000, "Insufficient Balance.");
116             require(repayAmt == calRepayAmount);
117
118             // transfer the calculated Repay Amount to Lender's account
119             lenderAddress.transfer(calRepayAmount * 1000000000000000000);
120
121             // Transfer the remaining amount to borrower's account after deducting the repay amount from msg.value amount
122             uint tempBalance = 0;
123             tempBalance = msg.value - (calRepayAmount * 1000000000000000000);
124             msg.sender.transfer(tempBalance);
125             loans[msg.sender].loanStatus = "Loan Amount Paid";
126
127             emit LoanRepaid();
128         }
129 }
                                                    ContractDefinition LoanPurse   0 reference(s)
```

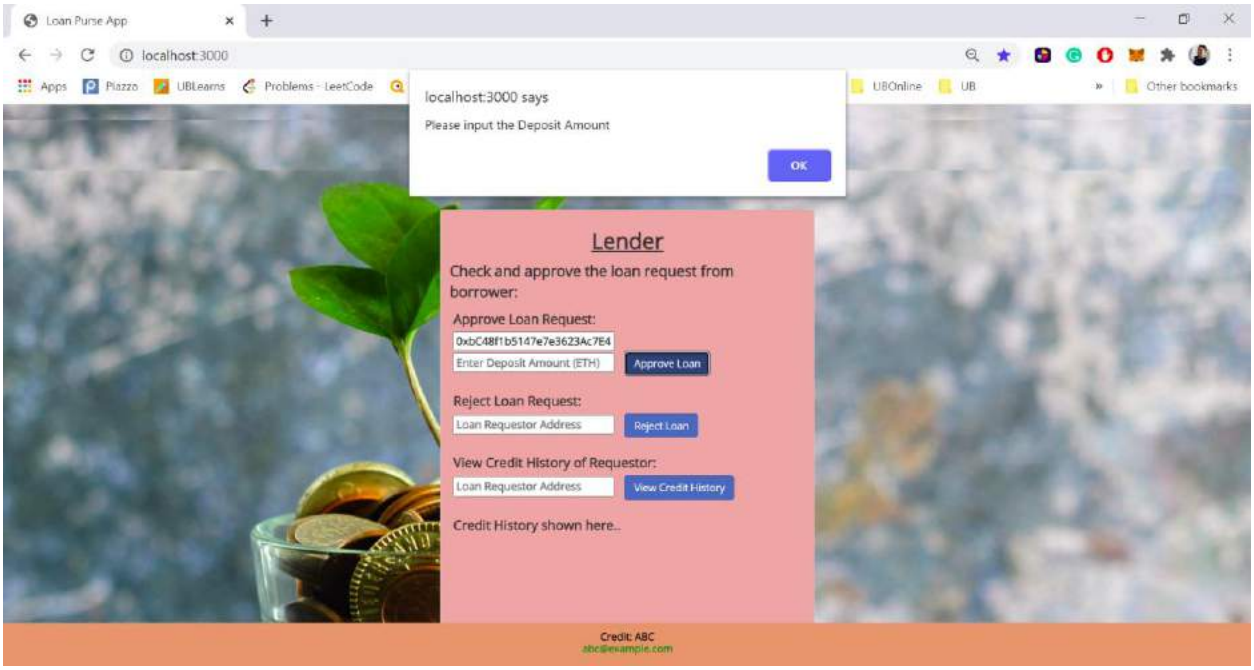
Web app(UI) screenshots:

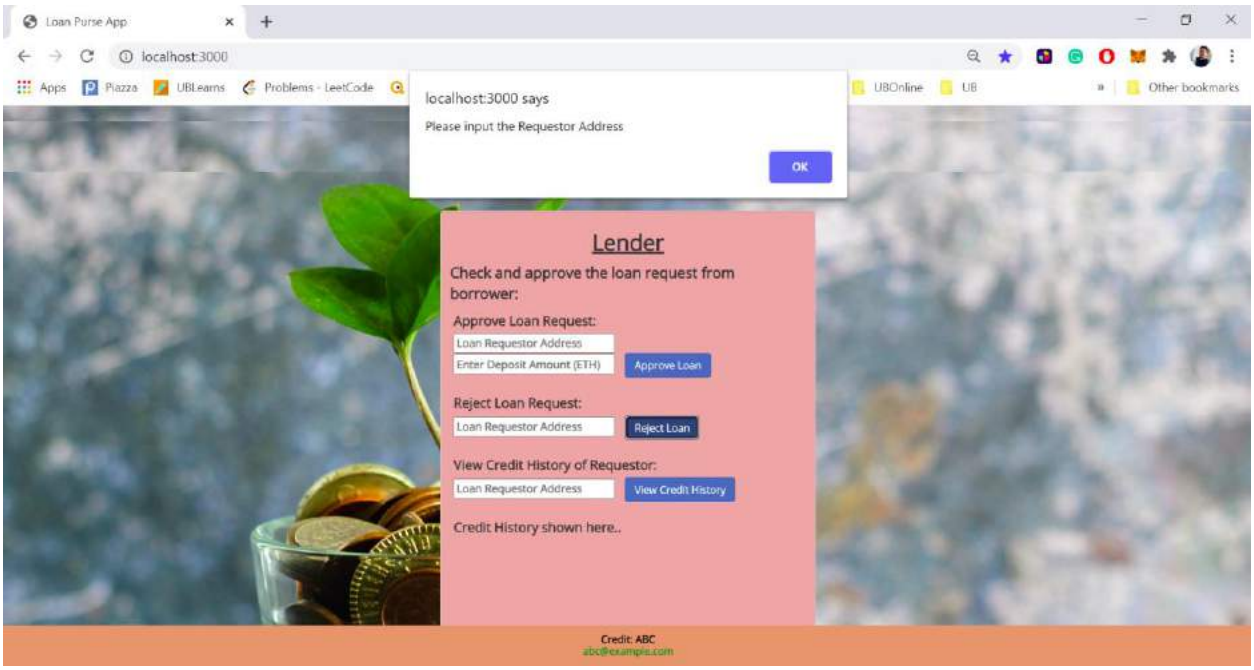
Lenders' Screen:

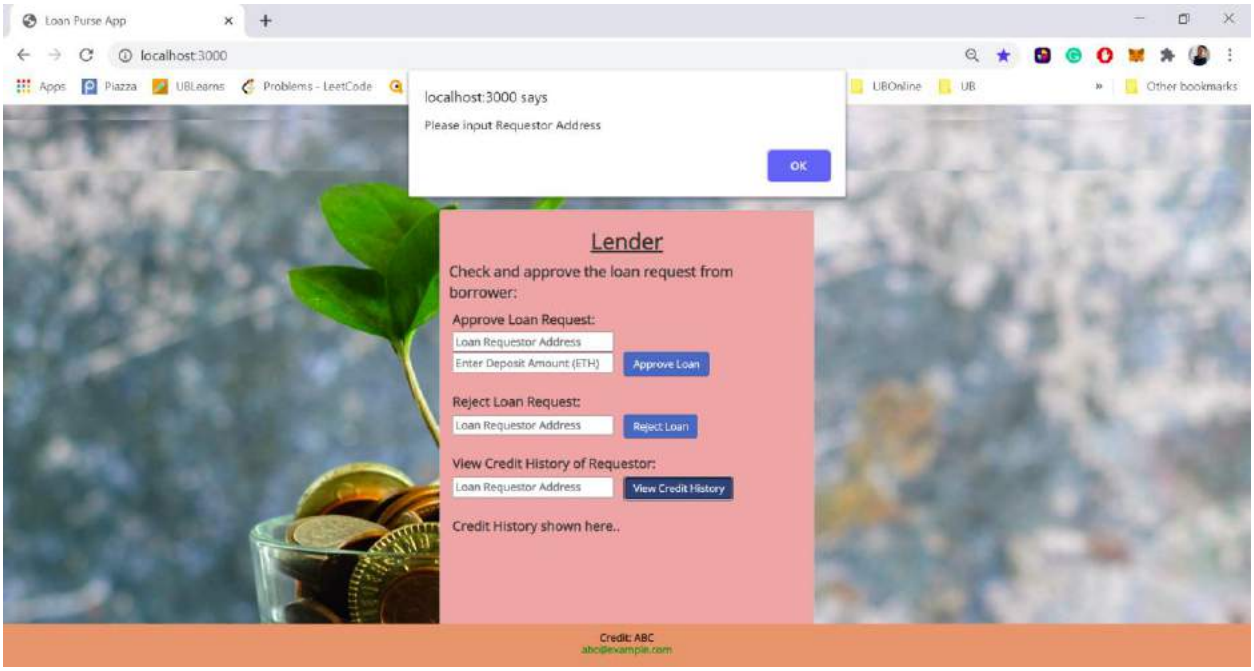


Validation of blank fields:

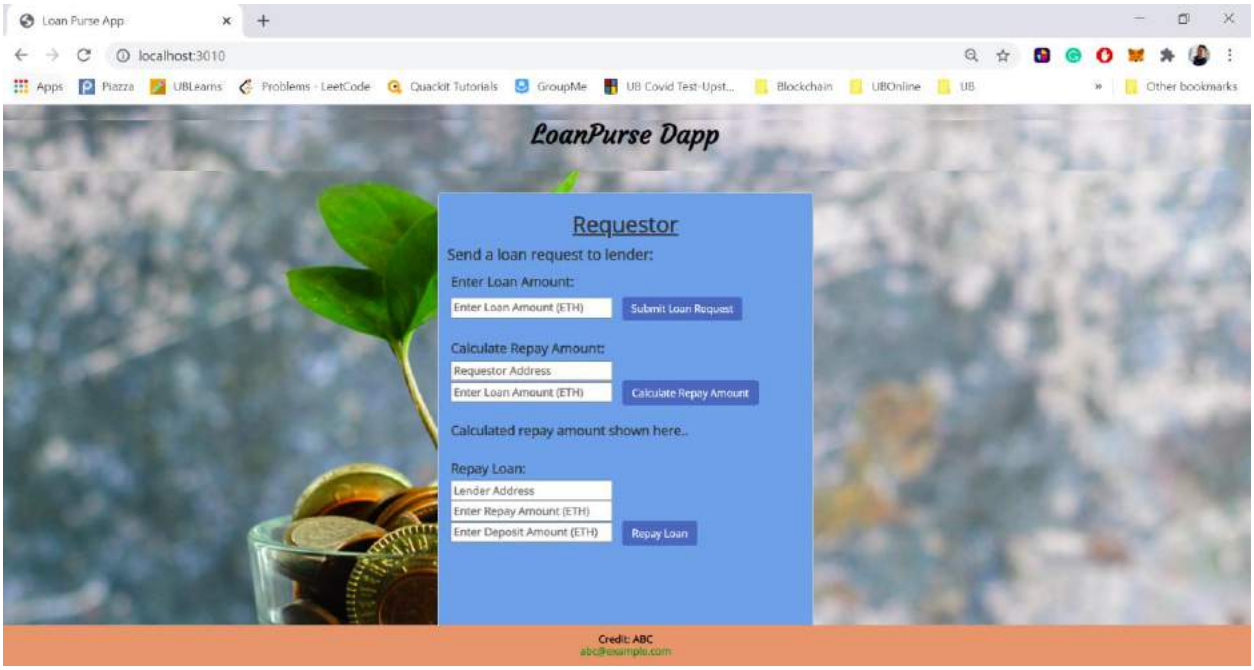




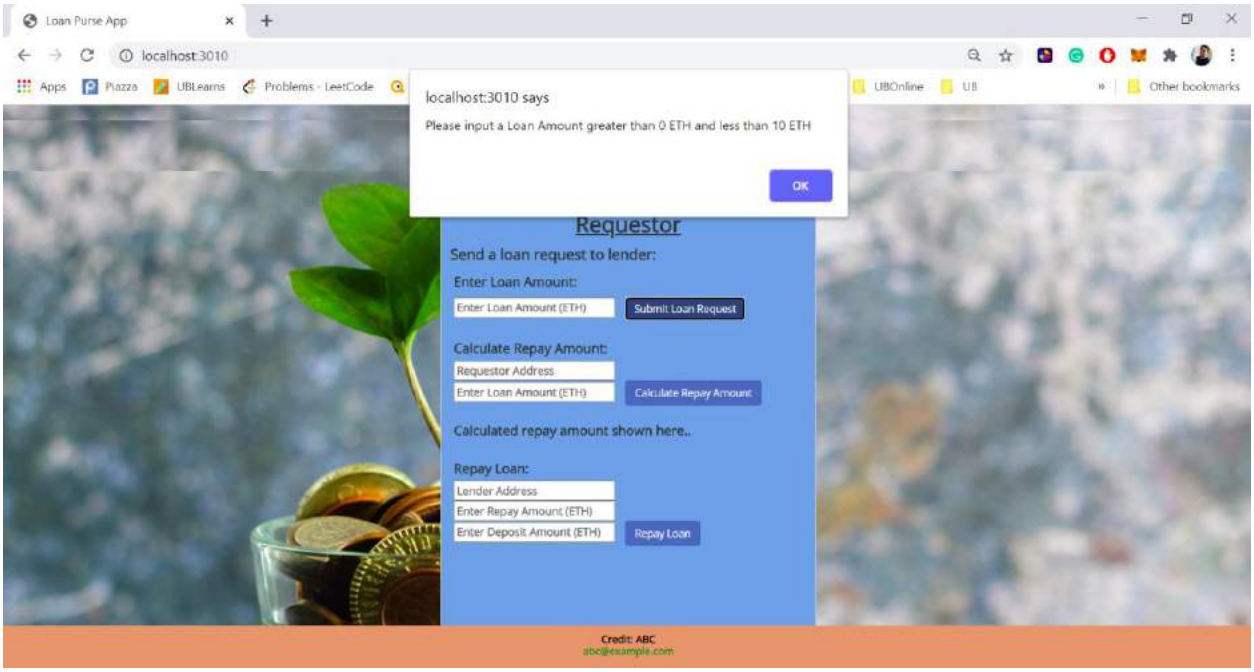

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
Problems - LeetCode
UBOnline
UB
Other bookmarks
localhost:3000 says
Please input the Requestor Address
OK
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

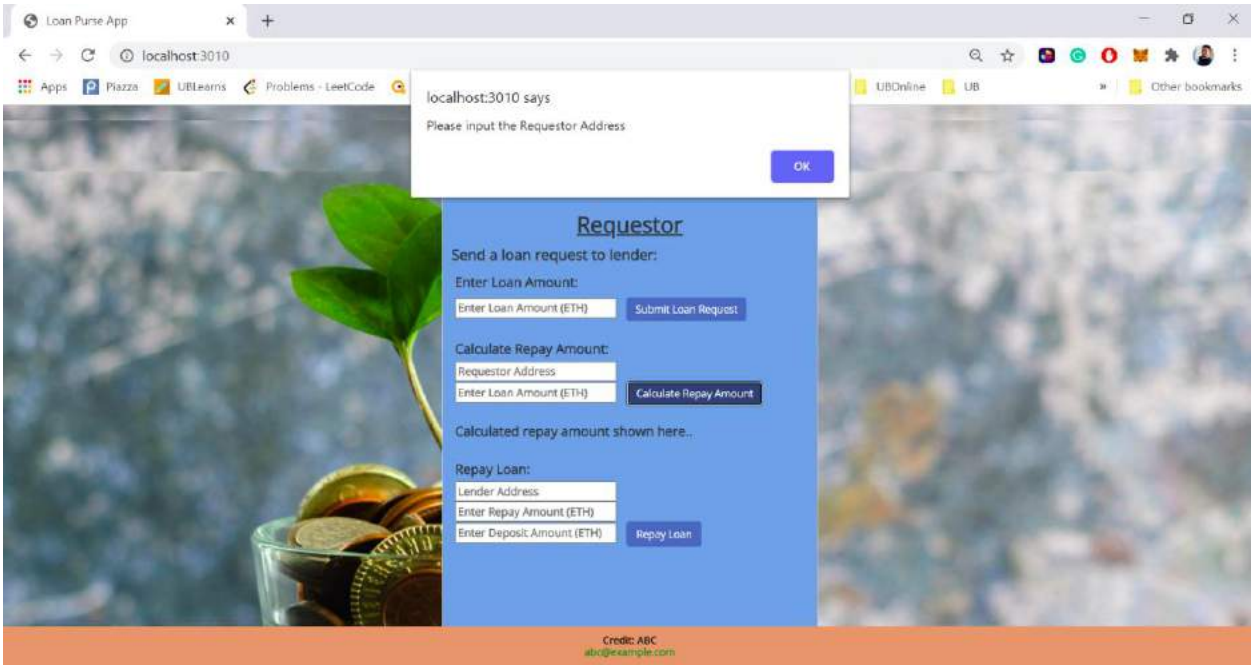

Loan Purse App
localhost:3000
Apps
Piazza
UBLearns
Problems - LeetCode
UBOnline
UB
Other bookmarks
localhost:3000 says
Please input Requestor Address
OK
Lender
Check and approve the loan request from borrower:
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH)
Approve Loan
Reject Loan Request:
Loan Requestor Address
Reject Loan
View Credit History of Requestor:
Loan Requestor Address
View Credit History
Credit History shown here..
Credit: ABC
abc@example.com

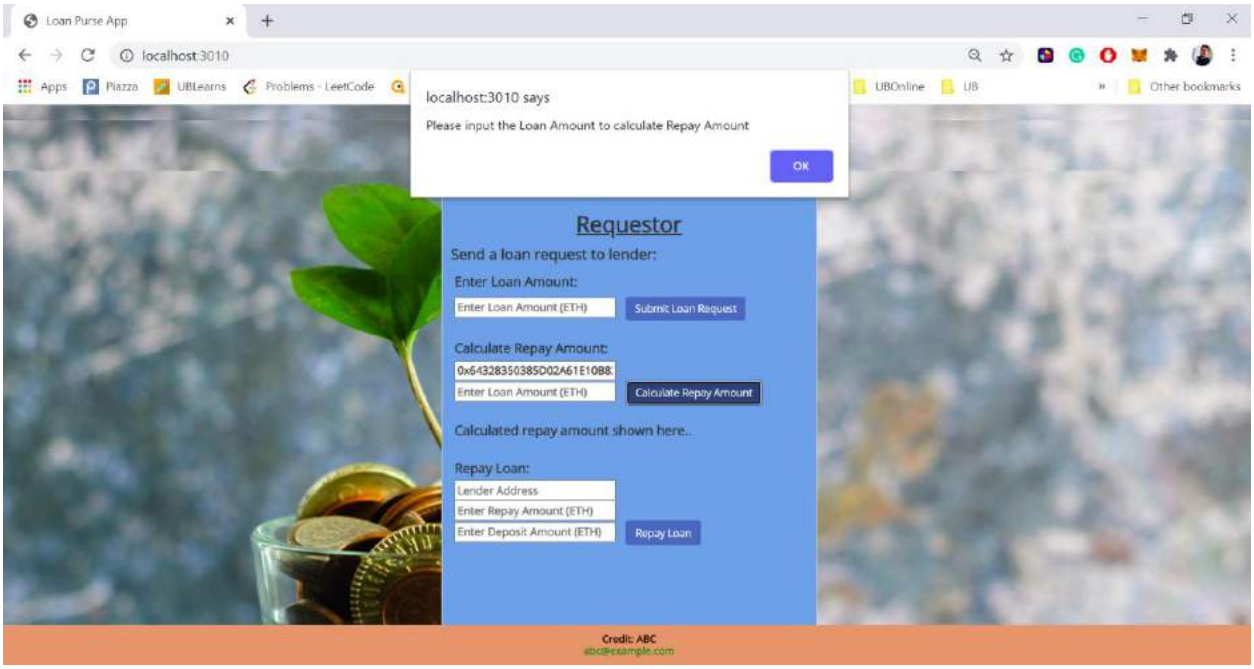
Requestors' screen:

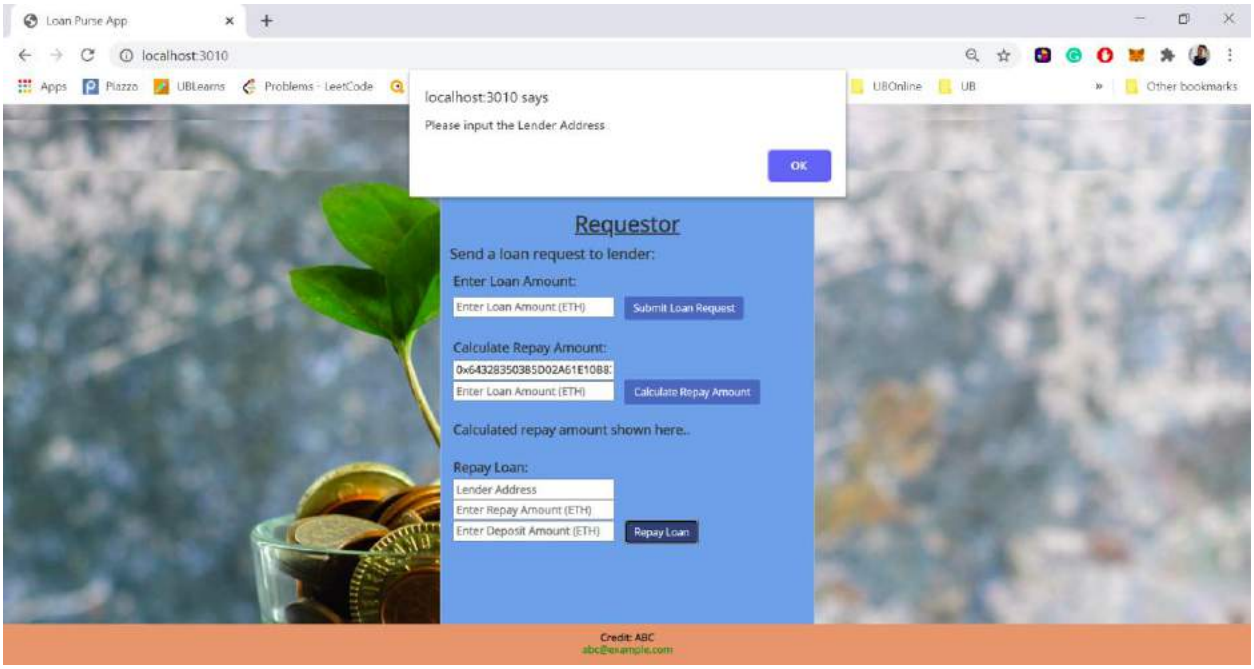


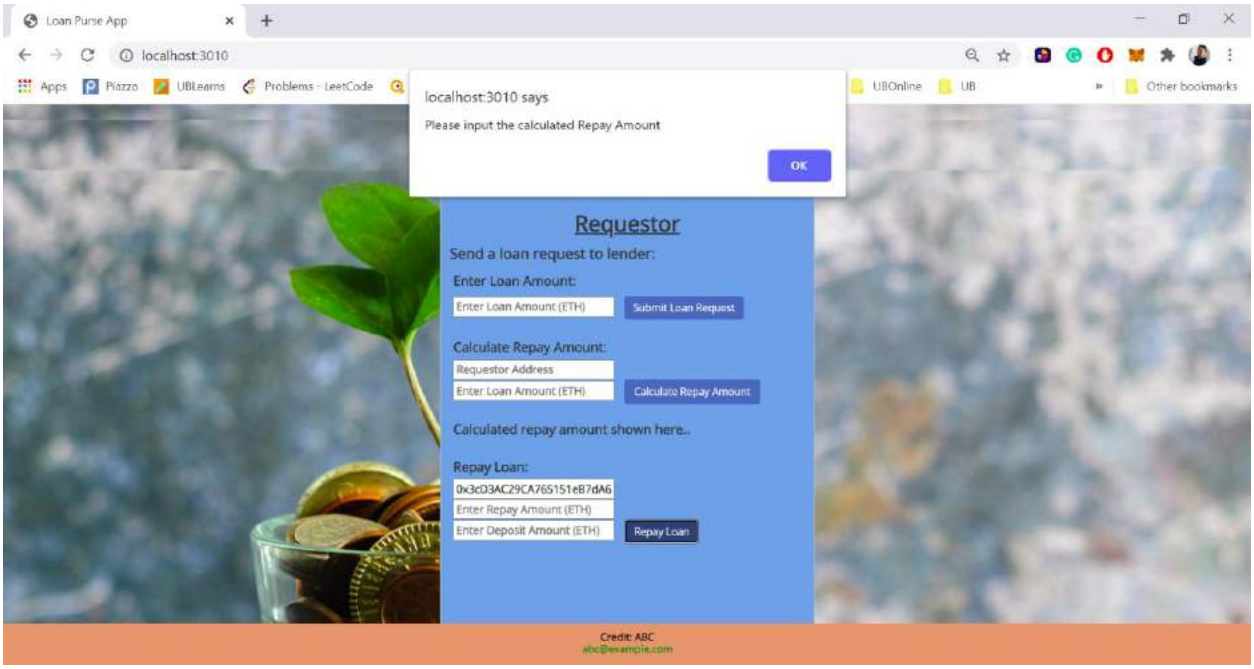
Validation of blank fields:

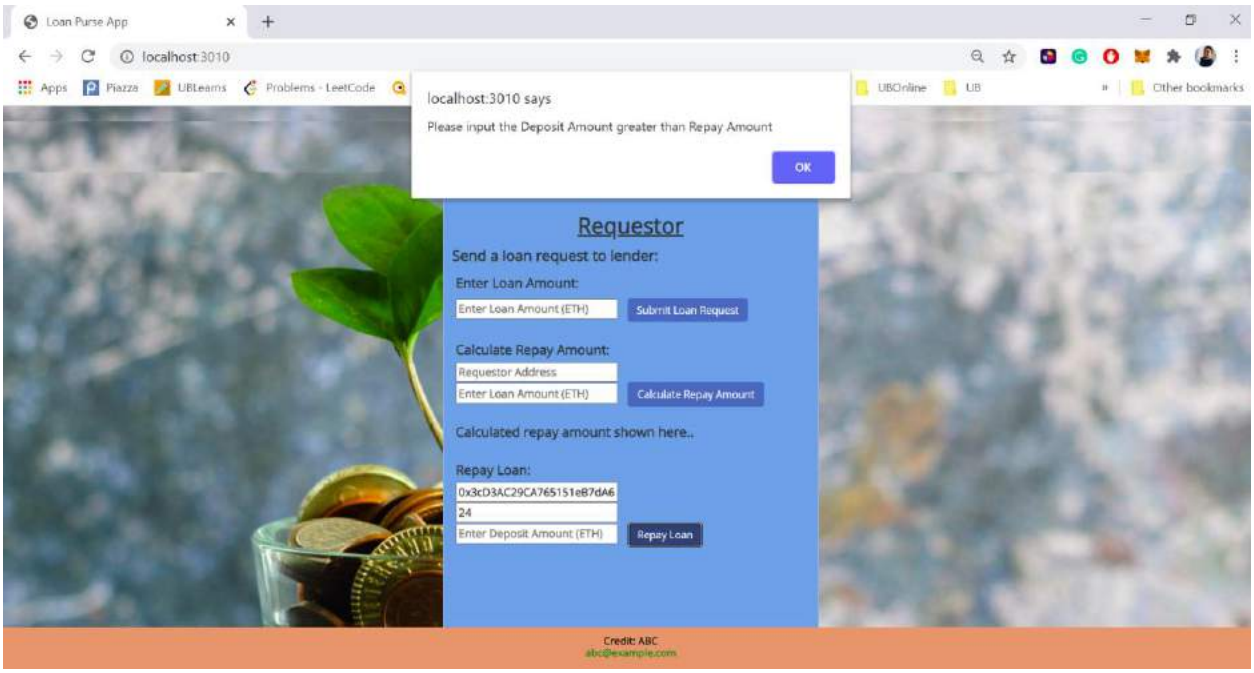



Loan Purse App
localhost:3010
Apps
Piazza
UBLearns
Problems - LeetCode
UBOnline
UB
Other bookmarks
localhost:3010 says
Please input the Requestor Address
OK
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
Requestor Address
Enter Loan Amount (ETH)
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Credit: ABC
abc@example.com


Loan Purse App
localhost:3010
Apps
Piazza
UBLearns
Problems - LeetCode
UBOnline
UB
Other bookmarks
localhost:3010 says
Please input the Loan Amount to calculate Repay Amount
OK
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0x64328350385D02A61E10B8
Enter Loan Amount (ETH)
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Credit: ABC
abc@example.com


Loan Purse App
localhost:3010
Apps
Piazza
UBLearns
Problems - LeetCode
UBOnline
UB
Other bookmarks
localhost:3010 says
Please input the Lender Address
OK
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
0x64328350385D02A61E10B8
Enter Loan Amount (ETH)
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
Lender Address
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Credit: ABC
abc@example.com


Loan Purse App
localhost:3010
Apps
Piazza
UBLearns
Problems - LeetCode
UBOnline
UB
Other bookmarks
localhost:3010 says
Please input the calculated Repay Amount
OK
Requestor
Send a loan request to lender:
Enter Loan Amount:
Enter Loan Amount (ETH)
Submit Loan Request
Calculate Repay Amount:
Requestor Address
Enter Loan Amount (ETH)
Calculate Repay Amount
Calculated repay amount shown here..
Repay Loan:
0x3c03AC29CA765151eB7dA6
Enter Repay Amount (ETH)
Enter Deposit Amount (ETH)
Repay Loan
Credit: ABC
abc@example.com



Conclusion:

The Loan Purse decentralized application(Dapp) has been successfully designed, implemented over different phases and finally deployed on the Infura cloud service for the use by the decentralized users.

Acknowledgement:

I would like to take an opportunity to thank the Professor Bina Ramamurthy for her constant support throughout the course and offering all the help while implementing this blockchain application term project. She has been a great mentor motivating each and everyone in the course to design their own end-to-end blockchain project along with the web-part implementation and deployment on Infura and providing all the support and guidance in completing this term project. I would also like to thank all the teaching assistants of this Blockchain course especially Chunwei Ma, who helped and provided constant support during his office hours to understand the project materials and answers to any queries that I had during different phases of this project. The course has provided a great learning in the blockchain domain with all the minute details for anyone who is even novice into this blockchain world. I am excited to have my own decentralized identity(Ropsten's account) now created during this course and hope to use the same in future for all the blockchain related activities.