

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

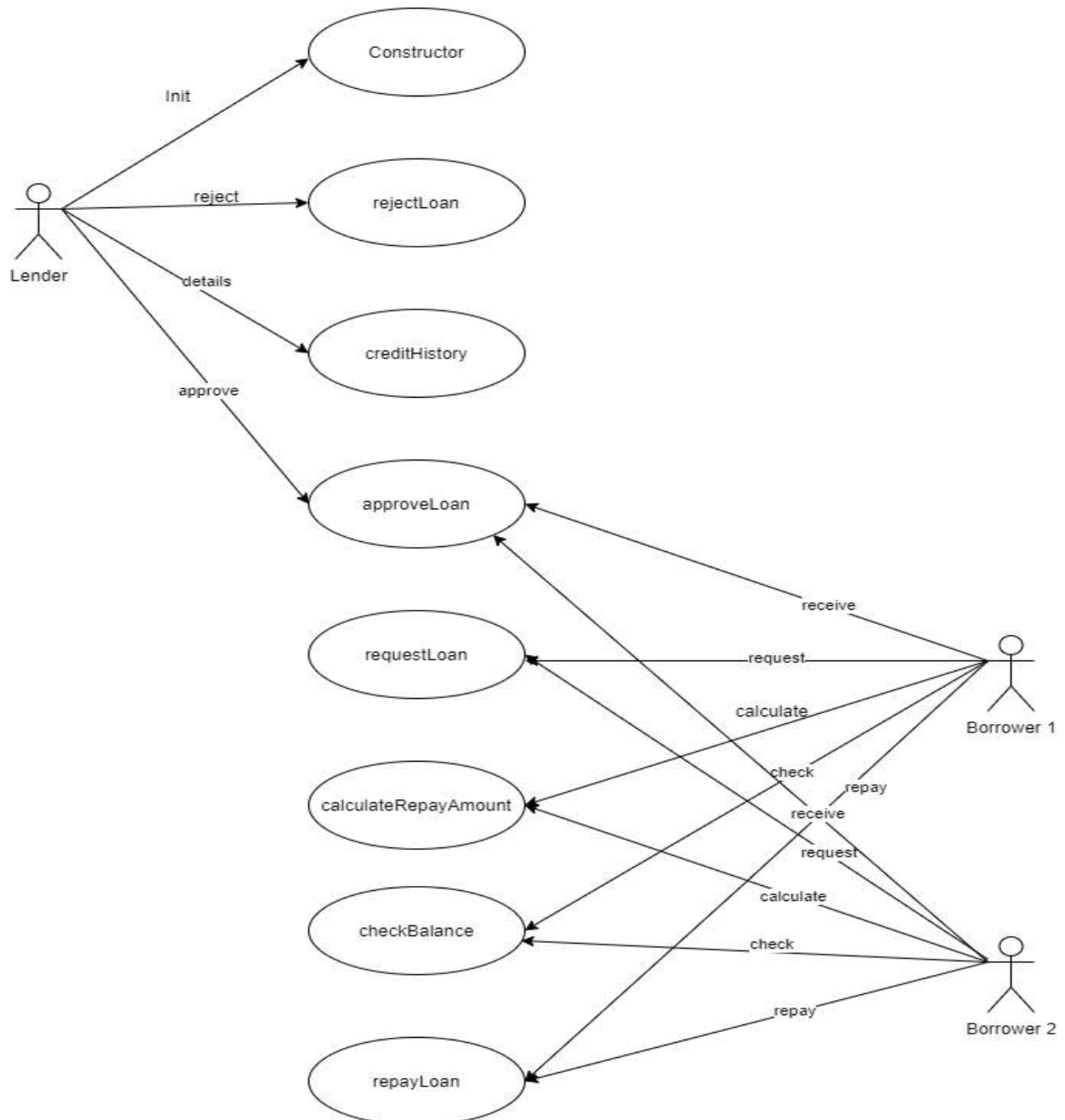
- **Major Area:** Financial Market Domain
- **Title:** Managing personal finance using Blockchain
- **Dapp name:** LoanPurse-Dapp
- **Clients:** Lenders, Borrowers(Requestors)
- **Abstract/Problem Statement:**
 - Financial markets are quite cumbersome for many of us and especially for the people who 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

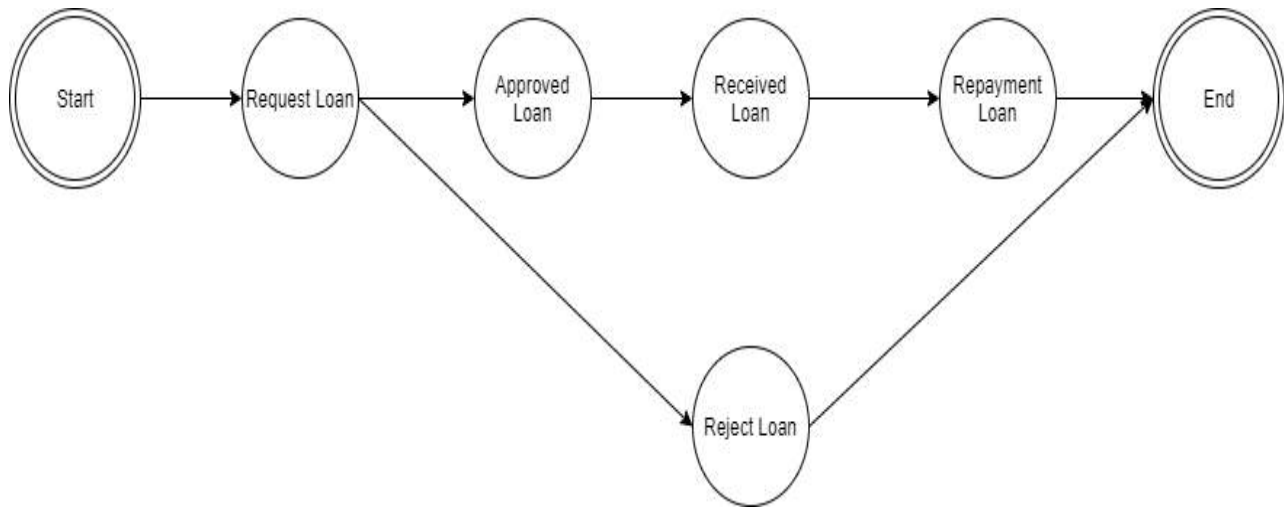
Design Diagrams

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

1) Use case diagram:



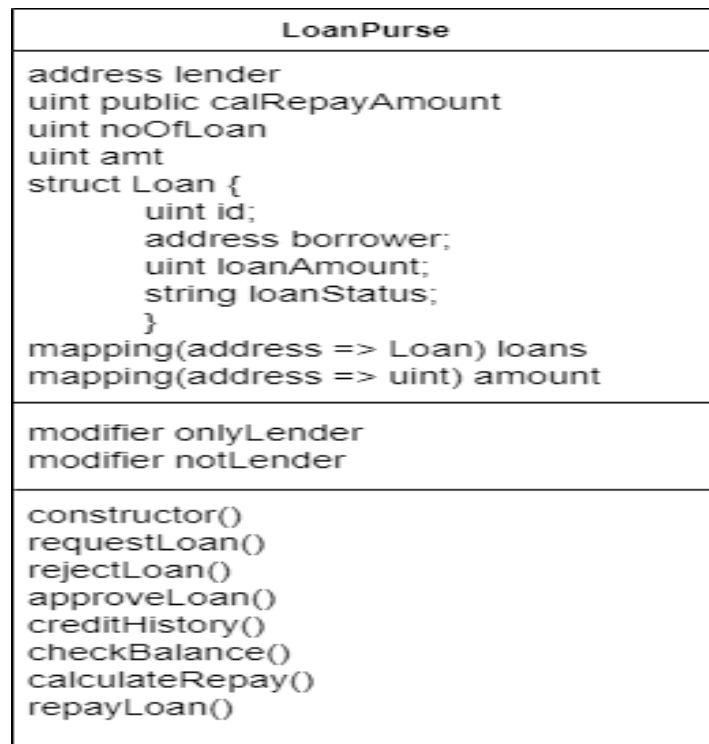
2) Finite State Machine (FSM) diagram:



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:



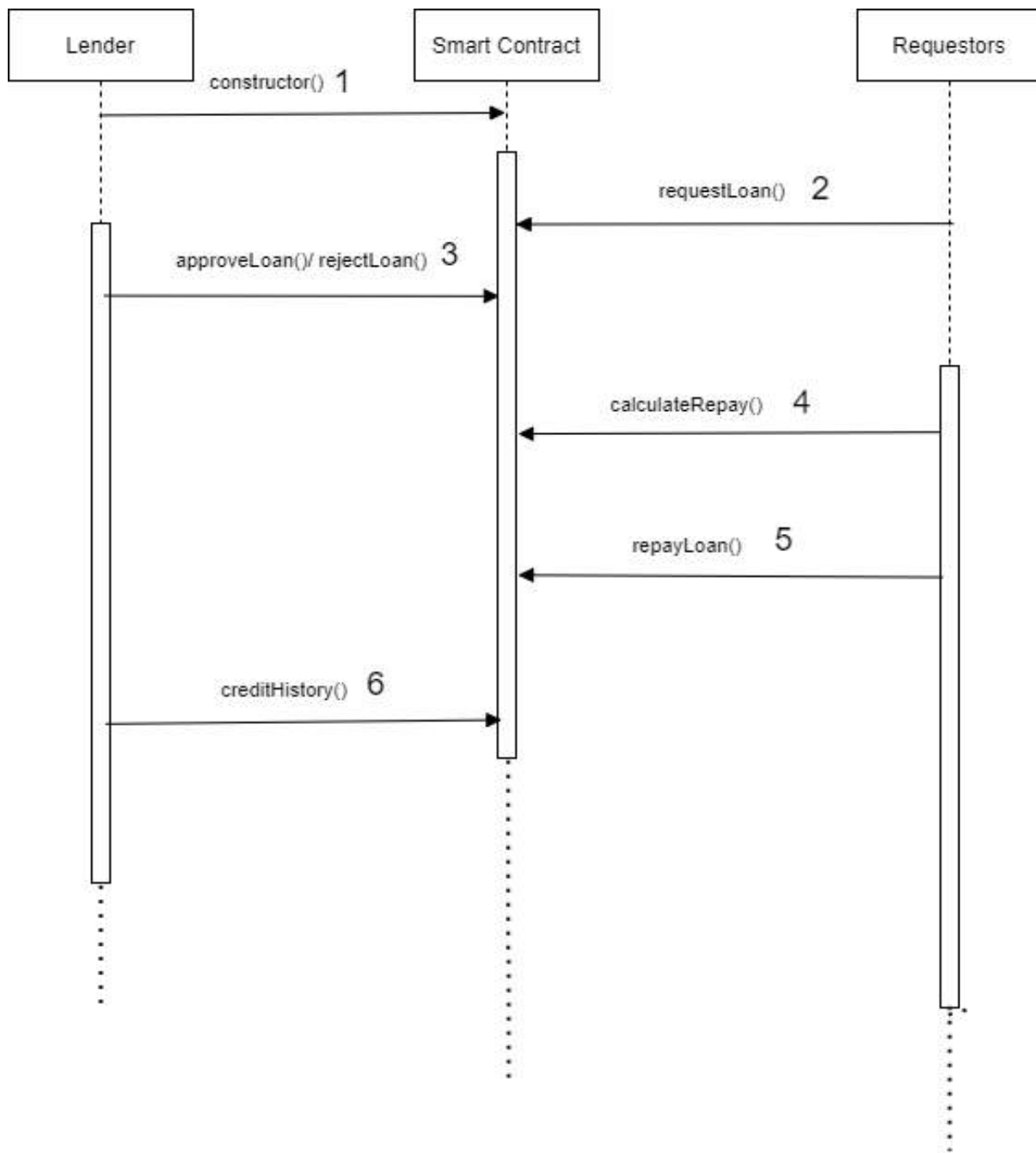
Modifiers:

- 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:

The screenshot displays the 'DEPLOY & RUN TRANSACTIONS' interface. Under the 'Deployed Contracts' section, a contract named 'LOANPURSE AT 0XD91...39138 (MEMORY)' is selected. Below this, a list of functions is shown, each with a colored button and a dropdown menu for arguments.

Function	Arguments
approveLoan	address requestorAddress
calculateRepay	address requestorAddress, uint256 requ
creditHistory	address requestorAddress
rejectLoan	address requestorAddress
repayLoan	address lenderAddress, uint256 repayAr
requestLoan	uint256 loanAmount
amount	address
calRepayAmount	
checkBalance	address addr
lender	

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

requestLoan

loanAmount: 20

transact

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

creditHistory

requestorAddress: 0xAb8483F64d9C6d1EcF9b849Ae677

transact

logs


```
[ { "from": "0xd9145CCE52D386f254917e481e844e9943F39138", "topic":  
"0x47136f20f9e1c025bcc0543df66a691d4b1f4c09109165fd3b8e0e1f3c8303df", "event":  
"loanCreditHistory", "args": { "0": "0", "1": "0", "loanCount1": "0", "loanAmt1":  
"0" } } ]
```


value 0 wei



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

approveLoan

requestorAddress:

 **transact**

ACCOUNT 

- 0x5B3...eddC4 (79.9999999999998280681 ether)
- 0xAb8...35cb2 (119.999999999999873044 ether)**
- 0x4B2...C02db (100 ether)
- 0x787...cabaB (100 ether)
- 0x617...5E7f2 (100 ether)
- 0x17F...8c372 (100 ether)
- 0x5c6...21678 (100 ether)

amount

0xAb8483F64d9C6d1EcF9b849Ae677i

0: uint256: 20000000000000000000

calRepayAmount

checkBalance

0xAb8483F64d9C6d1EcF9b849Ae677i

0: uint256: 119

lender

0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

➤ Credit history shows the details of loan request:

```


logs      [ { "from": "0xd9145CCE52D386f254917e481eB44e9943F39138", "topic":
           "0x47136f20f9e1c025bcc0543df66a691d4b1f4c09109165fd3b8e0e1f3c8303df", "event":
           "loanCreditHistory", "args": { "0": "5", "1": "20", "loanCount1": "5", "loanAmt1":
           "20" } } ]
value     0 wei

```

- Another Requestor requests for loan and this time Lender rejects the loan request:


requestLoan

loanAmount:

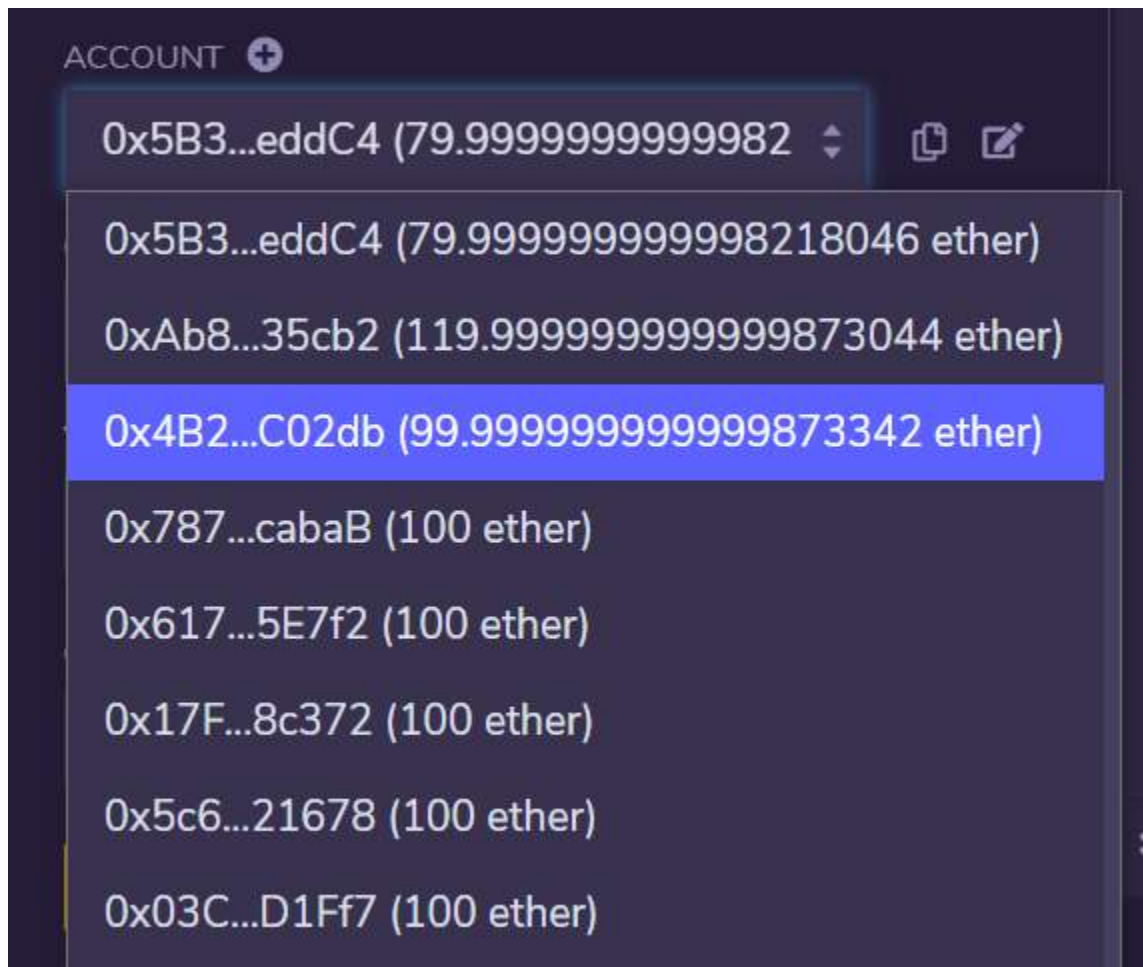
 **transact**

rejectLoan

requestorAddress:

 **transact**

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

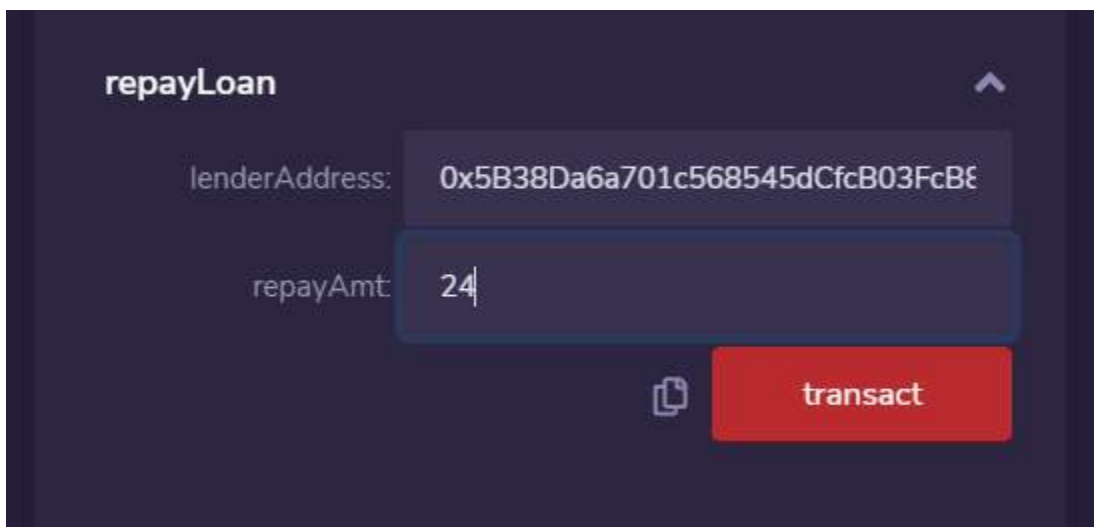


- Requestor calculates the repay amount to be paid (e.g., repay amount on 20 Ether loan amount):

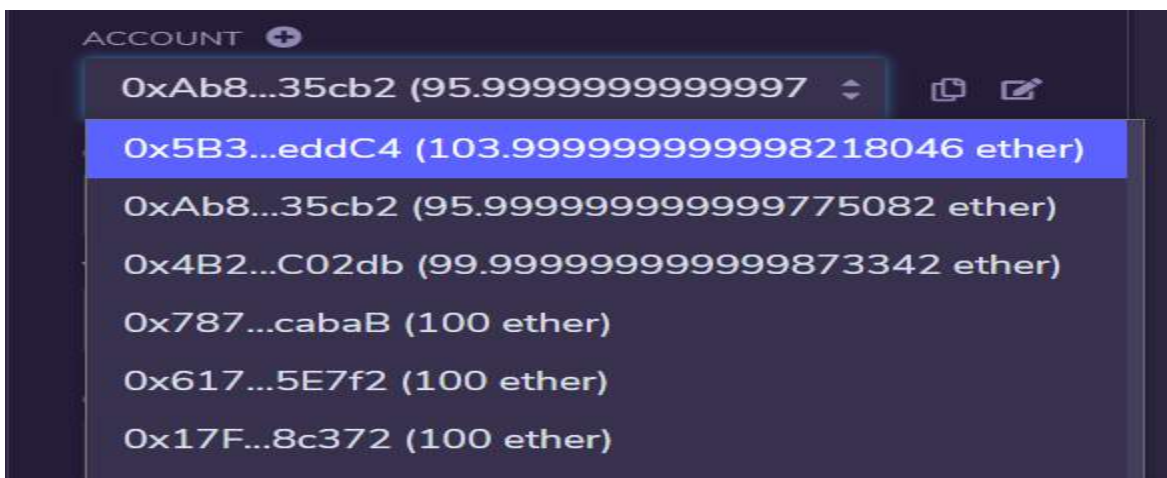
A screenshot of a web application interface showing a form titled 'calculateRepay'. The form has two input fields: 'requestorAddress' with the value '0xAb8483F64d9C6d1EcF9b849Ae677' and 'requestedLoanAmount' with the value '20'. There is a 'transact' button and a copy icon.



- 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):

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is visible, showing the environment set to 'JavaScript VM', account '0xA8...35cb2', gas limit '3000000', and value '0'. The contract 'LoanPurse - browser/LoanPurse.sol' is selected. The 'Deploy' button is highlighted. On the right, the Solidity code for 'LoanPurse.sol' is displayed, showing functions like 'creditHistory', 'checkBalance', and 'calculateRepay'. Below the code, the transaction log shows a failed transaction: 'transact to LoanPurse.approveLoan pending ...' followed by an error message: '[vm] from: 0xA8...35cb2 to: LoanPurse.approveLoan(address) 0xd91...39138 value: 0 wei data: 0xac6...35cb2 logs: 0 hash: 0x777...26456'. The error message states: 'transact to LoanPurse.approveLoan errored; VM error: revert. revert The transaction has been reverted to the initial state. Note: the called function should be payable if you send value and the value you send should be less than your current balance. Debug the transaction to get more information.'

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is visible, showing the environment set to 'JavaScript VM', account '0xA8...35cb2', gas limit '3000000', and value '0'. The contract 'LoanPurse - browser/LoanPurse.sol' is selected. The 'Deploy' button is highlighted. On the right, the Solidity code for 'LoanPurse.sol' is displayed, showing functions like 'creditHistory', 'checkBalance', and 'calculateRepay'. Below the code, the transaction log shows a failed transaction: 'transact to LoanPurse.rejectLoan pending ...' followed by an error message: '[vm] from: 0xA8...35cb2 to: LoanPurse.rejectLoan(address) 0xd91...39138 value: 0 wei data: 0xf53...c02db logs: 0 hash: 0xfab...c0b77'. The error message states: 'transact to LoanPurse.rejectLoan errored; VM error: revert. revert The transaction has been reverted to the initial state. Note: The called function should be payable if you send value and the value you send should be less than your current balance. Debug the transaction to get more information.'

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

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is visible with the environment set to 'JavaScript VM', account '0x5B3...eddC4', gas limit '3000000', and value '0'. The contract 'LoanPurse - browser/LoanPurse.sol' is selected. The main editor shows the Solidity code for 'LoanPurse.sol'. The transaction log at the bottom indicates a failed transaction: 'transact to LoanPurse.requestLoan pending ...' followed by an error: '[vm] from: 0x5B3...eddC4 to: LoanPurse.requestLoan(uint256) 0xd91...39138 value: 0 wei data: 0xd55...0001e logs: 0 hash: 0x78a...d0b5c'. The error message states: 'transact to LoanPurse.requestLoan errored: VM error: revert. revert The transaction has been reverted to the initial state. Note: The called function should be payable if you send value and the value you send should be less than your current balance. Debug the transaction to get more information.'

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is visible with the environment set to 'JavaScript VM', account '0x5B3...eddC4', gas limit '3000000', and value '0'. The contract 'LoanPurse - browser/LoanPurse.sol' is selected. The main editor shows the Solidity code for 'LoanPurse.sol'. The transaction log at the bottom indicates a failed transaction: 'transact to LoanPurse.repayLoan pending ...' followed by an error: '[vm] from: 0x5B3...eddC4 to: LoanPurse.repayLoan(address,uint256) 0xd91...39138 value: 0 wei data: 0xd55...0001e logs: 0 hash: 0x78a...d0b5c'. The error message states: 'transact to LoanPurse.repayLoan errored: VM error: revert. revert The transaction has been reverted to the initial state. Note: The called function should be payable if you send value and the value you send should be less than your current balance. Debug the transaction to get more information.'

Code snippets of the LoanPurse.sol

- Variables, mapping, struct, and modifiers:

```
1 pragma solidity >=0.4.22 <=0.6.0;
2
3 contract LoanPurse {
4     address payable public lender; // owner of the contract who deploy the contract
5     uint public calRepayAmount; // calculate the Repay Amount based on predefined interest rate
6     uint noOfLoan=0; // count of total loans
7     uint amt;
8     mapping(address => Loan) loans; // mapping address to loan
9     mapping(address => uint) public amount;
10
11     // Loan structure
12     struct Loan {
13         uint id;
14         address borrower;
15         uint loanAmount;
16         string loanStatus;
17     }
18
19     // Events are defined here
20     event Initialized();
21     event LoanRequested();
22     event LoanApproved();
23     event LoanRejected();
24     event LoanRepaid();
25     event myCalc(uint myRepay);
26     event loanCredithistory(uint loanCount1, uint loanAmt1);
27
28
29     // modifier to check the functions which are applicable to only Lender can be done
30     // by Lender (e.g. approveLoan, rejectLoan)
31     modifier onlyLender()
32     { require(msg.sender == lender);
33       _;
34     }
35
36     // modifier to check the functions which are not applicable to Lender can not be done
37     // by Lender (e.g. requestLoan, repayLoan)
38     modifier notLender()
39     { require(msg.sender != lender);
40       _;
41     }
42 }
```

➤ 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     emit Initialized();
48 }
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 * 1000000000000000000;
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 {...}
62
63 // function to reject the loan request by Lender
64 function rejectLoan(address requestorAddress) onlyLender public {...}
65
66 function creditHistory(address requestorAddress) public returns(uint , uint) {...}
67
68 // Check current Balance of the selected address
69 function checkBalance(address addr) view public returns(uint){...}
70
71 // function to check whether borrower is able to repay the amount to Lender's account address
72 function calculateRepay(address requestorAddress, uint requestedLoanAmount) notLender public returns (uint){...}
73
74 function repayLoan(address payable lenderAddress, uint repayAmt) notLender payable public {...}
75 }
```

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);
```

```

42
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 * 1000000000000000000;
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 {
62     require(msg.sender == lender);
63     require(msg.value > amt, "Insufficient Balance.");
64     amt = amount[requestorAddress];
65     requestorAddress.transfer(amt);
66
67     uint tempBalance = 0;
68     tempBalance = msg.value - amt;
69     msg.sender.transfer(tempBalance);
70
71     noOfLoan++;
72     loans[requestorAddress] = Loan(noOfLoan, requestorAddress, amt, "Loan Request Approved");
73     emit LoanApproved();
74 }
75
76 // function to reject the loan request by Lender
77 function rejectLoan(address requestorAddress) onlyLender public {
78     require(msg.sender == lender);
79     loans[requestorAddress].loanStatus="Loan Request Rejected";
80
81     emit LoanRejected();
82 }
83

```

```

84
85 function creditHistory(address requestorAddress) public returns(uint , uint) {
86     Loan memory loanDetails = loans[requestorAddress];
87     uint loanCount = loanDetails.id;
88     uint loanHist = (loanDetails.loanAmount / 1000000000000000000);
89     emit loanCreditHistory(loanCount , loanHist);
90
91     return (loanCount, loanHist);
92 }
93
94 // Check current Balance of the selected address
95 function checkBalance(address addr) view public returns(uint){
96     return (addr.balance / 1000000000000000000);
97 }
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){
101     // Calculating repay amount based on predefined interest rate on principal amount(LoanAmount)
102     calRepayAmount = (((amount[requestorAddress] *10*2)/100) + requestedLoanAmount * 1000000000000000000) / 1000000000000000000 ;
103     emit myCalc(calRepayAmount);
104
105     return calRepayAmount;
106 }
107
108

```

```

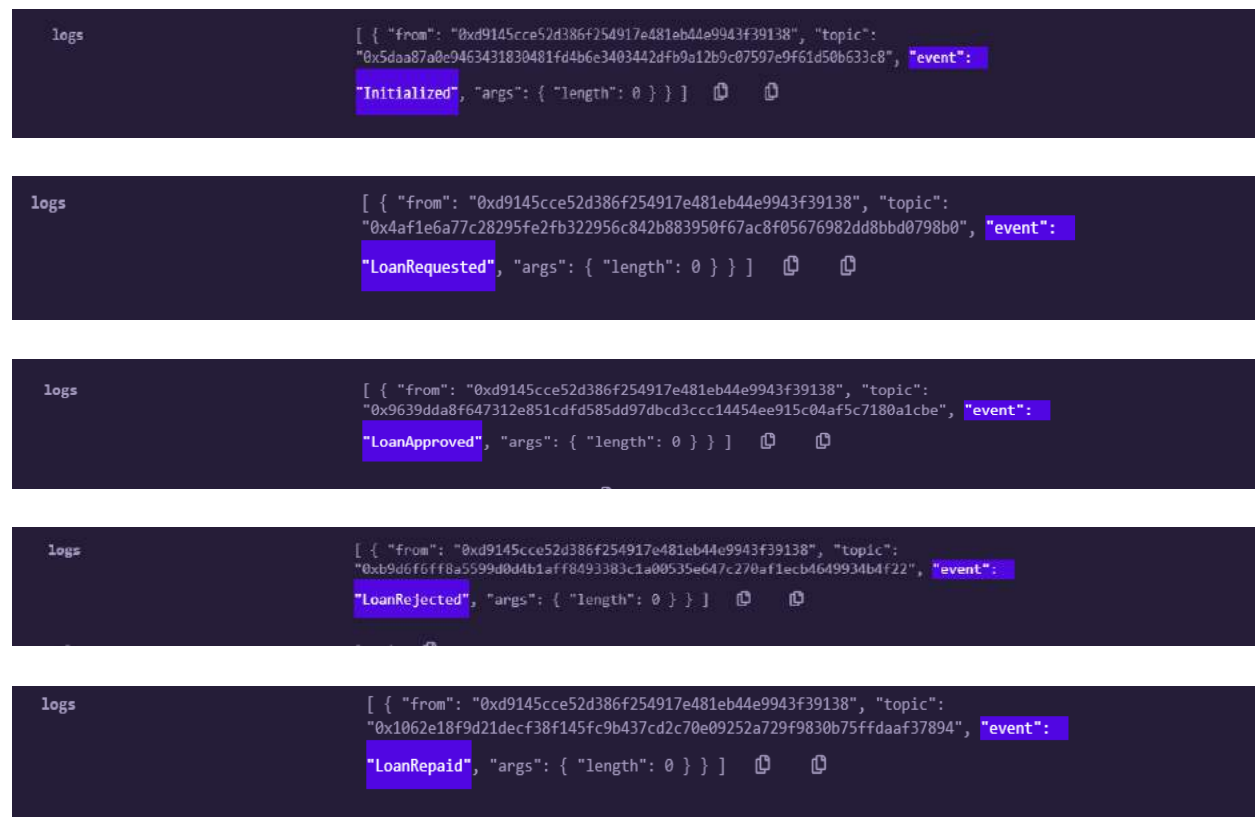
109 function repayLoan(address payable lenderAddress, uint repayAmt) notLender payable public {
110     require(msg.sender.balance >= repayAmt);
111     require(msg.value > repayAmt+1000000000000000000, "Insufficient Balance.");
112     require(repayAmt == calRepayAmount);
113
114     // transfer the calculated Repay Amount to Lender's account
115     lenderAddress.transfer(calRepayAmount * 1000000000000000000);
116
117     // Transfer the remaining amount to borrower's account after deducting the repay amount from msg.value amount
118     uint tempBalance = 0;
119     tempBalance = msg.value - (calRepayAmount * 1000000000000000000);
120     msg.sender.transfer(tempBalance);
121     loans[msg.sender].loanStatus = "Loan Amount Paid";
122
123     emit LoanRepaid();
124 }
125 }

```

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:



```
logs [ { "from": "0xd9145CCE520386f254917e481e844e9943f39138", "topic":  
      "0x0188859915b2d59d30b054f2a8a3c2a14910b0f068975dc9fba7fced31e4e5aa", "event": "myCalc", "args": [ "0": "24", "myRepay": "24"  
    ] } ]
```

```
logs [ { "from": "0xd9145CCE520386f254917e481e844e9943f39138", "topic":  
      "0x47136f20f9e1c025bcc0543df66a691d4b1f4c09109165fd3b8e0e1f3c8303df", "event": "loanCreditHistory", "args": { "0": "5", "1":  
      "20", "loanCount1": "5", "loanAmt1": "20" } } ]
```

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.

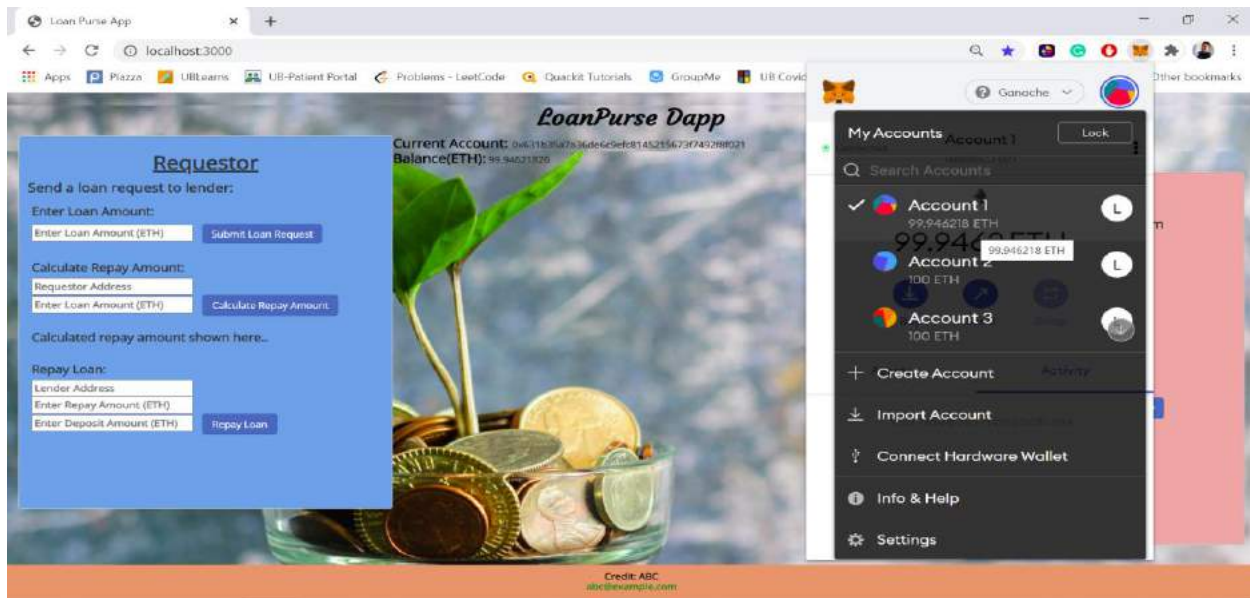
Phase 5

Web App Development (Dapp)

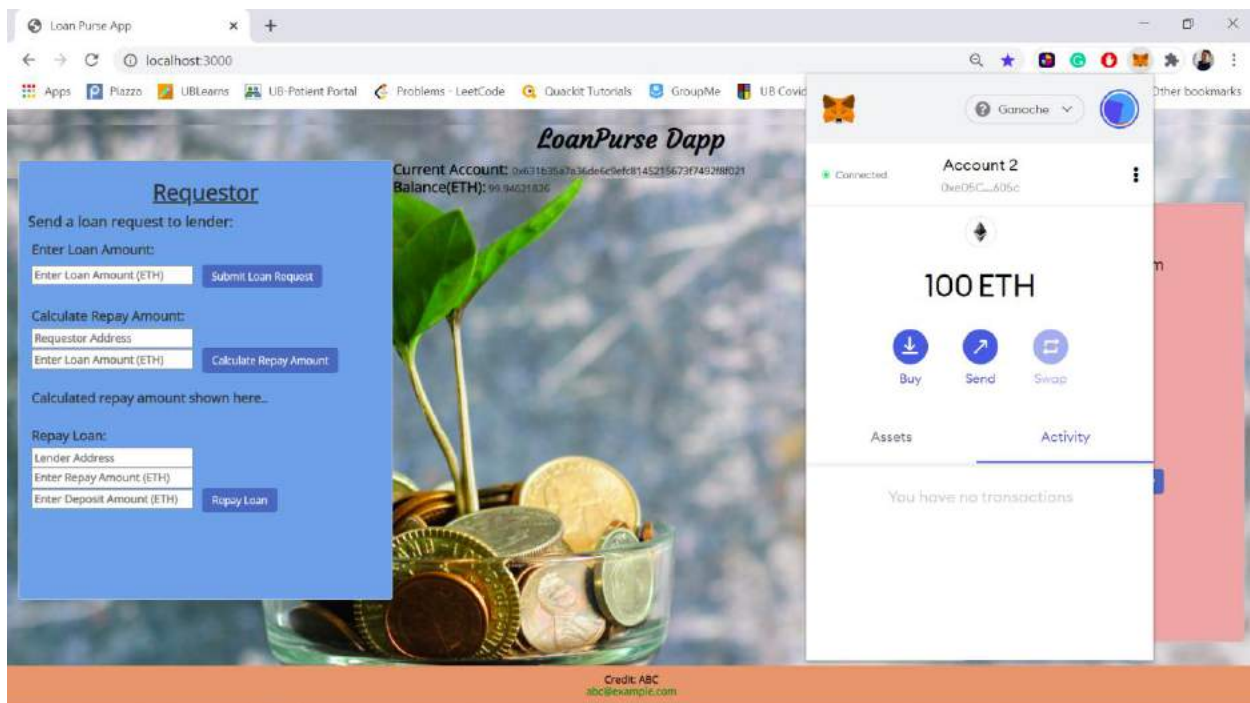
The below screenshot shows the web part of the LoanPurse Dapp deployed on localhost server using Ganache. The web application consists of all the required fields as part of loan request and approval process. The corresponding fields to each of the stakeholders like Requestor and Lender are shown categorically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

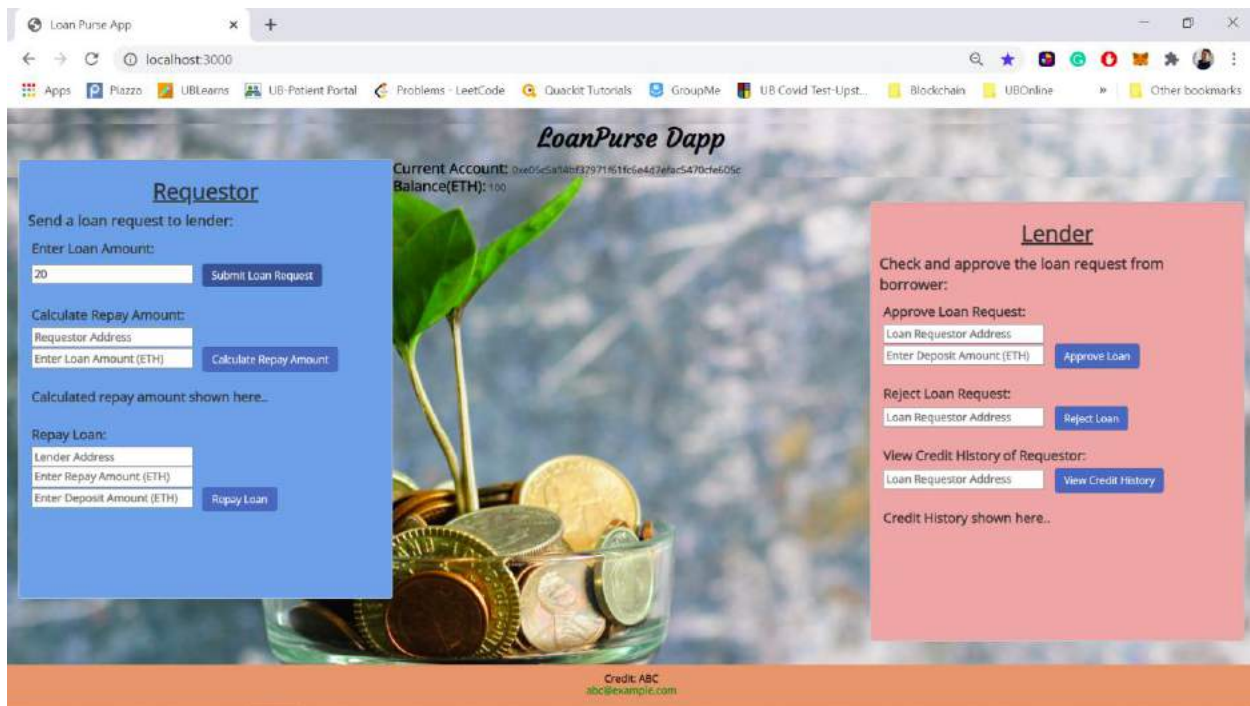
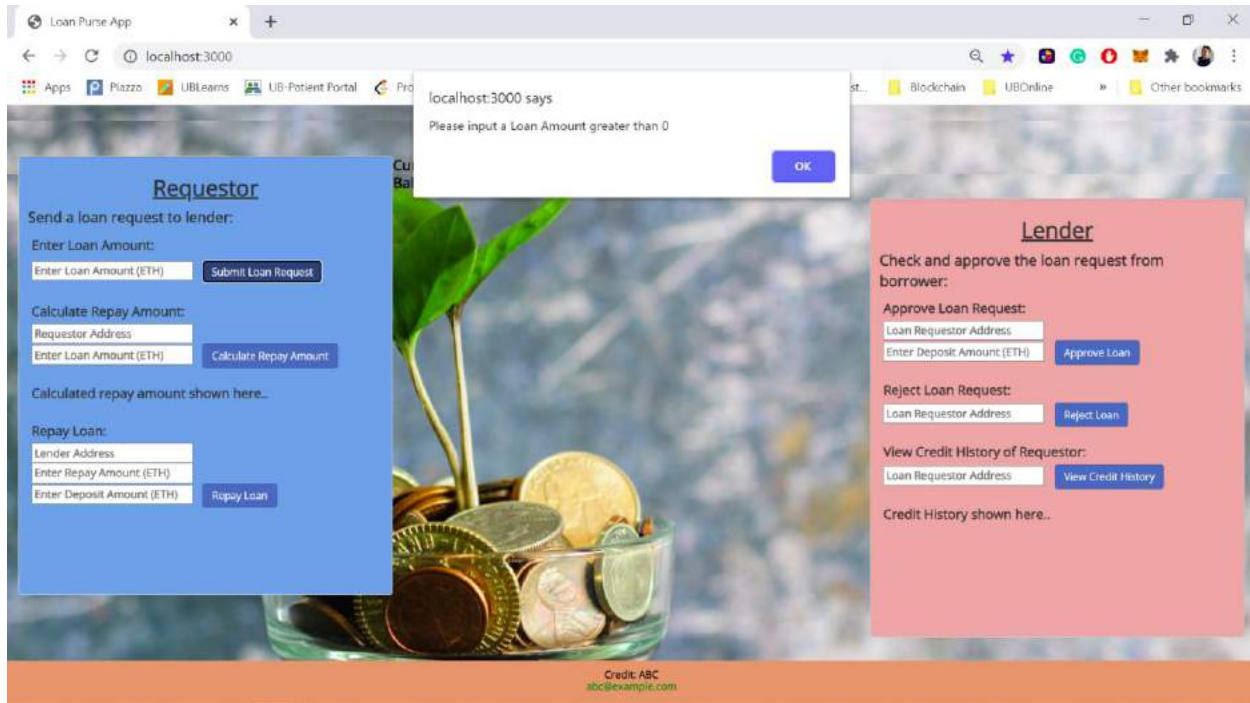
The screenshot displays the LoanPurse Dapp web interface in a browser window. The browser's address bar shows 'localhost:3000'. The page features a background image of a small green plant growing out of a glass jar filled with various coins. The interface is divided into two main colored panels: a blue 'Requestor' panel on the left and a pink 'Lender' panel on the right. At the top center, the text 'LoanPurse Dapp' is displayed. Below it, the 'Current Account' details are shown: 'Current Account: 0x6c31b35a7a36d6c9efc8145215673f745268021' and 'Balance(ETH): 99.946321836'. The 'Requestor' panel includes a 'Send a loan request to lender:' section with an 'Enter Loan Amount' input field and a 'Submit Loan Request' button. Below this is a 'Calculate Repay Amount:' section with 'Requestor Address' and 'Enter Loan Amount (ETH)' input fields, and a 'Calculate Repay Amount' button. A line of text 'Calculated repay amount shown here..' follows. The 'Repay Loan:' section contains 'Lender Address', 'Enter Repay Amount (ETH)', and 'Enter Deposit Amount (ETH)' input fields, along with a 'Repay Loan' button. The 'Lender' panel has a 'Check and approve the loan request from borrower:' section. It includes an 'Approve Loan Request:' section with 'Loan Requestor Address' and 'Enter Deposit Amount (ETH)' input fields, and an 'Approve Loan' button. Below that is a 'Reject Loan Request:' section with a 'Loan Requestor Address' input field and a 'Reject Loan' button. Further down is a 'View Credit History of Requestor:' section with a 'Loan Requestor Address' input field and a 'View Credit History' button. A final line of text 'Credit History shown here..' is at the bottom of the Lender panel. At the very bottom of the page, a small credit line reads 'Credit: ABC abc@example.com'.

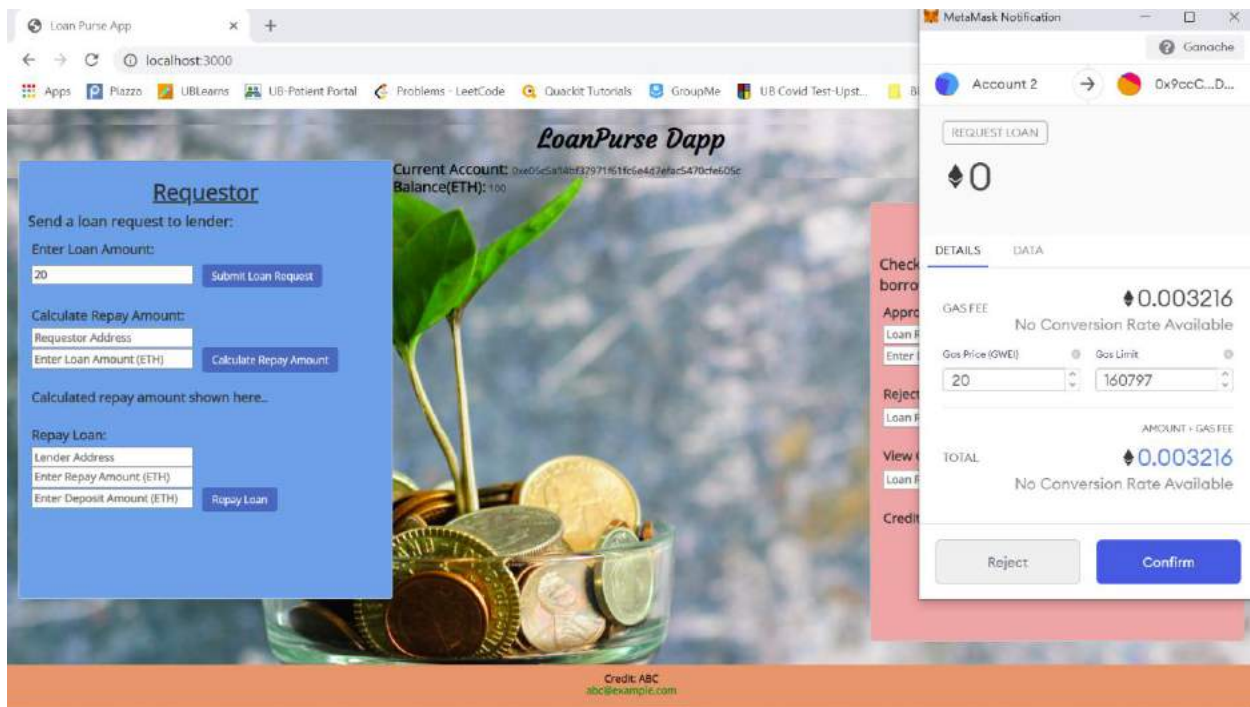
This screenshot shows that all the account has been connected with the web app, and the smart contract on web app was deployed by the 1st account(Smart Contract Owner) in Ganache for which small transaction charges are deducted.



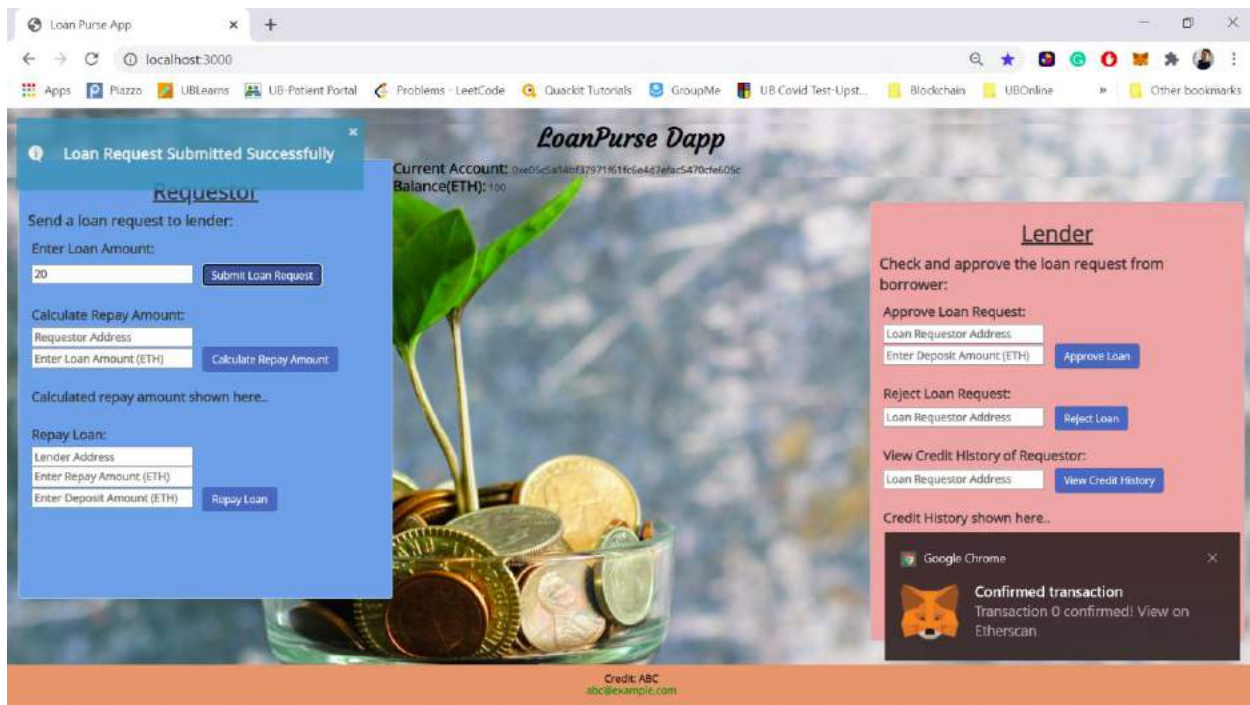
Account 2(Requestor) requests for the loan amount to the Lender by entering the required loan amount in the field and submitting the loan request.



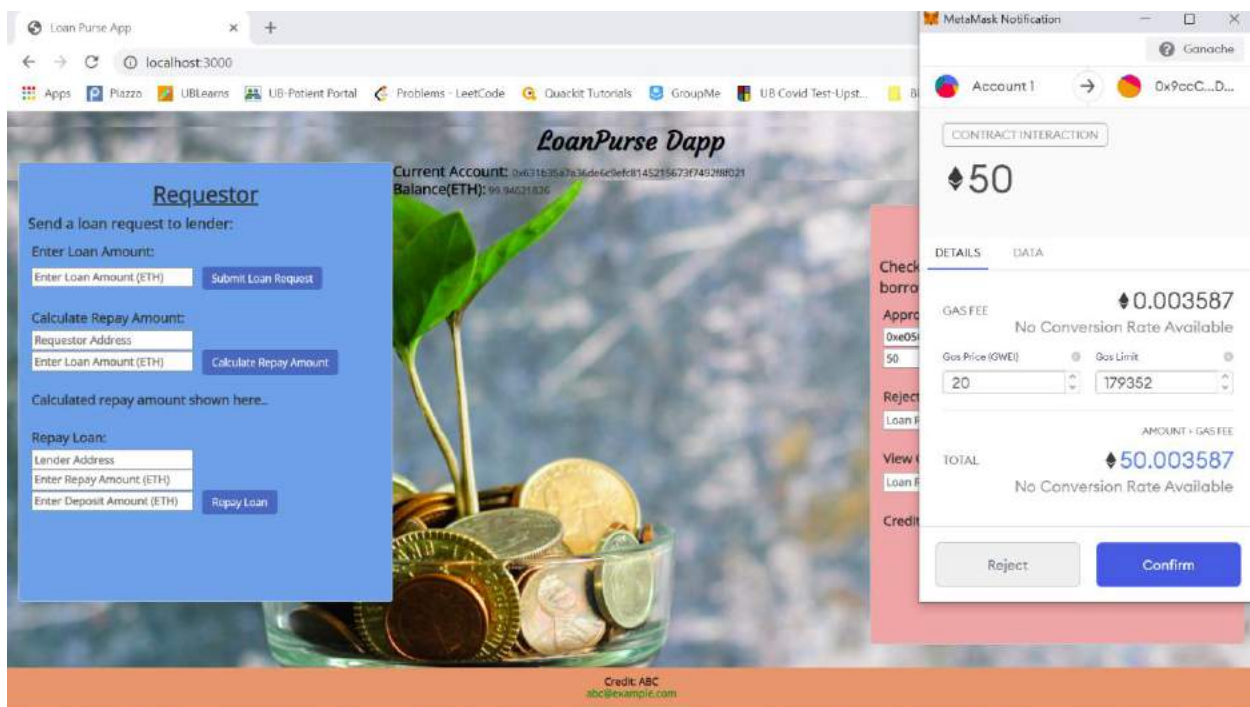
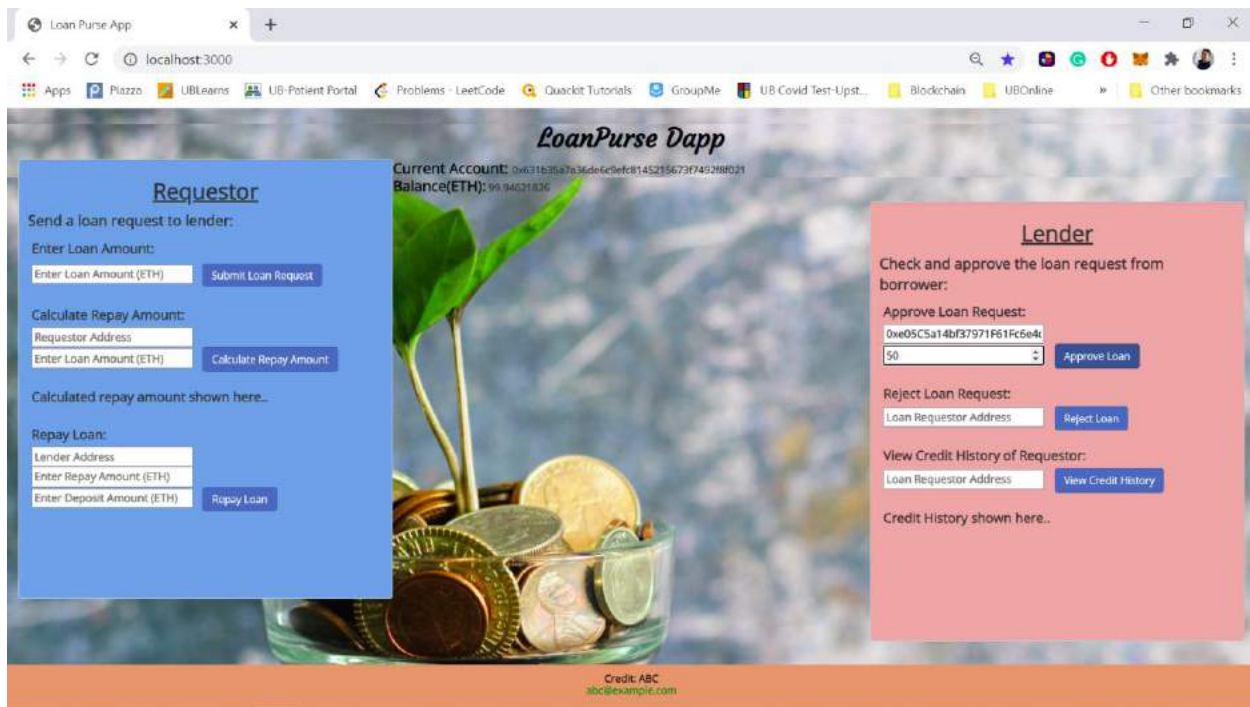


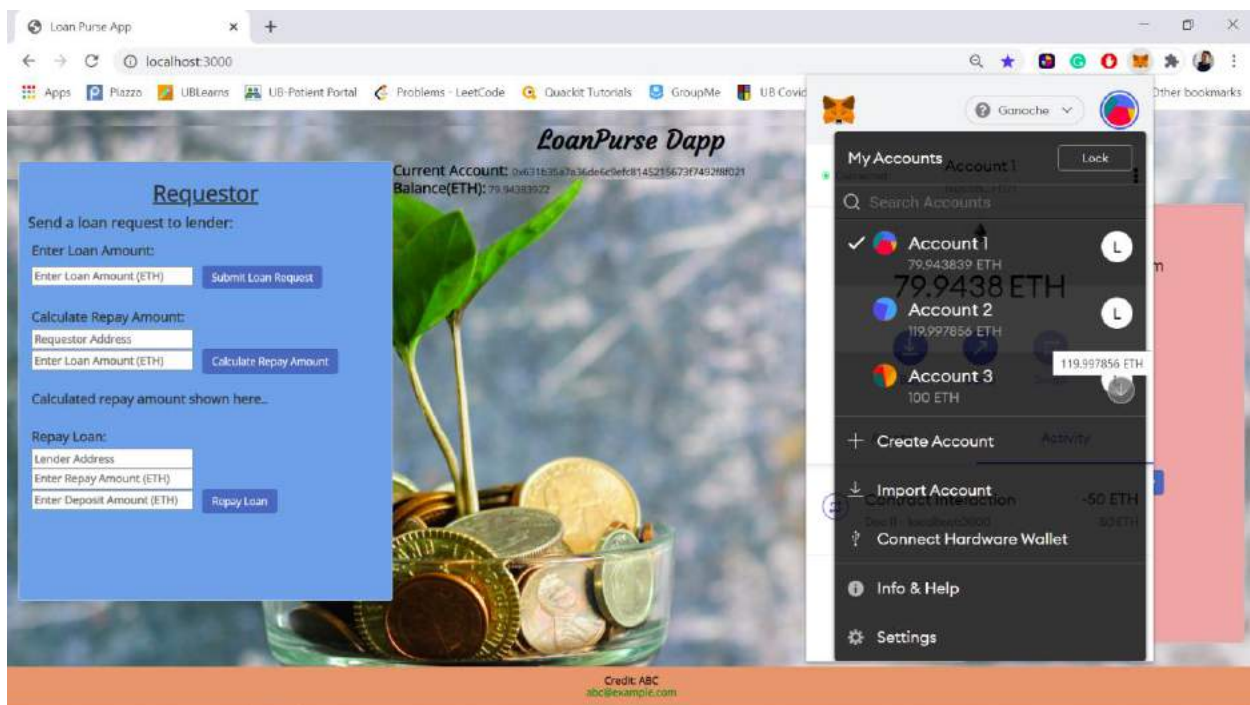
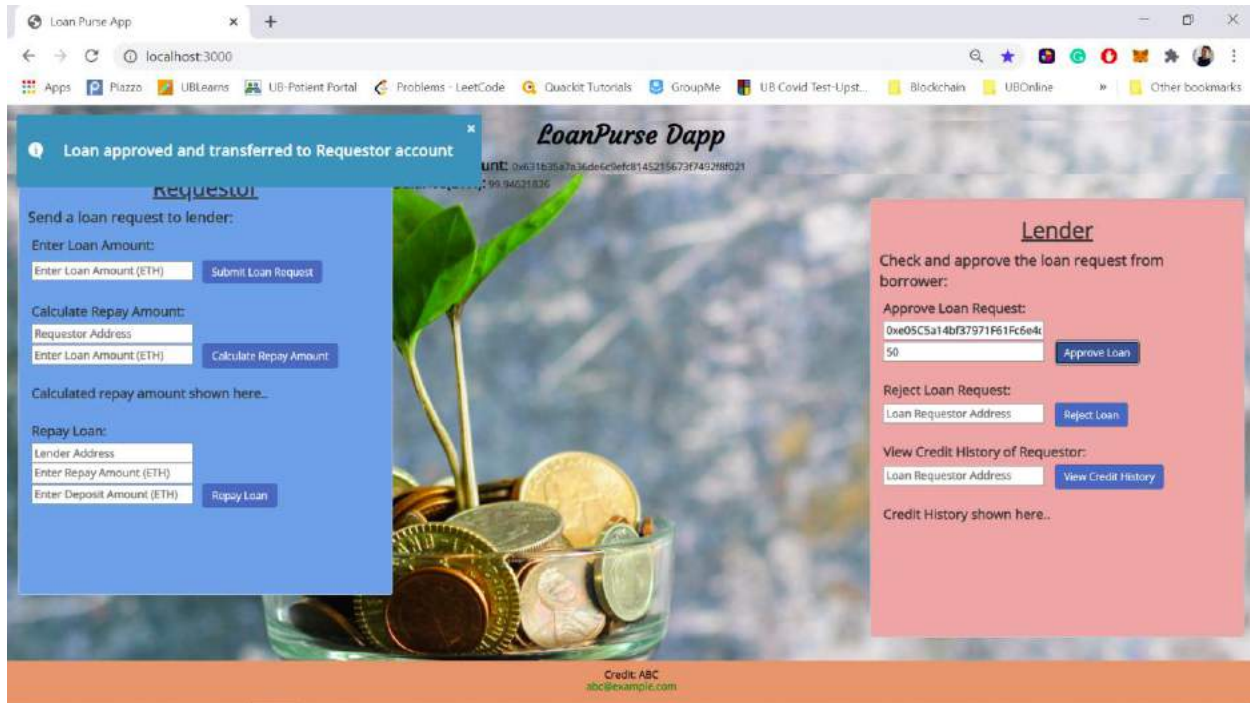


Toast message shows that the loan request was successful.



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.





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

Ganache					
ACCOUNTS BLOCKS TRANSACTIONS CONTRACTS EVENTS LOGS UPDATE AVAILABLE					
CURRENT BLOCK 10 GAS PRICE 20000000000 GAS LIMIT 6721975 HARDFOK 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'/0/account_index		
ADDRESS	BALANCE	TX COUNT	INDEX		
0x631b35a7a36de6c9Efc8145215673f7492F8F021	79.94 ETH	9	0		
ADDRESS	BALANCE	TX COUNT	INDEX		
0xe05C5a14bf37971F61Fc6e4d7EfAC5470cFE605c	120.00 ETH	1	1		
ADDRESS	BALANCE	TX COUNT	INDEX		
0x63c25820418481C728C46f5446332414AD9d243A	100.00 ETH	0	2		
ADDRESS	BALANCE	TX COUNT	INDEX		
0x37bB953186e41385D756Fa57fbaE6E2613592cfC	100.00 ETH	0	3		
ADDRESS	BALANCE	TX COUNT	INDEX		
0x6f18874D52b6ADcA844b969900Ff7d95E664Aaad	100.00 ETH	0	4		
ADDRESS	BALANCE	TX COUNT	INDEX		
0x037242D140d47147ec6E9407672e848719b40dc3	100.00 ETH	0	5		

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

Loan Purse App

localhost:3000

Apps Plazzo UBLeans UB-Patient Portal Pro

localhost:3000 says
Please input the Loan Amount to calculate Repay Amount.

Requestor

Send a loan request to lender:

Enter Loan Amount:

Enter Loan Amount (ETH)

Submit Loan Request

Calculate Repay Amount:

0xe05C5a14bf37971F61Fc6e4d7EfAC5470cFE605c

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

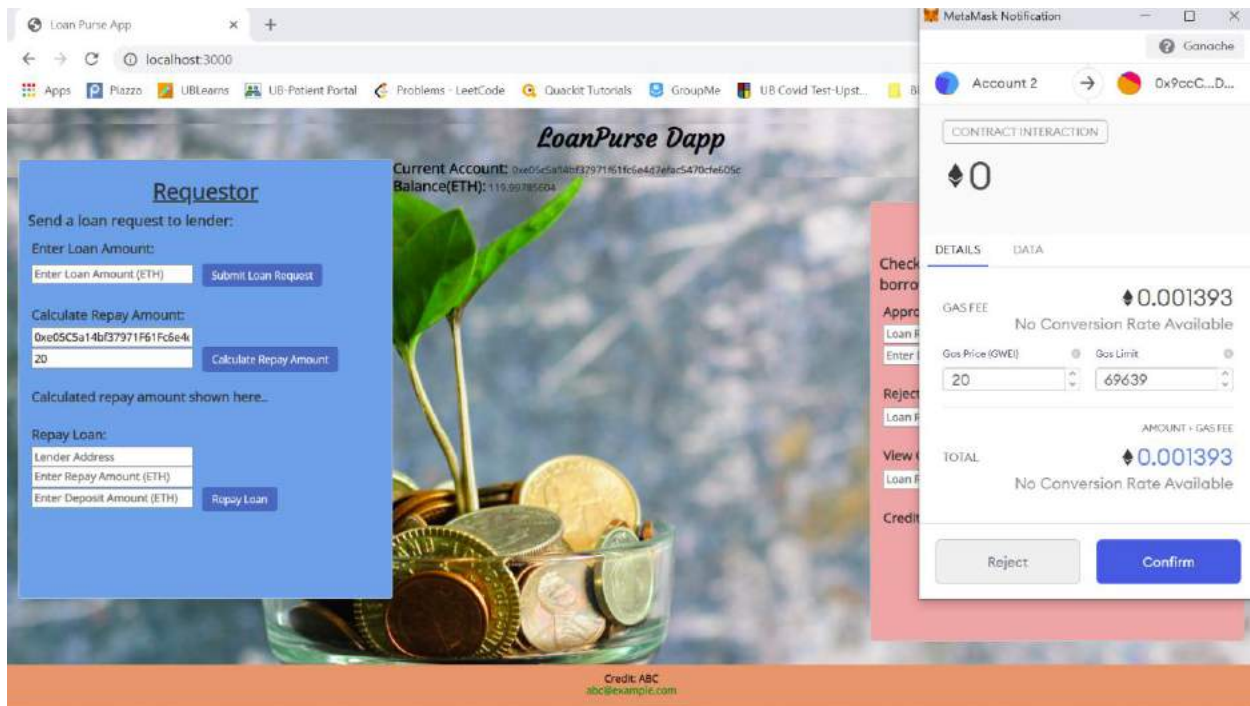
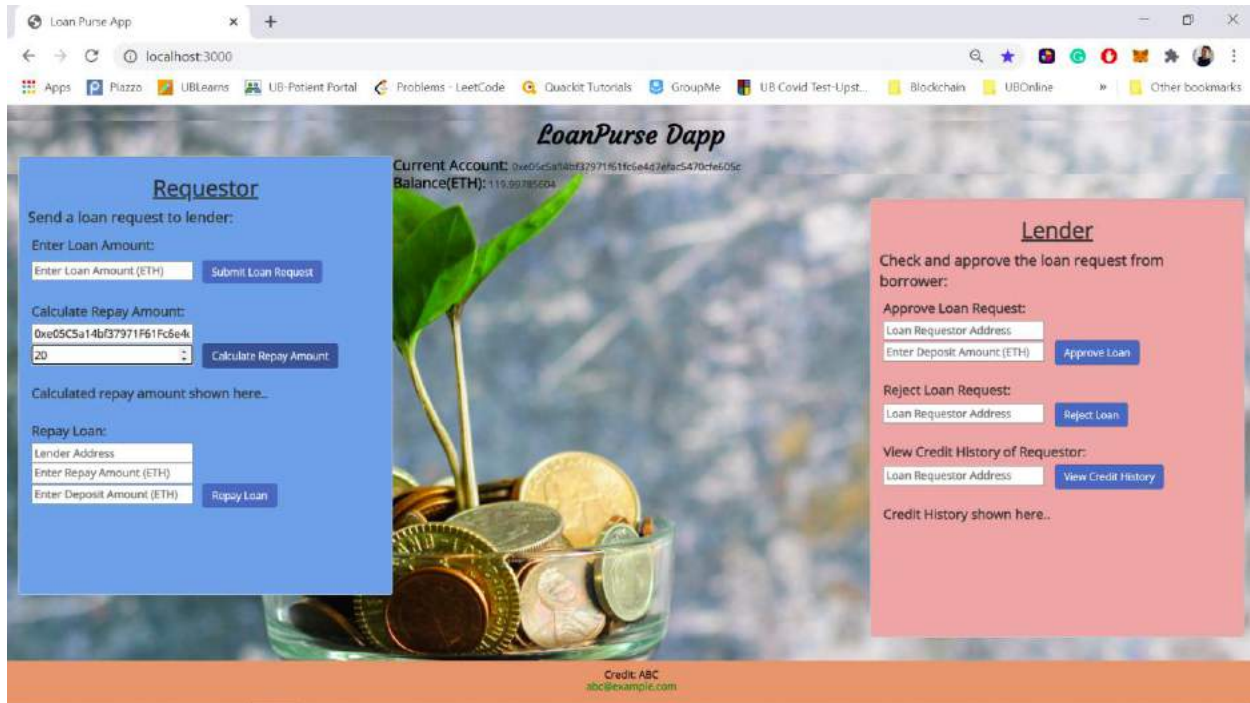
View Credit History of Requestor:

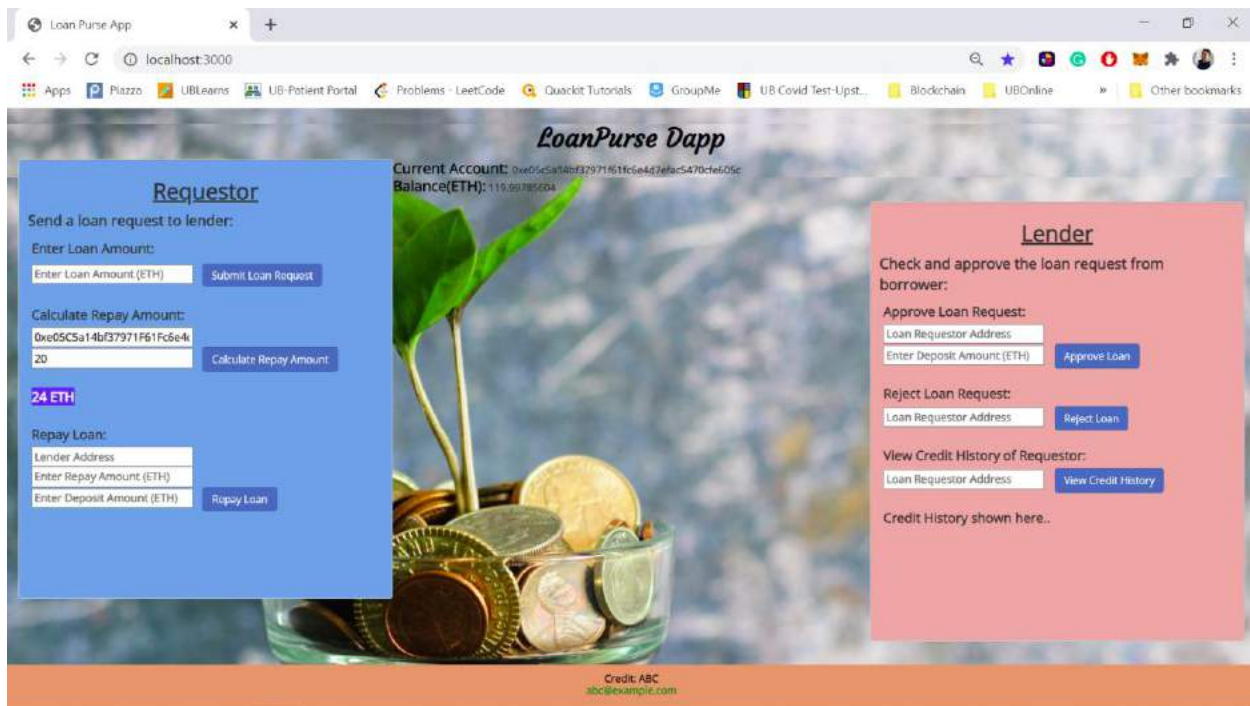
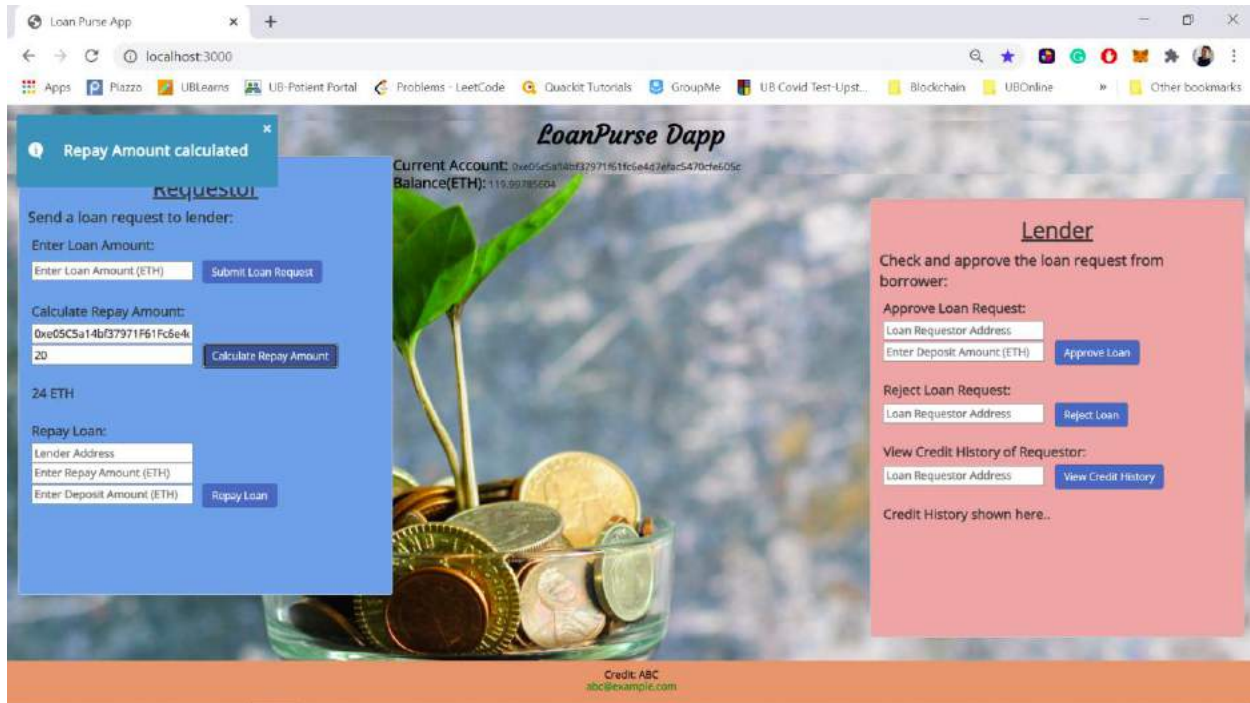
Loan Requestor Address

View Credit History

Credit History shown here..

Credit: ABC
abc@example.com





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

Loan Purse App

localhost:3000

localhost:3000 says
Please input the Lender Address.

Requestor

Send a loan request to lender:

Enter Loan Amount:
Enter Loan Amount (ETH) Submit Loan Request

Calculate Repay Amount:
0xe05C5a14b37971f61Fc6e4k
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

localhost:3000 says
Please input the calculated Repay Amount.

Requestor

Send a loan request to lender:

Enter Loan Amount:
Enter Loan Amount (ETH) Submit Loan Request

Calculate Repay Amount:
0xe05C5a14b37971f61Fc6e4k
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:

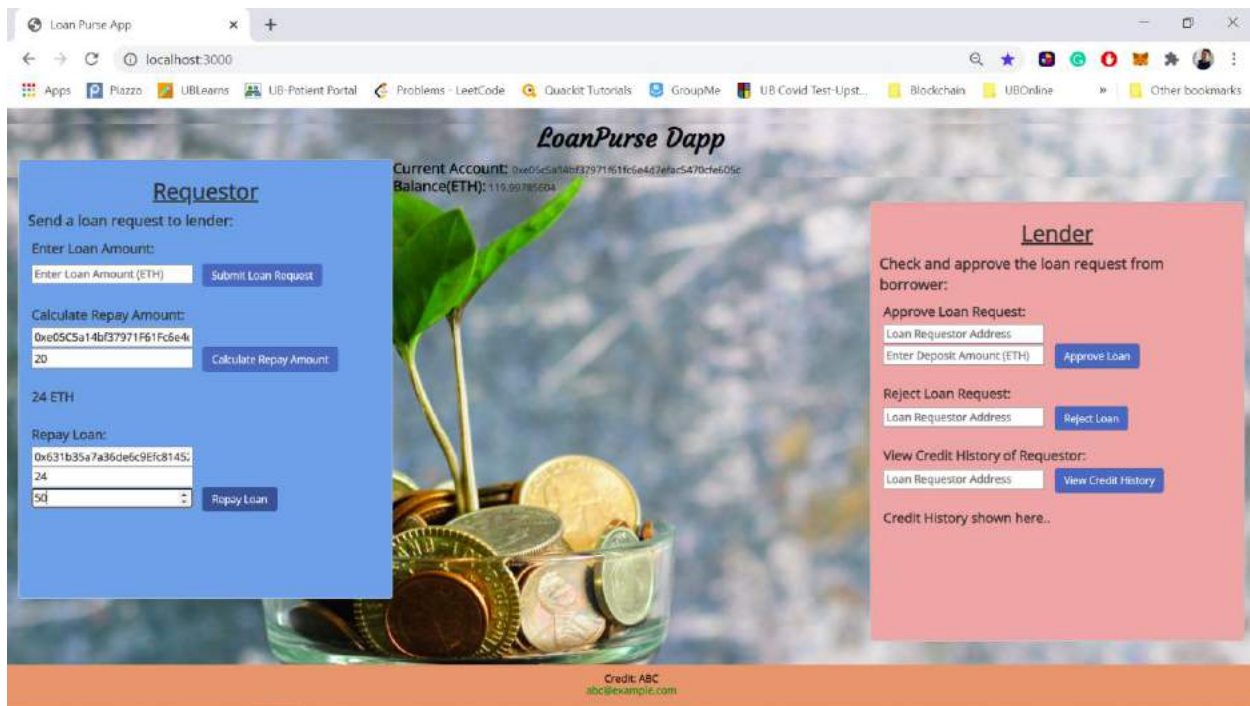
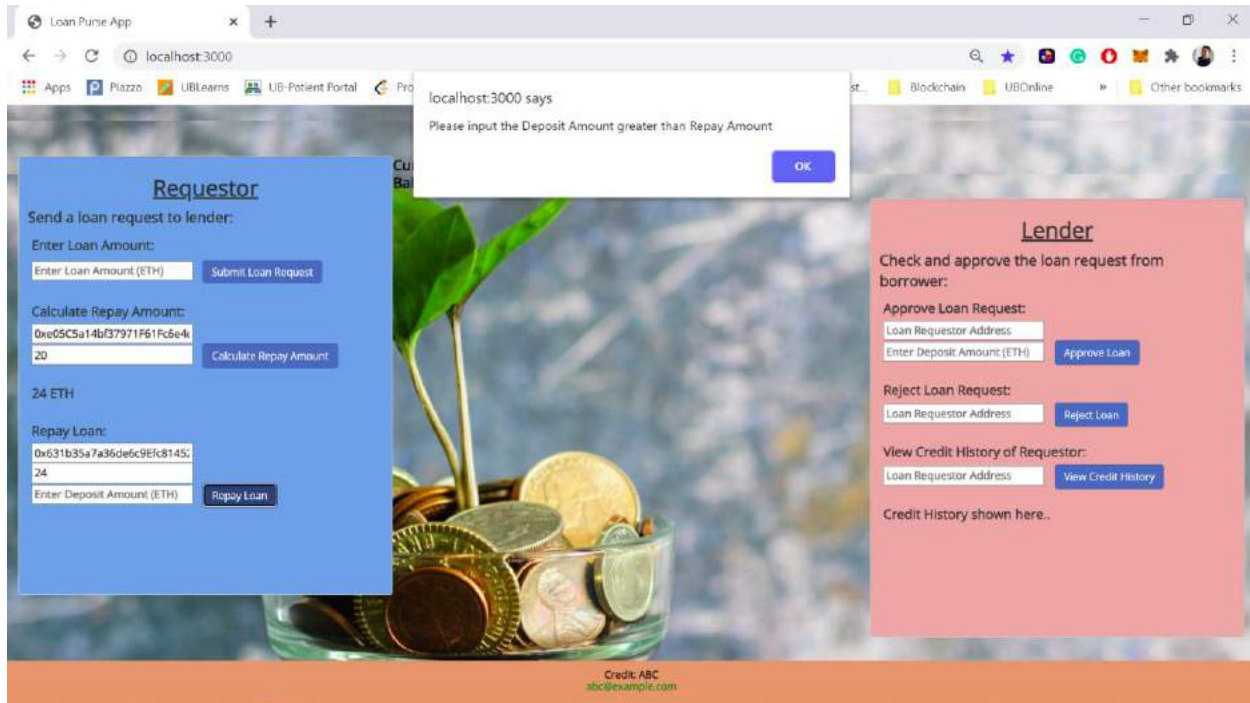
Approve Loan Request:
Loan Requestor Address
Enter Deposit Amount (ETH) Approve Loan

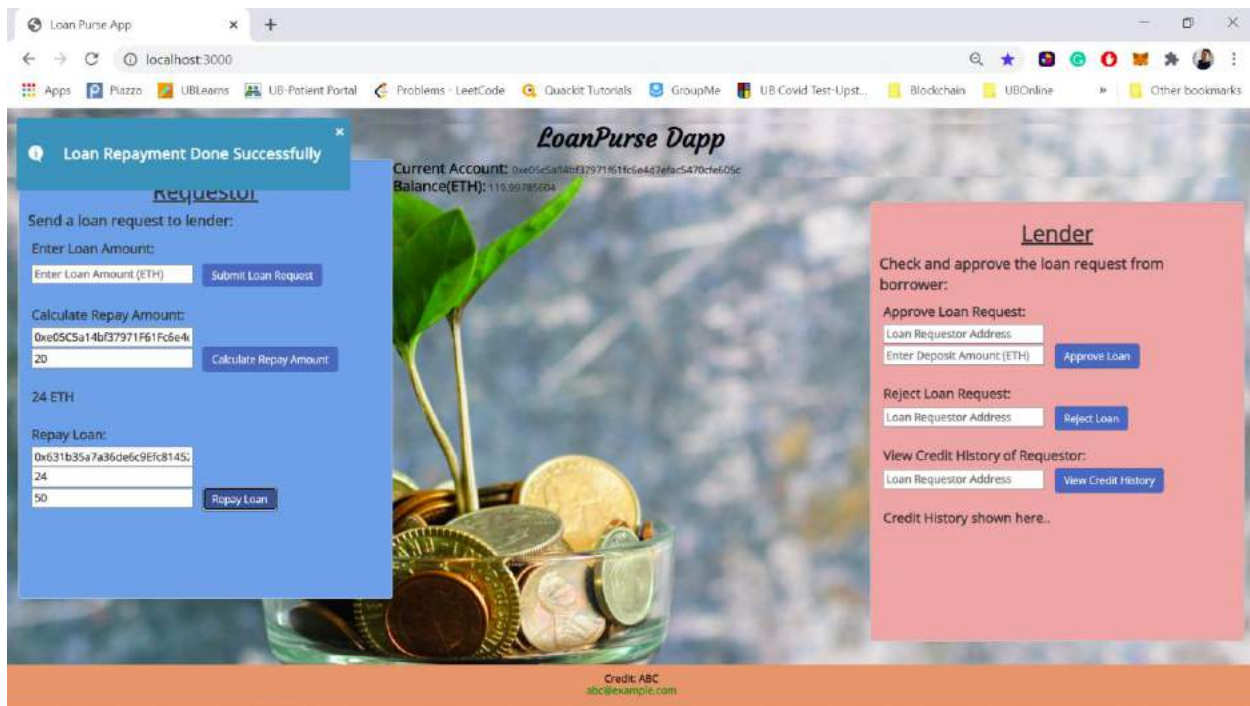
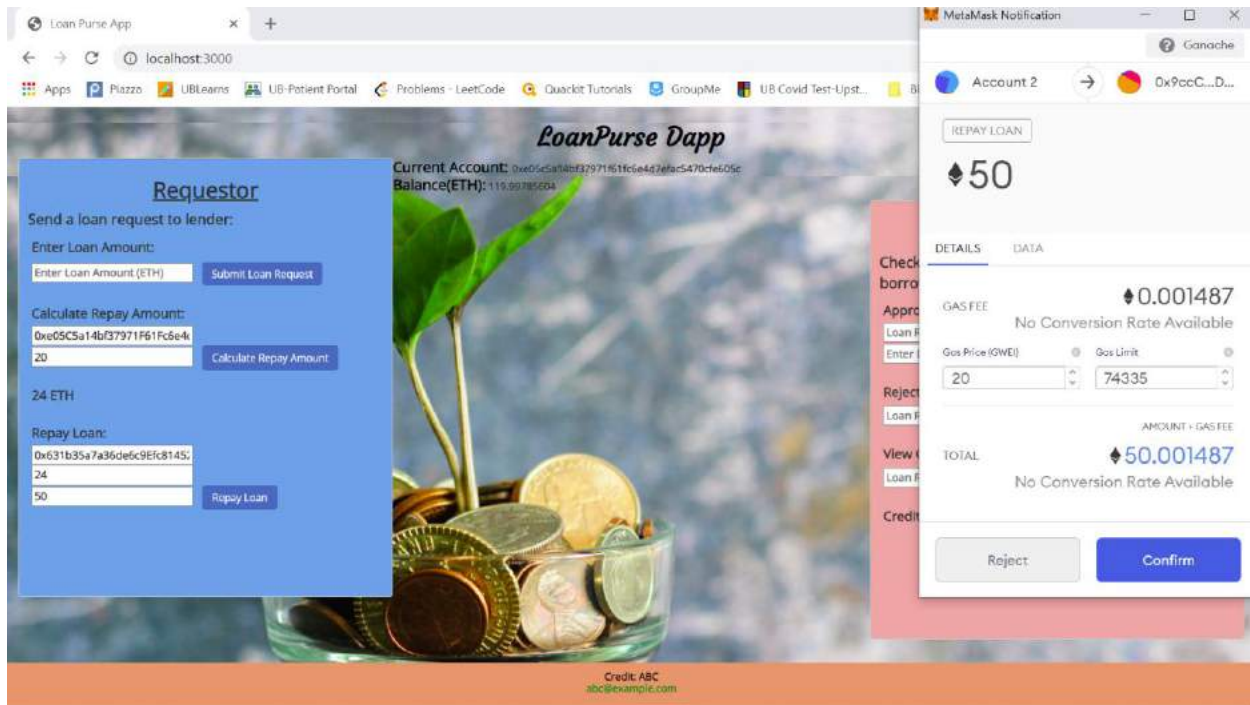
Reject Loan Request:
Loan Requestor Address Reject Loan

View Credit History of Requestor:
Loan Requestor Address View Credit History

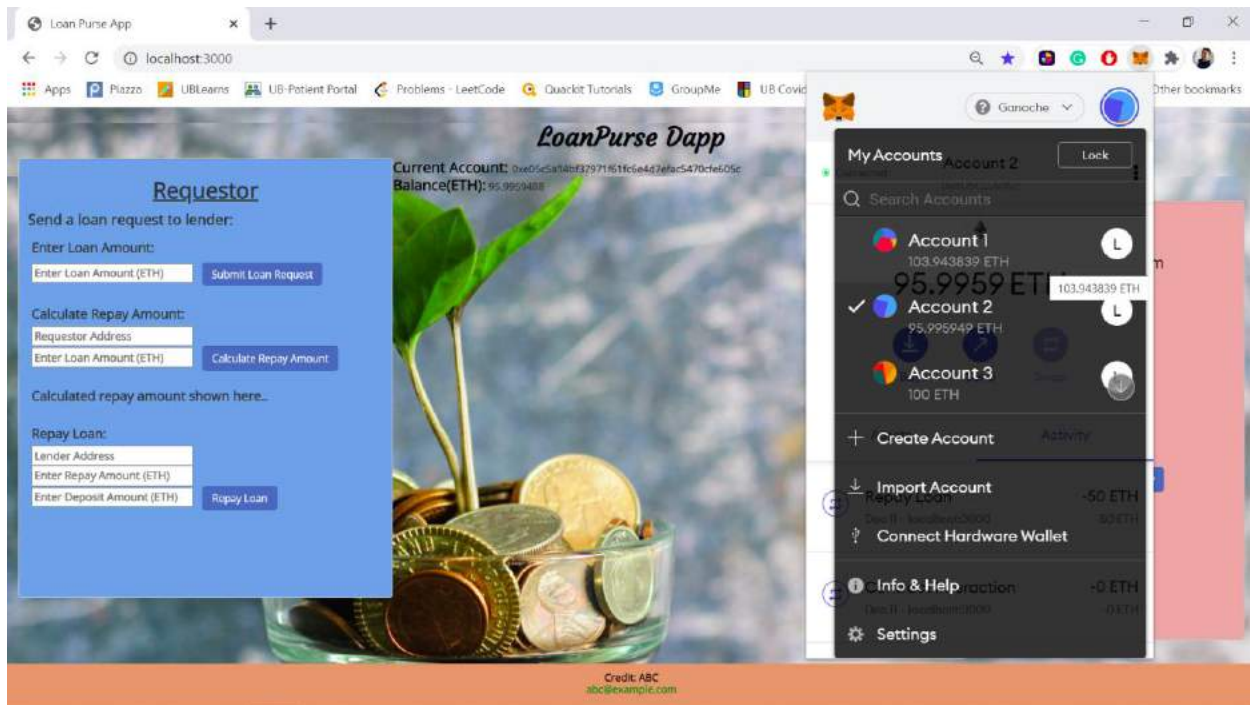
Credit History shown here..

Credit: ABC
abc@example.com





Account 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).



ACCOUNTS

BLOCKS

TRANSACTIONS

CONTRACTS

EVENTS

LOGS

UPDATE AVAILABLE

SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK

12

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

9

INDEX

0

ADDRESS

0xe05C5a14bf37971F61Fc6e4d7EfAC5470cFE605c

BALANCE

96.00 ETH

TX COUNT

3

INDEX

1

ADDRESS

0x63c25820418481C728C46f5446332414AD9d243A

BALANCE

100.00 ETH

TX COUNT

0

INDEX

2

ADDRESS

0x37bB953186e41385D756Fa57fbaE6E2613592cfC

BALANCE

100.00 ETH

TX COUNT

0

INDEX

3

ADDRESS

0x6f18874D52b6ADcA844b96990Ff7d95E664Aaad

BALANCE

100.00 ETH

TX COUNT

0

INDEX

4

ADDRESS

0x037242D140d47147ec6E9407672e848719b40dc3

BALANCE

100.00 ETH

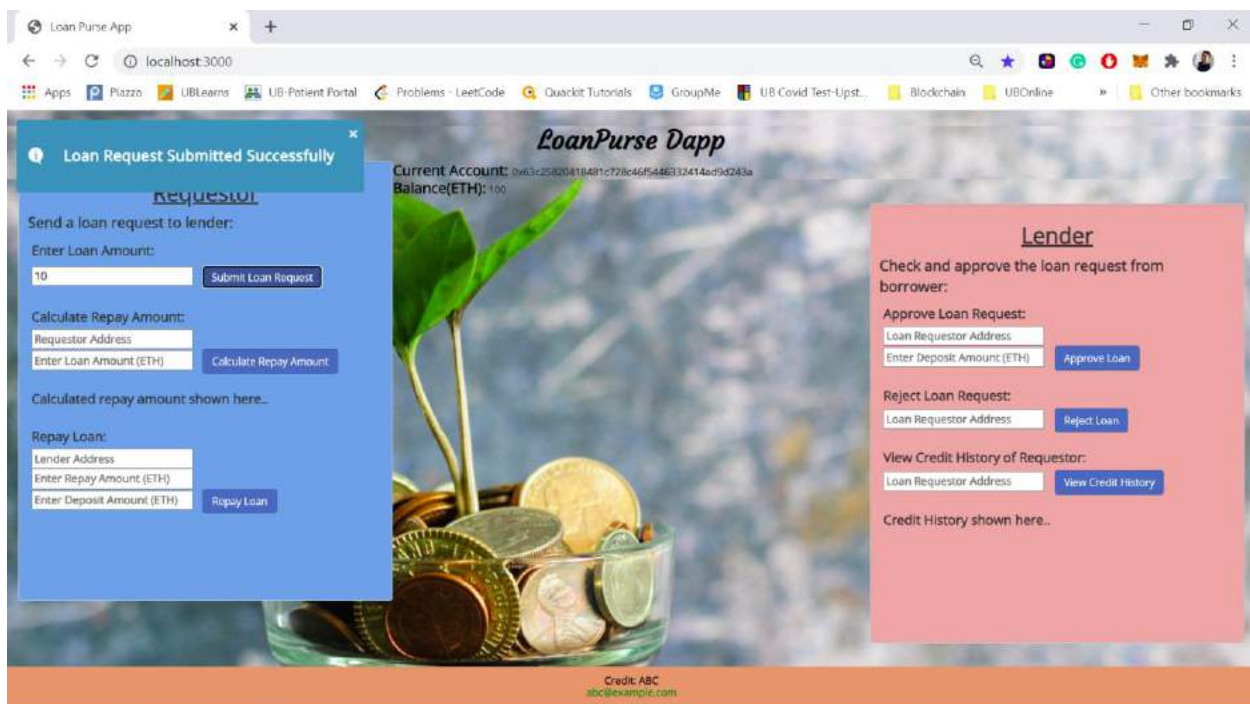
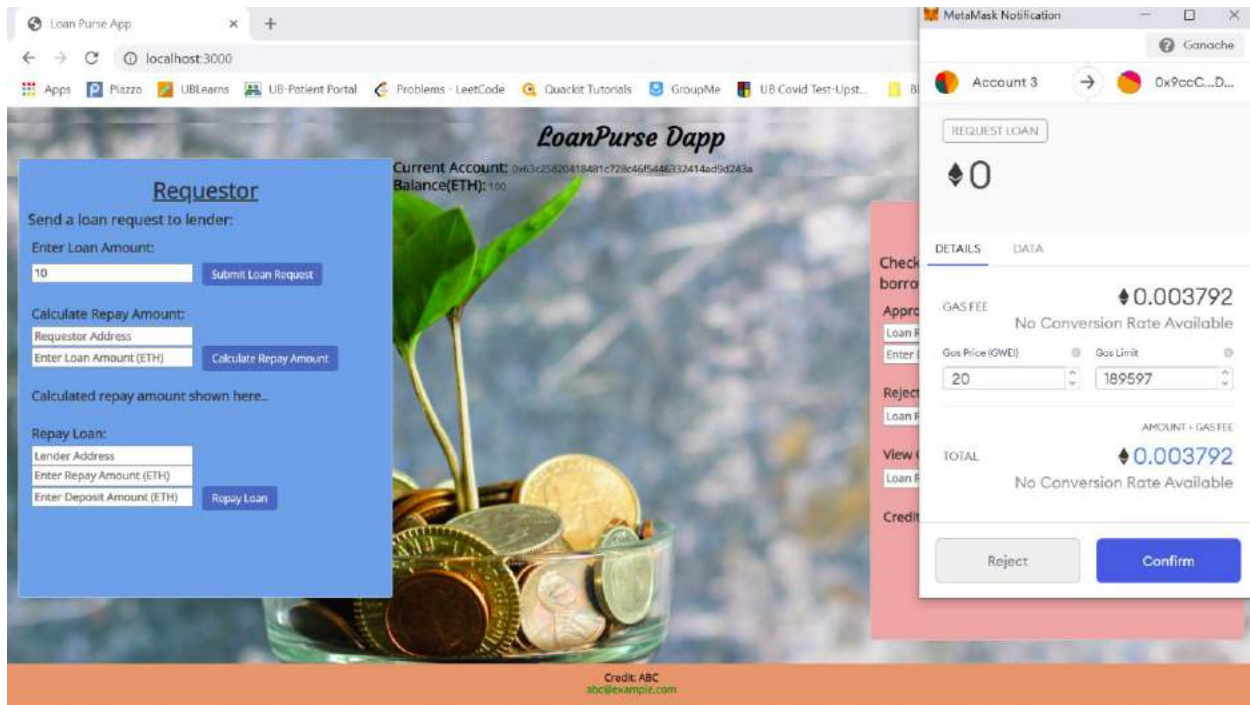
TX COUNT

0

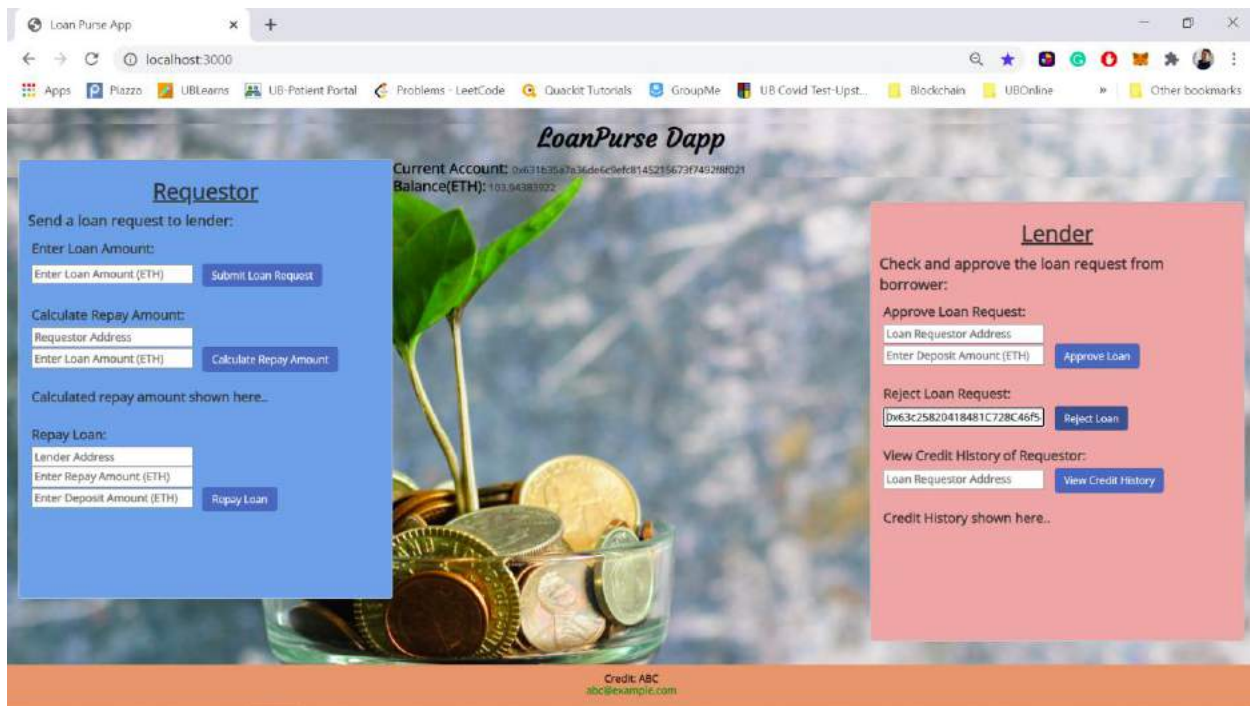
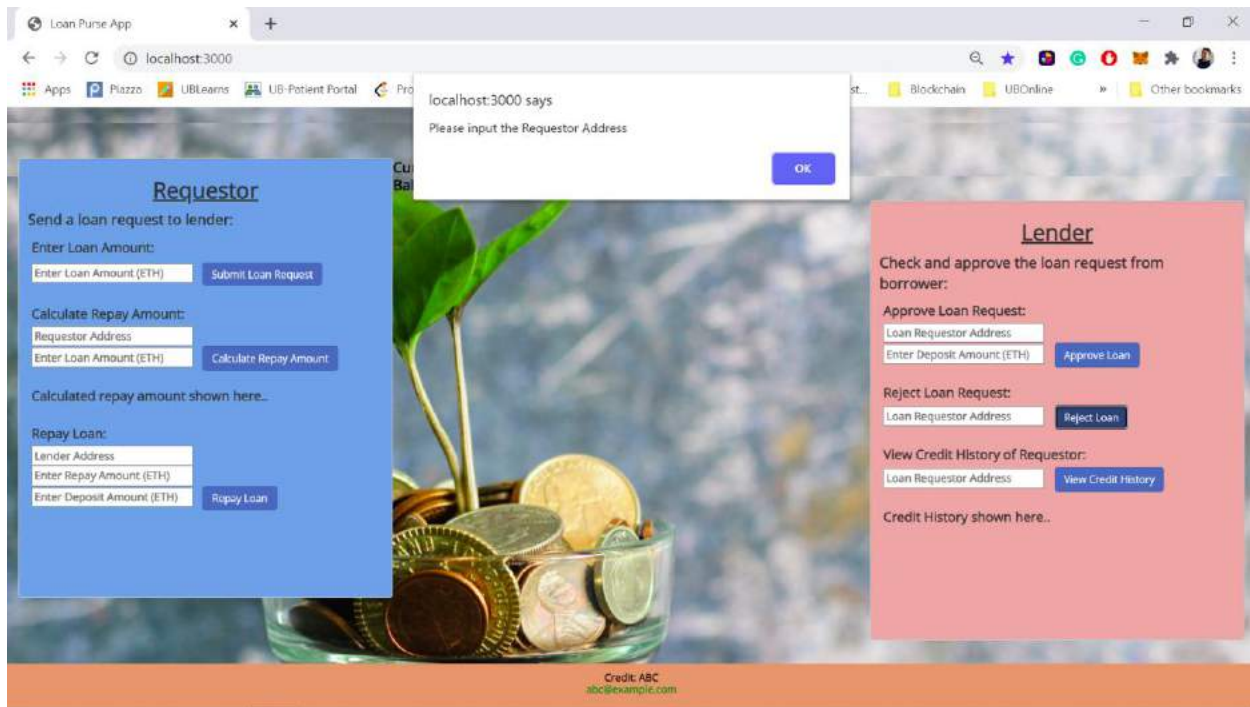
INDEX

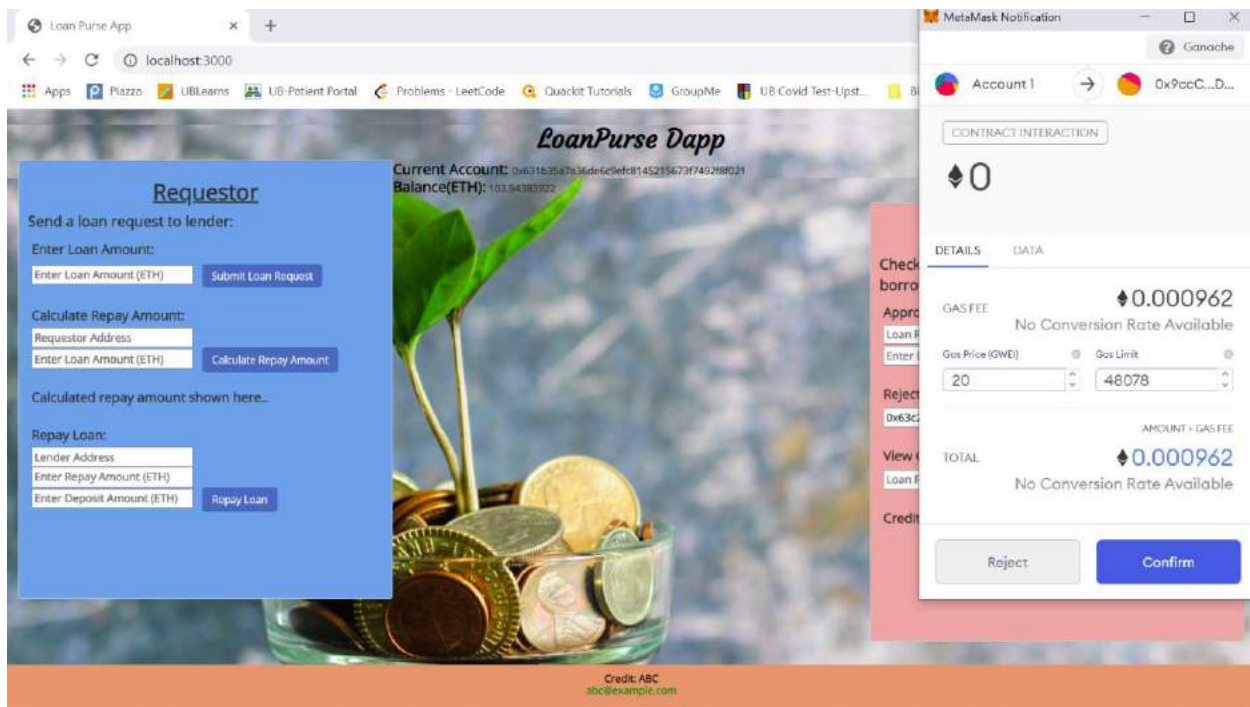
5

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.

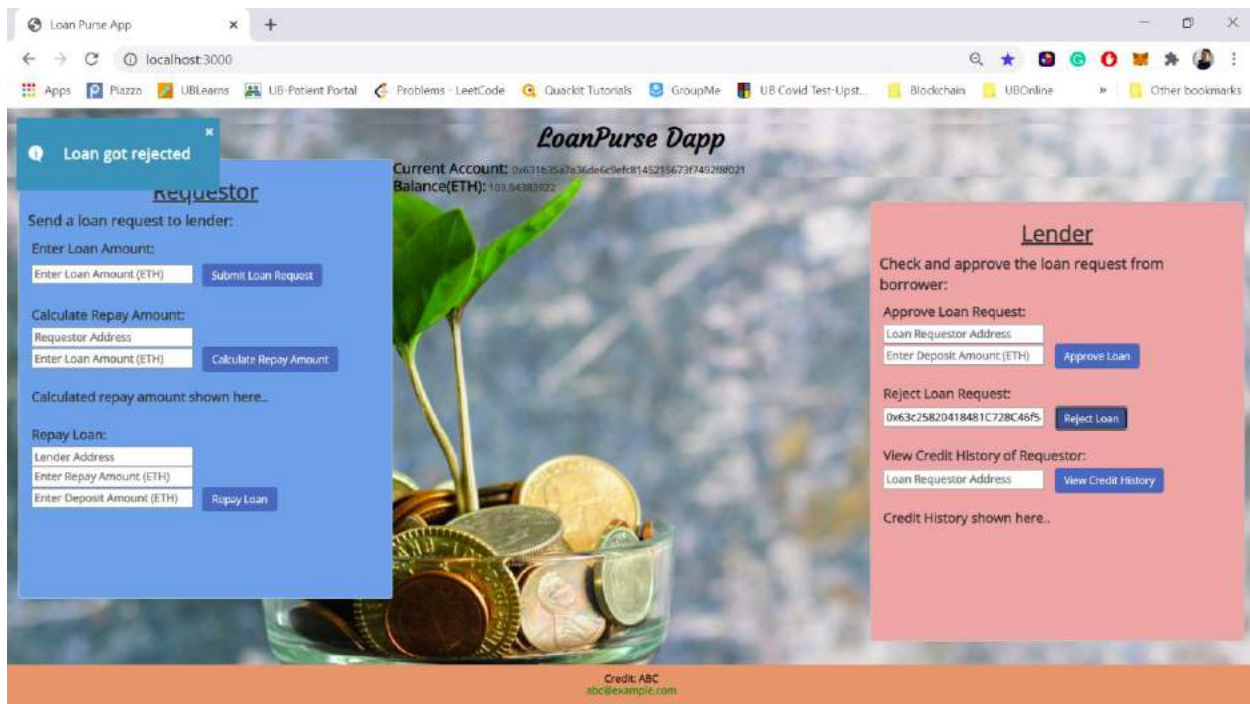


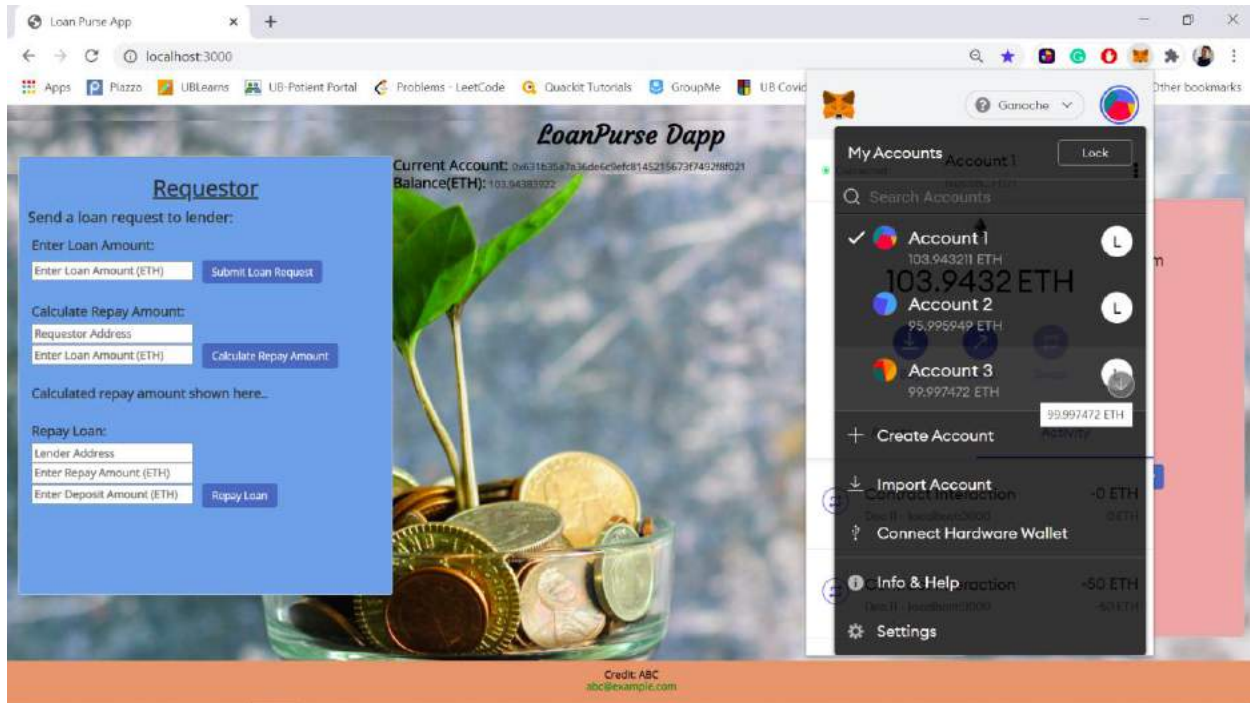
Account 1(Lender) rejects the loan request for the Account 3(Requestor).





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).





Ganache									
ACCOUNTS									
BLOCKS									
TRANSACTIONS									
CONTRACTS									
EVENTS									
LOGS									
UPDATE AVAILABLE									
SEARCH FOR BLOCK NUMBERS OR TX HASHES									
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									
ACCOUNTS									
ADDRESS	BALANCE	TX COUNT	INDEX						
0x631b35a7a36de6c9Efc8145215673f7492f8F021	103.94 ETH	10	0						
ADDRESS	BALANCE	TX COUNT	INDEX						
0xe05C5a14bf37971F61Fc6e4d7EfAC5470cFE605c	96.00 ETH	3	1						
ADDRESS	BALANCE	TX COUNT	INDEX						
0x63c25820418481C728C46f5446332414AD9d243A	100.00 ETH	1	2						
ADDRESS	BALANCE	TX COUNT	INDEX						
0x37bB953186e41385D756Fa57fbaE6E2613592cfC	100.00 ETH	0	3						
ADDRESS	BALANCE	TX COUNT	INDEX						
0x6f18874D52b6ADcA844b969900Ff7d95E664Aaad	100.00 ETH	0	4						
ADDRESS	BALANCE	TX COUNT	INDEX						
0x037242D140d47147ec6E9407672e848719b40dc3	100.00 ETH	0	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

MNEMONIC
valid token avoid talent negative ship vessel rose add taxi advance shoe

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0x631b35a7a36de6c9Efc8145215673f7492F8F021	103.94 ETH	10	0
0xe05C5a14bf37971F61Fc6e4d7EfAC5470cFE605c	96.00 ETH	3	1
0x63c25820418481C728C46f5446332414AD9d243A	100.00 ETH	1	2
0x37bB953186e41385D756Fa57fbaE6E2613592cfC	100.00 ETH	0	3
0x6f18874D52b6ADcA844b969900Ff7d95E664Aaad	100.00 ETH	0	4
0x037242D140d47147ec6E9407672e848719b40dc3	100.00 ETH	0	5

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

BLOCK	MINED ON	GAS USED	TRANSACTION
14	2020-12-11 01:16:55	31427	1 TRANSACTION
13	2020-12-11 01:15:11	126398	1 TRANSACTION
12	2020-12-11 01:06:37	48936	1 TRANSACTION
11	2020-12-11 01:02:49	46426	1 TRANSACTION
10	2020-12-11 00:59:52	118952	1 TRANSACTION
9	2020-12-11 00:57:04	107198	1 TRANSACTION
8	2020-12-11 00:50:06	27338	1 TRANSACTION
7	2020-12-11 00:50:05	1088004	1 TRANSACTION
6	2020-12-11 00:50:05	42338	1 TRANSACTION

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 0xc9294efe5eeb547af97f138ab7aaa35ee64f9579a9132663dfd08de625aad288		CONTRACT CALL	
FROM ADDRESS 0x631b35a7a36de6c9Efc8145215673f7492F8F021	TO CONTRACT ADDRESS 0x9cc12A76D943f1696F83Fe244099f2647faD281	GAS USED 31427	VALUE 0

TX HASH 0x9d86fe0c74dff4fb6f4781149036bdc6b66b22956fa7ee7a594404aad270efda7		CONTRACT CALL	
FROM ADDRESS 0x63c25820418481C728C46f5446332414AD9d243A	TO CONTRACT ADDRESS 0x9cc12A76D943f1696F83Fe244099f2647faD281	GAS USED 126398	VALUE 0

TX HASH 0x7d9c1b97e3cd1acc63a5415844995ff8ee01608b83391d7ea59dc613f729cfc5		CONTRACT CALL	
FROM ADDRESS 0xe05C5a14bf37971F61Fc6e4d7EFAC5470cFE605c	TO CONTRACT ADDRESS 0x9cc12A76D943f1696F83Fe244099f2647faD281	GAS USED 48936	VALUE 5000000000000000000

TX HASH 0x5d30dfae91fee9d4028ffc9cc0a09216a3098fc675d53e780e7937c6cee42e8c		CONTRACT CALL	
FROM ADDRESS 0xe05C5a14bf37971F61Fc6e4d7EFAC5470cFE605c	TO CONTRACT ADDRESS 0x9cc12A76D943f1696F83Fe244099f2647faD281	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 0xecdfe0cfc666631d3348341ae38d16d417ed6fe052d2f2519491d5f498ac572		CONTRACT CALL	
FROM ADDRESS 0xe05C5a14bf37971F61Fc6e4d7EFAC5470cFE605c	TO CONTRACT ADDRESS 0x9cc12A76D943f1696F83Fe244099f2647faD281	GAS USED 107198	VALUE 0

TX HASH 0x2566707ce88541e764c198e84dfc74abe2e472d39b81953eda0775ca1dd697f7		CONTRACT CALL	
FROM ADDRESS 0x631b35a7a36de6c9Efc8145215673f7492F8F021	TO CONTRACT ADDRESS 0x6D43A45927D53786d9Df245Fa677f0Bd064c8d6A	GAS USED 27338	VALUE 0

TX HASH 0x52fa74f456c6d0698b6733517790fb979be56bc53836162d209c10ebda787cc3		CONTRACT CREATION	
FROM ADDRESS 0x631b35a7a36de6c9Efc8145215673f7492F8F021	CREATED CONTRACT ADDRESS 0x9cc12A76D943f1696F83Fe244099f2647faD281	GAS USED 1088094	VALUE 0

TX HASH 0x5aac1beff49cf9bdd4f07f8fa59fae1999230cb03eff41f808fab9e7afa3c849		CONTRACT CALL	
FROM ADDRESS 0x631b35a7a36de6c9Efc8145215673f7492F8F021	TO CONTRACT ADDRESS 0x6D43A45927D53786d9Df245Fa677f0Bd064c8d6A	GAS USED 42338	VALUE 0

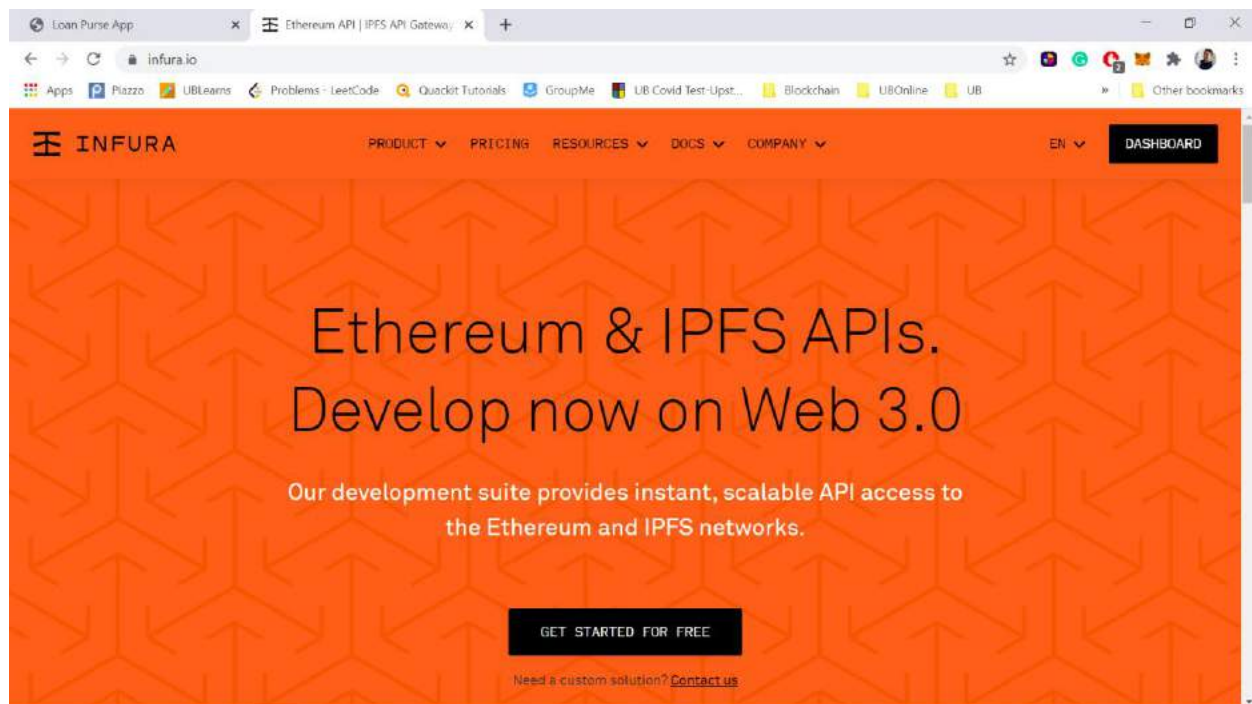
Phase 6

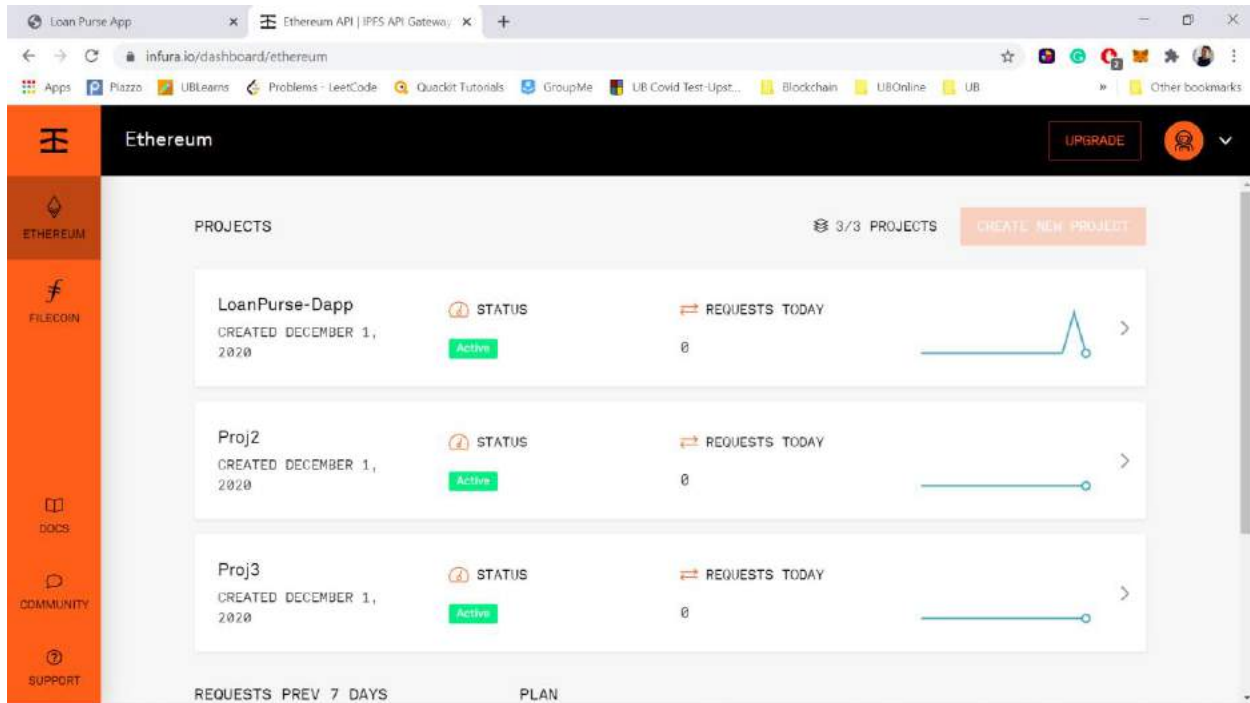
Dapp Deployment on Infura

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.

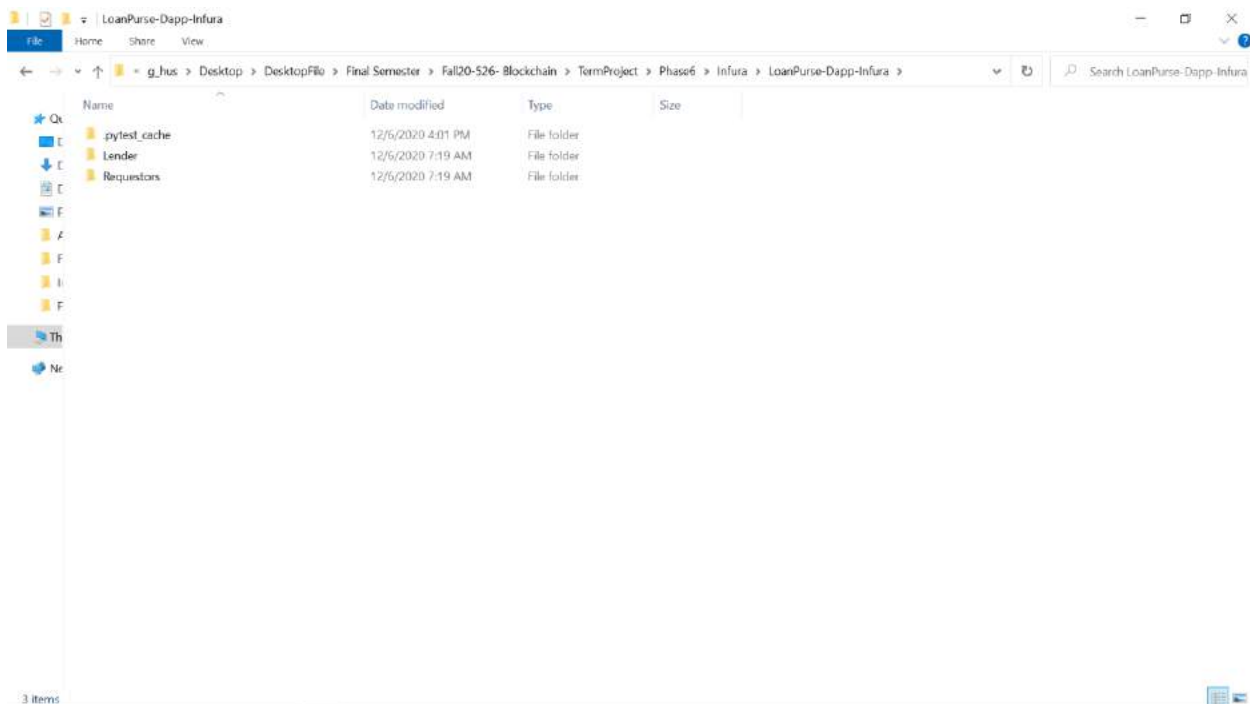
I have created a project called (LoanPurse -Dapp) on the Infura to host my Dapp as shown below.

Infura details:

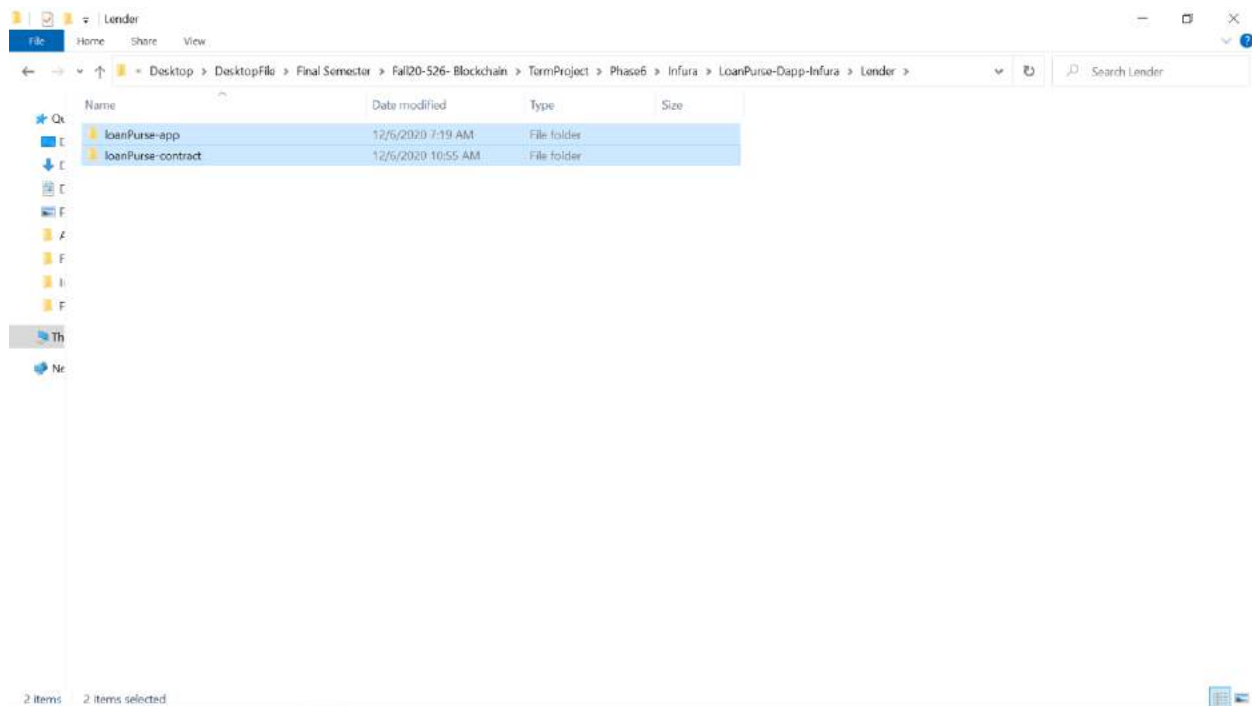




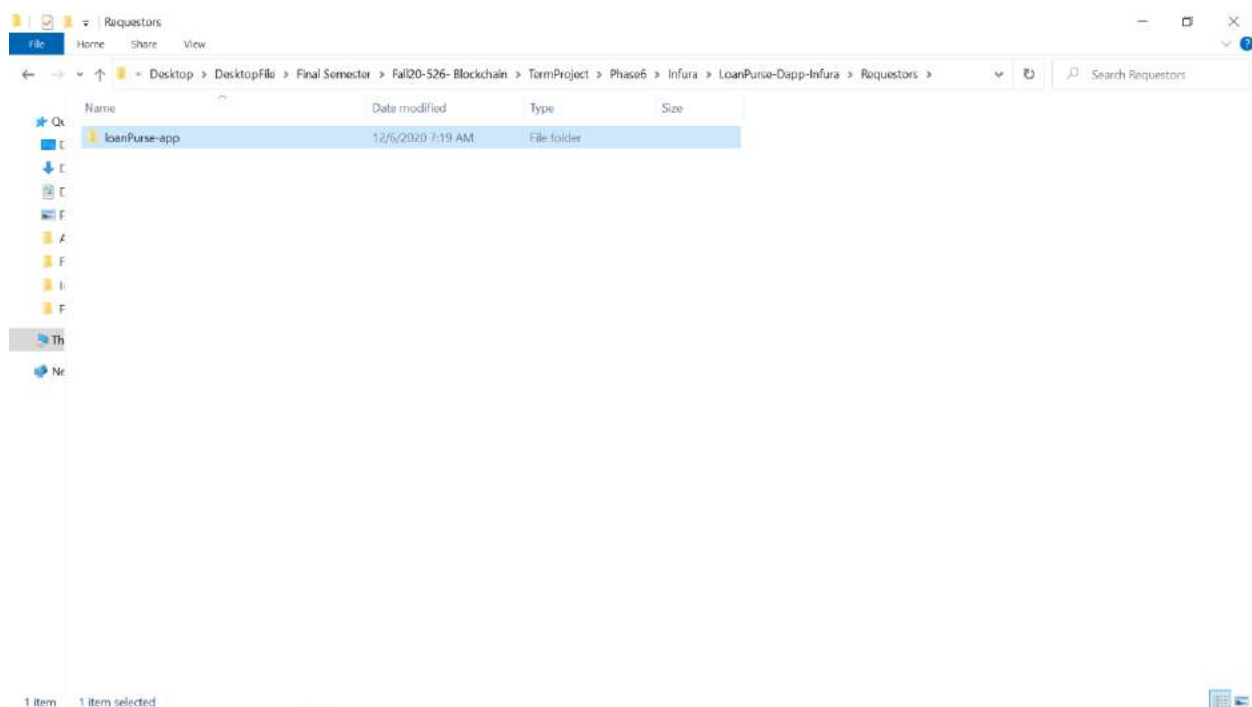
Dapp folder structure:



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');
const lender = '0x...';

module.exports = {
  // see <http://truffleframework.com/docs/advanced/configuration>
  // to customize your Truffle configuration!
  networks: {
    ropsten: {
      provider: () => new HDWalletProvider(
        // [Redacted]
      ),
      network_id: 3,
      gas: 5000000,
      skipDryRun: false
    },
    development: {
      host: 'localhost',
      port: 7545,
      network_id: '*' // "5777"
    }
  },
  compilers: {
    solc: {
      version: '0.5.0'
    }
  }
}
```

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": ""
}
```

Final Semester > Fall20-526- Blockchain > TermProject > Phase6 > Infura > LoanPurse-Dapp-Infura > Lender > loanPurse-contract > contracts

Name	Date modified	Type	Size
LoanPurse.sol	12/6/2020 10:17 PM	SOL File	6 KB
Migrations.sol	10/26/1985 4:15 AM	SOL File	1 KB

Deployment of smart contract using truffle with Ropsten account:

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


Smart Contract Address:

```
Select Command Prompt

> gas used:      198011 (0x2e63b)
> gas price:     20 gwei
> value sent:    0 ETH
> total cost:    0.00388022 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:    0.00388022 ETH

2_deploy_contracts.js
=====
> Blocks: 1      Seconds: 28
> contract address: 0x2f0Eb24ab1850E0804Cd03C7EC8215B32d62dF72
> block number:    9212172
> block timestamp: 1607316294ab89aaf7a32c7bf641e84886238ace961cb34f425943dbb55bd1fba
> account:         0x3cD3AC29CA765151e87dA6eb8aA2877C35a9DE1F
> balance:         83.736405488840432366
> gas used:        1075907 (0x106ac3)
> gas price:       20 gwei
> value sent:      0 ETH
> total cost:      0.02151814 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:      0.02151814 ETH

Summary
=====
> Total deployments: 2
> Final cost:        0.02531836 ETH

C:\Users\g_hus\Desktop\DesktopFile\Final Semester\Fall120-526- Blockchain\TermProject\Phase6\Infura\LoanPurse-Dapp-Infura\Lender\loanPurse-contract>
```

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

module.exports = {
  // see <http://truffleframework.com/docs/advanced/configuration>
  // to customize your Truffle configuration!
  networks: {
    ropsten: {
      provider: () => new HDWalletProvider('...', 'https://ropsten.infura.io/v3/57777'),
      network_id: 3,
      gas: 800000,
      skipDryRun: false
    },
    development: {
      host: "localhost",
      port: 7545,
      network_id: "*" //("5777"),
    },
  },
  compilers: {
    solc: {
      version: "0.5.0"
    }
  }
}
```

Infura URL where Lender has deployed the Dapp:

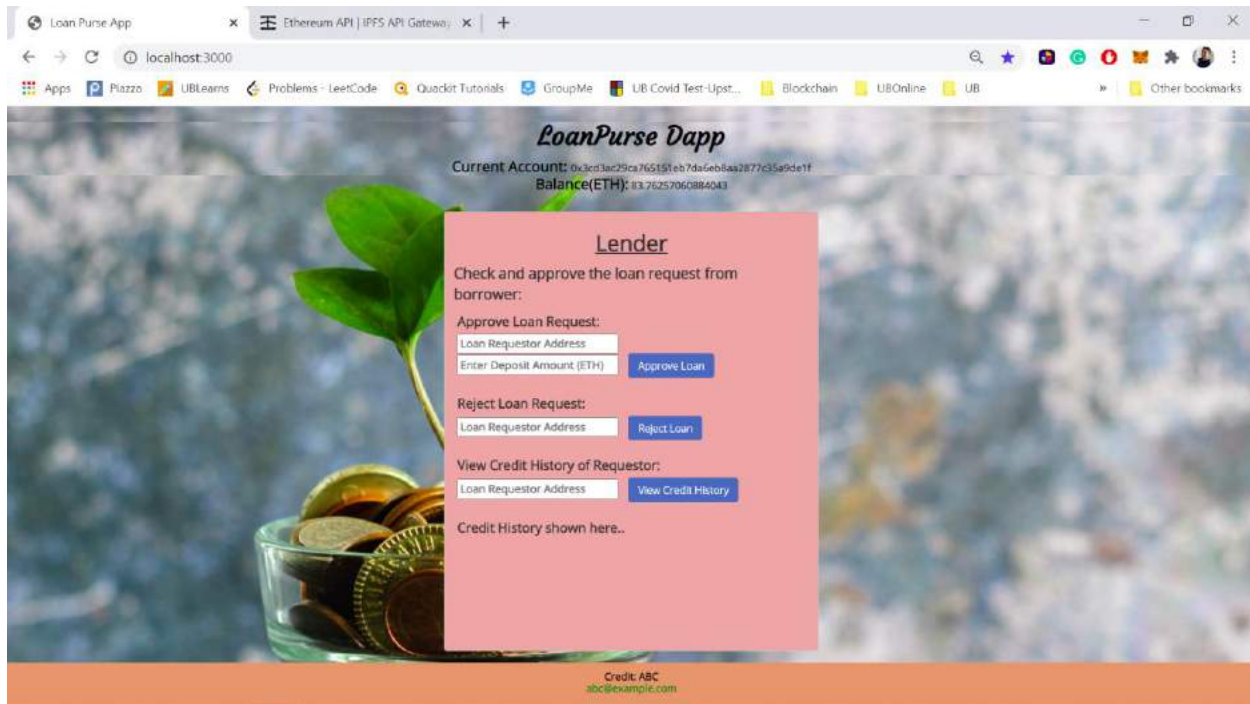
```
Lender > loanPurse-app > src > js > JS app.js > ...
1  App = {
2    web3Provider: null,
3    contracts: {},
4    names: new Array(),
5    // url: 'http://127.0.0.1:7545',
6    contractOwner: null,
7    currentAccount: null,
8    myRepayAmount: 0,
9    loanCount: 0,
10   loanAmount: 0,
11   // network_id: 5777,
12   url: 'https://ropsten.infura.io/v3/...',
13
14
```

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

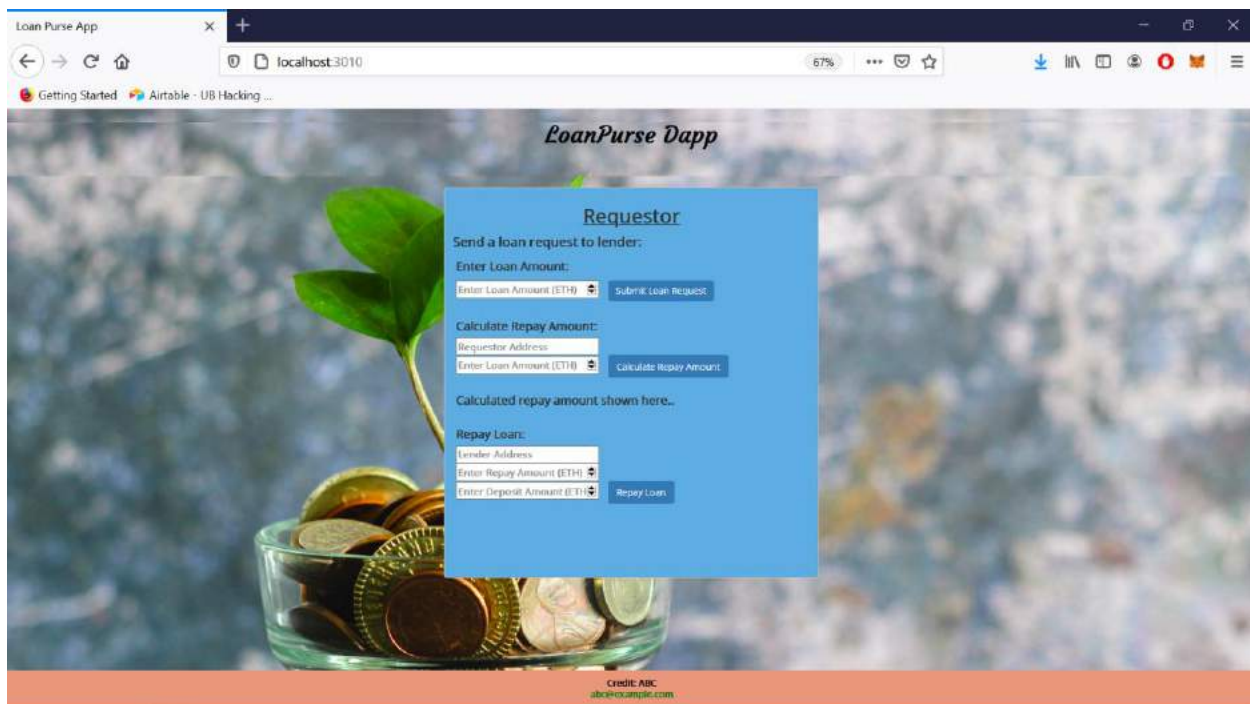
```
JS app.js  X  <> index.html
Requestors > loanPurse-app > src > js > JS app.js > address
1  App = {
2    web3Provider: null,
3    contracts: {},
4    names: new Array(),
5    // url: 'http://127.0.0.1:7545',
6    contractOwner: null,
7    currentAccount1: null,
8    myRepayAmount: 0,
9    loanCount: 0,
10   loanAmount: 0,
11   // network_id: 5777,
12   address: '0x2f0Eb24ab1B58E0804Cd03C7EC8215B32d82df72',
13
14
15   init: function() {
```

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.

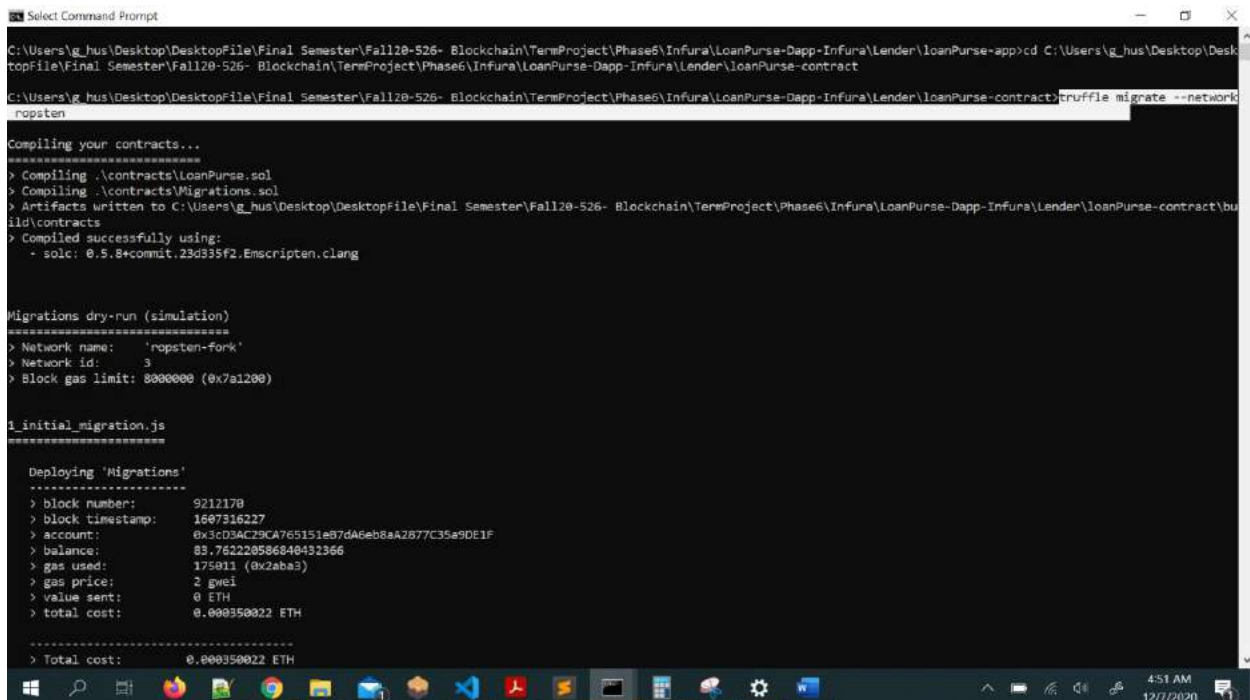
```
npm install
truffle migrate --network Ropsten
```

- 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
C:\Users\g_hus\Desktop\DesktopFile\Final Semester\Fall120-526- Blockchain\TermProject\Phase6\Infura\LoanPurse-Dapp-Infura\Lender\loanPurse-app>cd C:\Users\g_hus\Desktop\DesktopFile\Final Semester\Fall120-526- Blockchain\TermProject\Phase6\Infura\LoanPurse-Dapp-Infura\Lender\loanPurse-contract
C:\Users\g_hus\Desktop\DesktopFile\Final Semester\Fall120-526- Blockchain\TermProject\Phase6\Infura\LoanPurse-Dapp-Infura\Lender\loanPurse-contract>truffle migrate --network ropsten

Compiling your contracts...
=====
> Compiling .\contracts\LoanPurse.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to C:\Users\g_hus\Desktop\DesktopFile\Final Semester\Fall120-526- Blockchain\TermProject\Phase6\Infura\LoanPurse-Dapp-Infura\Lender\loanPurse-contract\build\contracts
> Compiled successfully using:
   - solc: 0.5.8+commit.23d335f2.Emscripten.clang

Migrations dry-run (simulation)
=====
> Network name: 'ropsten-fork'
> Network id: 3
> Block gas limit: 8000000 (0x7a1200)

1_initial_migration.js
=====
Deploying 'Migrations'
-----
> block number: 9212178
> block timestamp: 1607316227
> account: 0x3cD3AC7DCA765151e07dA6eb8AA2877C35e9DE1F
> balance: 89.7622208586840492366
> gas used: 175011 (0x2abaa3)
> gas price: 2 gwei
> value sent: 0 ETH
> total cost: 0.000350022 ETH

-----
> Total cost: 0.000350022 ETH
```

```
Select Command Prompt
> Block gas limit: 8000000 (0x7a1200)

1_initial_migration.js
=====

Deploying 'Migrations'
-----
> block number:      9212170
> block timestamp:   1607316227
> account:          0x3cD3AC29CA765151e87dA6eb8AA2877C35a9DE1F
> balance:          83.762220586840432366
> 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 (0x10302b)
> gas price:         2 gwei
> value sent:        0 ETH
> total cost:        0.002121814 ETH
-----
> Total cost:        0.002121814 ETH

Summary
=====
```

```
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:  0xbce97918bcbdd17d0eb23b8e214724660872d16073e55d848a36dab229a1a2fc
> Blocks: 1         Seconds: 44
> contract address: 0x9410e24c6d60101a86003C9610682a4762a58eae
> block number:     9212169
> block timestamp:  1607316217
> account:          0x3cD3AC29CA765151e87dA6eb8AA2877C35a9DE1F
> balance:          83.758770388840432366
> gas used:         190011 (0x2e63b)
> gas price:         20 gwei
> value sent:        0 ETH
> total cost:        0.003800022 ETH
-----

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:        0.003800022 ETH
```


Smart Contract Address:

```
Select Command Prompt
> 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

2_deploy_contracts.js
=====
> Blocks: 1 Seconds: 28
> contract address: 0x2f0Eb24ab1858E0804Cd03C7EC8215B32d62df72
> block number: 9211172
> block timestamp: 1607316294ab89aafd7a32c7bf641e84886238ace961cb34f425943dbb55bd1fba
> account: 0x3cD9AC29CA765151a87dA6eb8aA2877C35a9DE1F
> balance: 83.736405488840432366
> gas used: 1075907 (0x106ac3)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.02151814 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.02151814 ETH

Summary
=====
> Total deployments: 2
> Final cost: 0.02531836 ETH

C:\Users\g_hus\Desktop\DesktopFile\Final Semester\Fall120-526- Blockchain\TermProject\Phase6\Infura\LoanPurse-Dapp-Infura\Lender\loanPurse-contract>
```

Phase 6 - Conclusion:

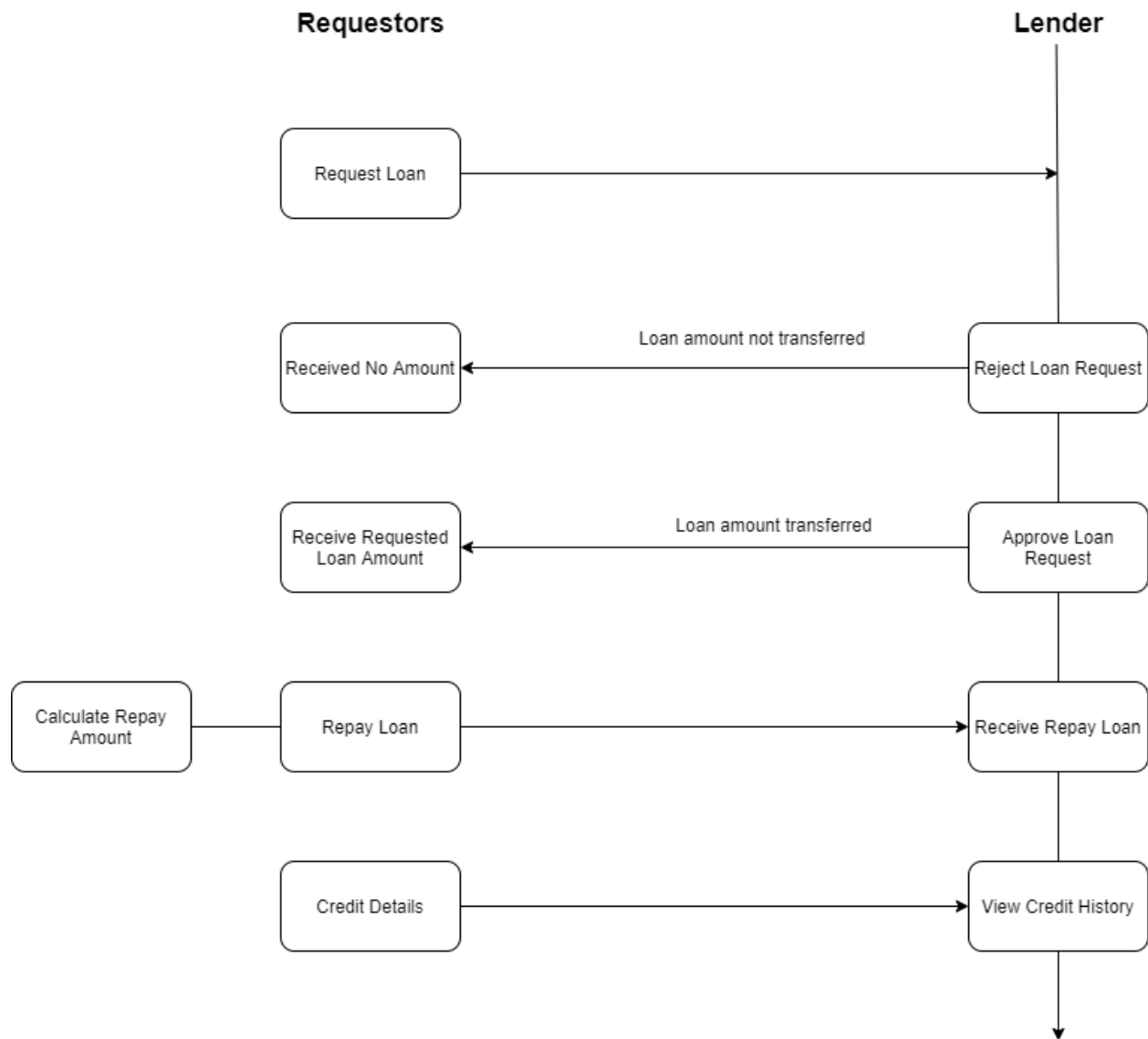
The Loan Purse Dapp has been successfully deployed on Infura cloud-like platform and available for the decentralized end-users to start using the application and interact with it for further requests.

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.

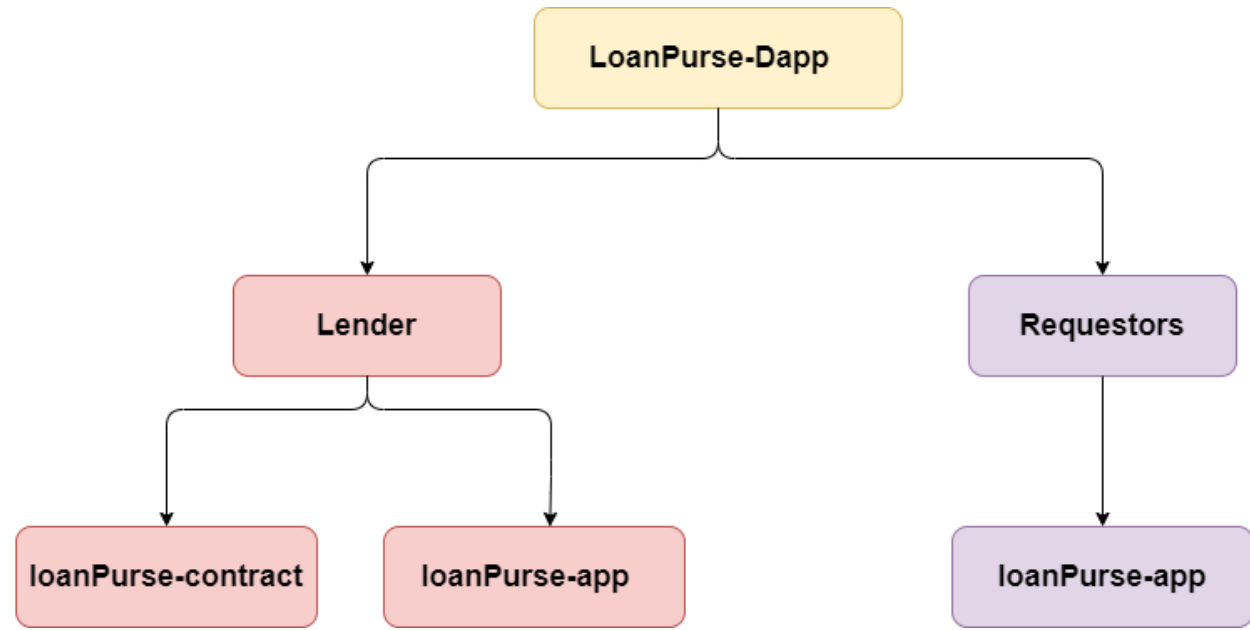
Interaction Diagram:



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:

```
Select Command Prompt
> 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

2_deploy_contracts.js
=====
> Blocks: 1 Seconds: 28
> contract address: 0x2f0eb24eb1858e0864Cd03C7EC8215832d82dF72
> block number: 9212172
> block timestamp: 1607316294ab89aaf7a32c7b641e84886238ace961cb34f425943dbb55bd1fba
> account: 0x5cD3AC29CA765151eB7dA6eb8aA2677C35a90E1F
> balance: 83.736405488840432366
> gas used: 1675907 (0x106ac3)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.02151814 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.02151814 ETH

Summary
```

Smart Contract Code Snippets:

```
1 pragma solidity >=0.4.22 <=0.6.0;
2
3 contract LoanPurse {
4     address payable public lender; // owner of the contract who deploy the contract
5     uint public calRepayAmount; // calculate the Repay Amount based on predefined interest rate
6     uint noOfLoan=0; // count of total Loans
7     uint amt;
8     mapping(address => Loan) loans; // mapping address to Loan
9     mapping(address => uint) public amount;
10
11     // Loan structure
12     struct Loan {
13         uint id;
14         address borrower;
15         uint loanAmount;
16         string loanStatus;
17     }
18
19     // Events are defined here
20     event Initialized();
21     event LoanRequested();
22     event LoanApproved();
23     event LoanRejected();
24     event LoanRepaid();
25     event myCalc(uint myRepay);
26     event loanCreditHistory(uint loanCount1, uint loanAmt1);
27
28
29     // modifier to check the functions which are applicable to only Lender can be done
30     // by Lender (e.g. approveLoan, rejectLoan)
31     modifier onlyLender()
32     { require(msg.sender == lender);
33     }
34
35     // modifier to check the functions which are not applicable to Lender can not be done
36     // by Lender (e.g. requestLoan, repayLoan)
37     modifier notLender()
38     { require(msg.sender != lender);
39     }
40
41     // constructor() is executed when the smart contract is deployed into the network
42     // and 'msg.sender' which is the Lender/owner, is the account creating this contract.
43     constructor () public {
44         lender = msg.sender;
45         emit Initialized();
46     }
47
48     // function to request Loan by borrower to lender by giving few details about the borrower and reason for loan
49     function requestLoan(uint loanAmount) notLender public {
50         // add Loan object to mapping
51         amount[msg.sender] = loanAmount * 1000000000000000000;
52         loans[msg.sender] = Loan(noOfLoan, msg.sender, loanAmount, "Loan Requested");
53         emit LoanRequested();
54     }
55
56     // function to approve the loan request by lender and transfer the loan amount to borrower account
57     function approveLoan(address payable requestorAddress) onlyLender payable public {
58         require(msg.sender == lender);
59         require(msg.value > amt, "Insufficient Balance.");
60         amt = amount[requestorAddress];
61         requestorAddress.transfer(amt);
62
63         // Transfer the remaining amount to lender's account after transferring the requested loan amount from msg.value amount
64         uint tempBalance = 0;
65         tempBalance = msg.value - amt;
66
67         // transfer the requested Loan Amount to Requestor's account
68         msg.sender.transfer(tempBalance);
69
70         noOfLoan++;
71         loans[requestorAddress] = Loan(noOfLoan, requestorAddress, amt, "Loan Request Approved");
72         emit LoanApproved();
73     }
74
75     // function to reject the Loan request by Lender
76     function rejectLoan(address requestorAddress) onlyLender public {
77         require(msg.sender == lender);
78         loans[requestorAddress].loanStatus="Loan Request Rejected";
79
80         emit LoanRejected();
81     }
82
83     // function to view the Credit History of the Requestor
84     function creditHistory(address requestorAddress) public returns(uint , uint) {
85         Loan memory loanDetails = loans[requestorAddress];
86         uint loanCount = loanDetails.id;
87         uint loanHist = (loanDetails.loanAmount / 1000000000000000000);
88         emit loanCreditHistory(loanCount , loanHist);
89
90         return (loanCount, loanHist);
91     }
92
93     // Check current Balance of the selected address
94     function checkBalance(address addr) view public returns(uint){
95         return (addr.balance / 1000000000000000000);
96     }
97
98 }
99
100
101
102
```

```

192
193 // function to check whether borrower is able to repay the amount to Lender's account address
194 function calculateRepay(address requestorAddress, uint requestedLoanAmount) notLender public returns (uint){
195     // Calculating repay amount based on predefined interest rate on principal amount(LoanAmount)
196     calRepayAmount = (((amount[requestorAddress] *10*2)/100) + requestedLoanAmount * 1000000000000000000 / 1000000000000000000);
197     emit myCalc(calRepayAmount);
198
199     return calRepayAmount;
200 }
201
202 // function to repay the loan to the Lender
203 function repayLoan(address payable lenderAddress, uint repayAmt) notLender public payable {
204     require(msg.sender.balance >= repayAmt);
205     require(msg.value > repayAmt+1000000000000000000, "Insufficient Balance.");
206     require(repayAmt == calRepayAmount);
207
208     // transfer the calculated Repay Amount to Lender's account
209     lenderAddress.transfer(calRepayAmount * 1000000000000000000);
210
211     // Transfer the remaining amount to borrower's account after deducting the repay amount from msg.value amount
212     uint tempBalance = 0;
213     tempBalance = msg.value - (calRepayAmount * 1000000000000000000);
214     msg.sender.transfer(tempBalance);
215     loans[msg.sender].loanStatus = "Loan Amount Paid";
216
217     emit LoanRepaid();
218 }
219 }

```

ContractDefinition LoanPurse 0 reference(s)

Web app(UI) screenshots:

Lenders' Screen:

LoanPurse Dapp

Current Account: 0x3cd3ac25ca765d5feb74956eb5a2877c3a9de1f
Balance(ETH): 79.73399334854828

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

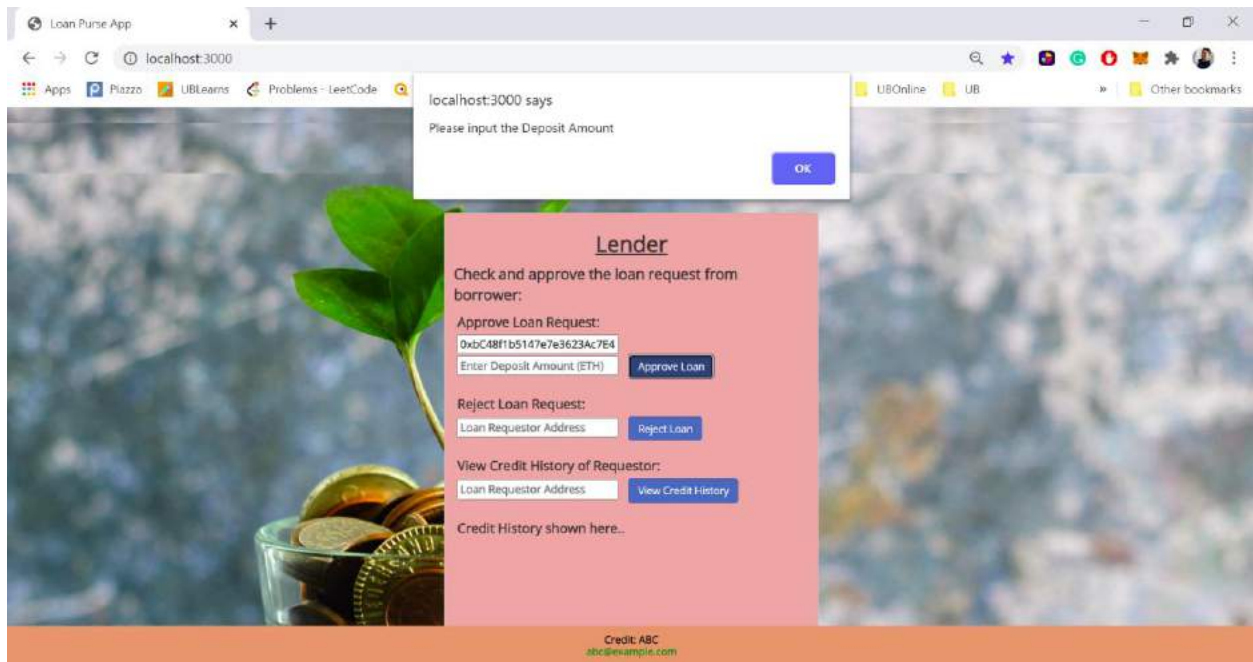
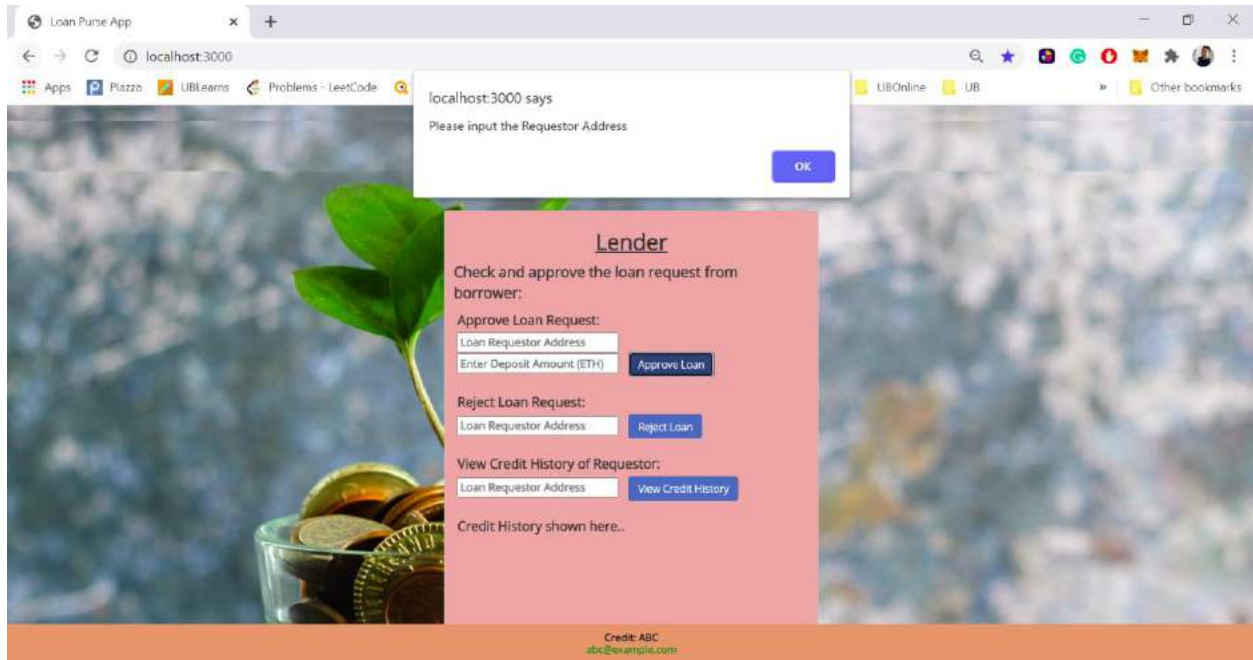
View Credit History of Requestor:

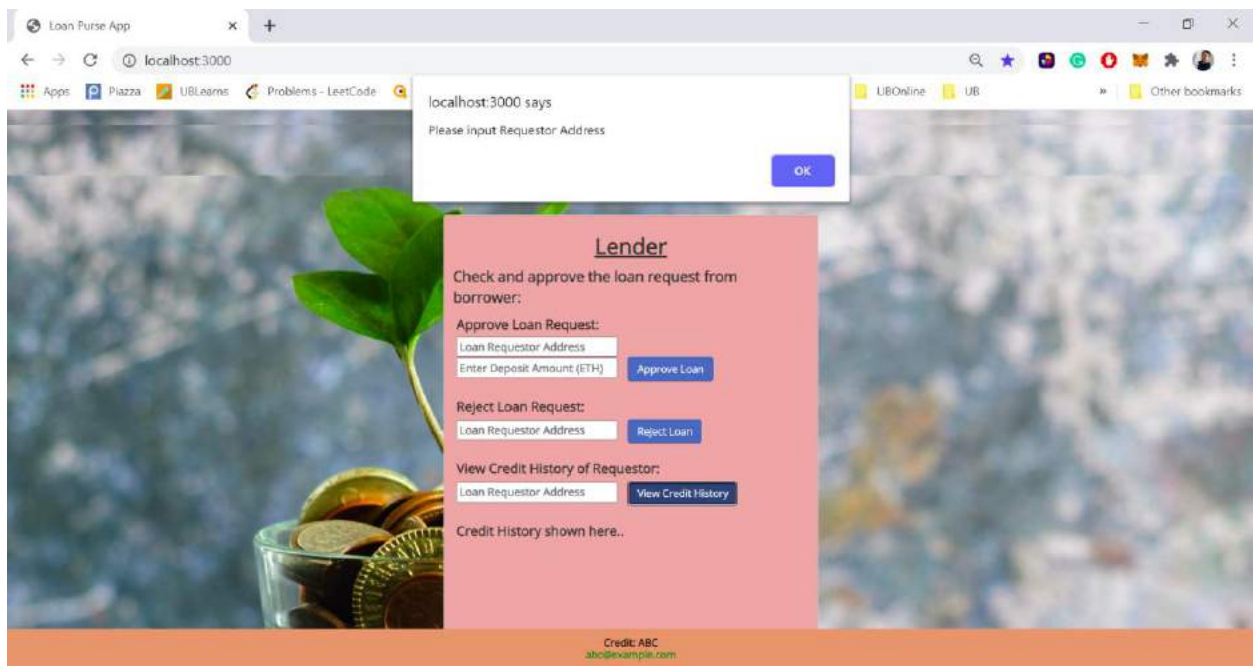
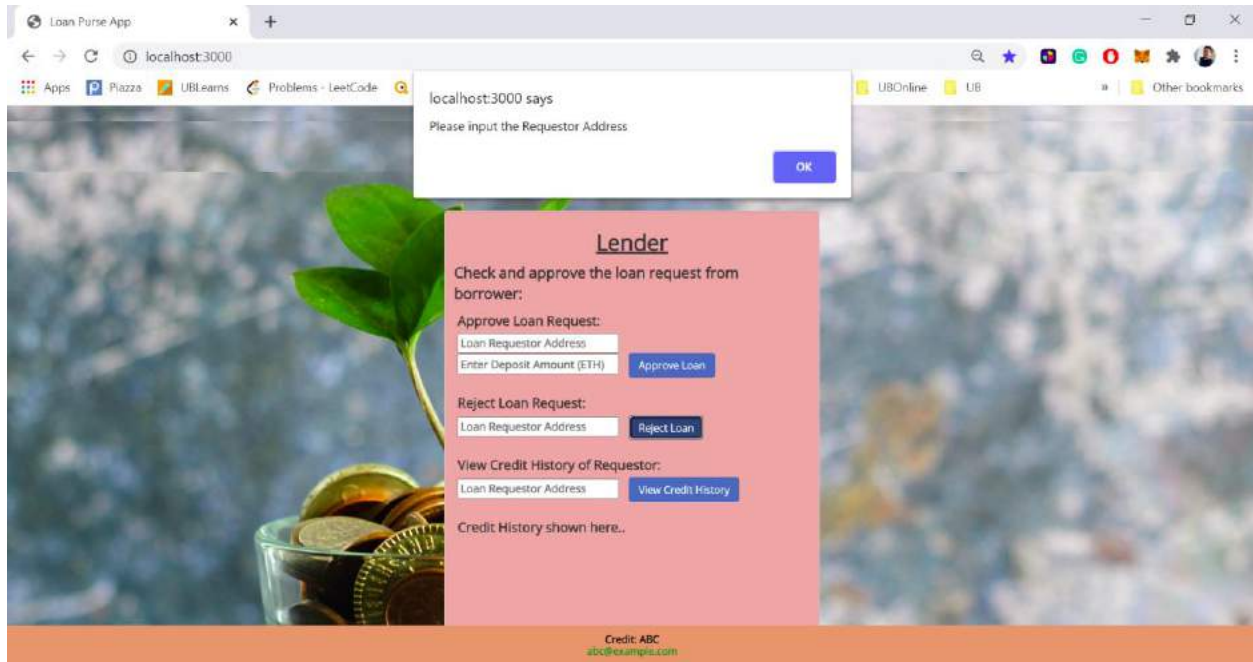
Loan Requestor Address: View Credit History

Credit History shown here..

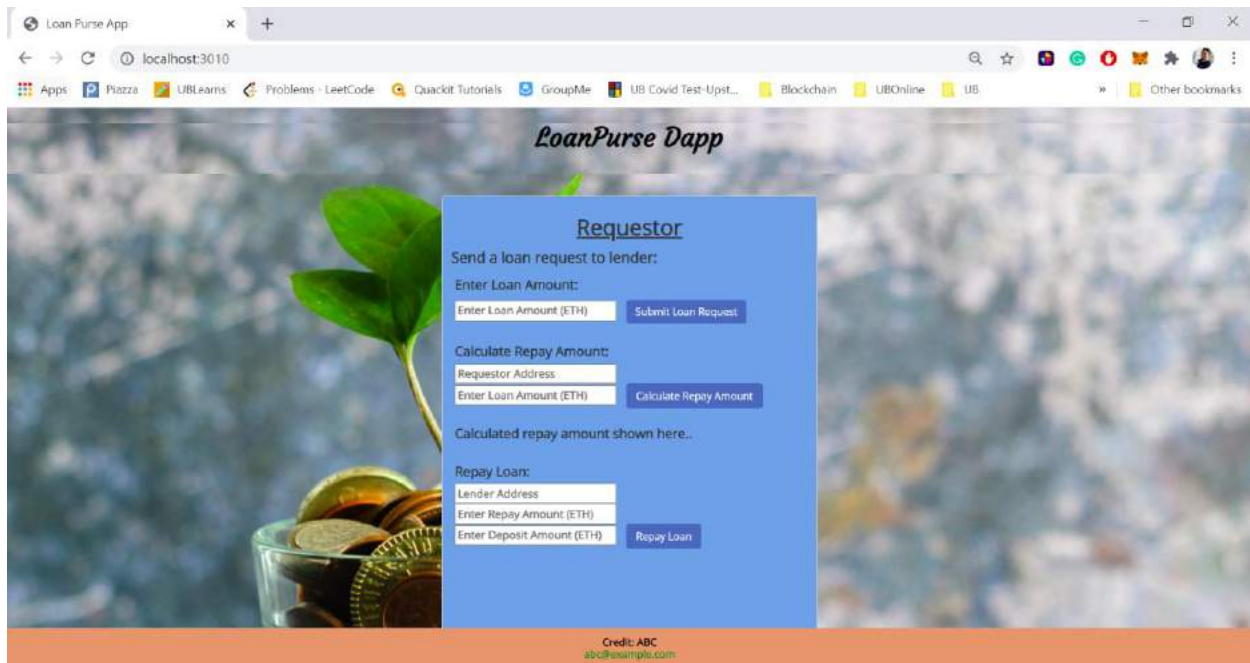
Credit: ABC
abc@evample.com

Validation of blank fields:





Requestors' screen:



The screenshot shows a web browser window titled "LoanPurse App" at the address "localhost:3010". The page features a background image of a green plant growing from a glass filled with gold coins. A blue overlay contains the "Requestor" form. The form has three main sections: "Send a loan request to lender:", "Calculate Repay Amount:", and "Repay Loan:". Each section contains input fields and a corresponding button. At the bottom of the blue overlay, it says "Credit: ABC" and "abc@example.com".

LoanPurse Dapp

Requestor

Send a loan request to lender:

Enter Loan Amount:

Enter Loan Amount (ETH)

Calculate Repay Amount:

Requestor Address

Enter Loan Amount (ETH)

Calculated repay amount shown here..

Repay Loan:

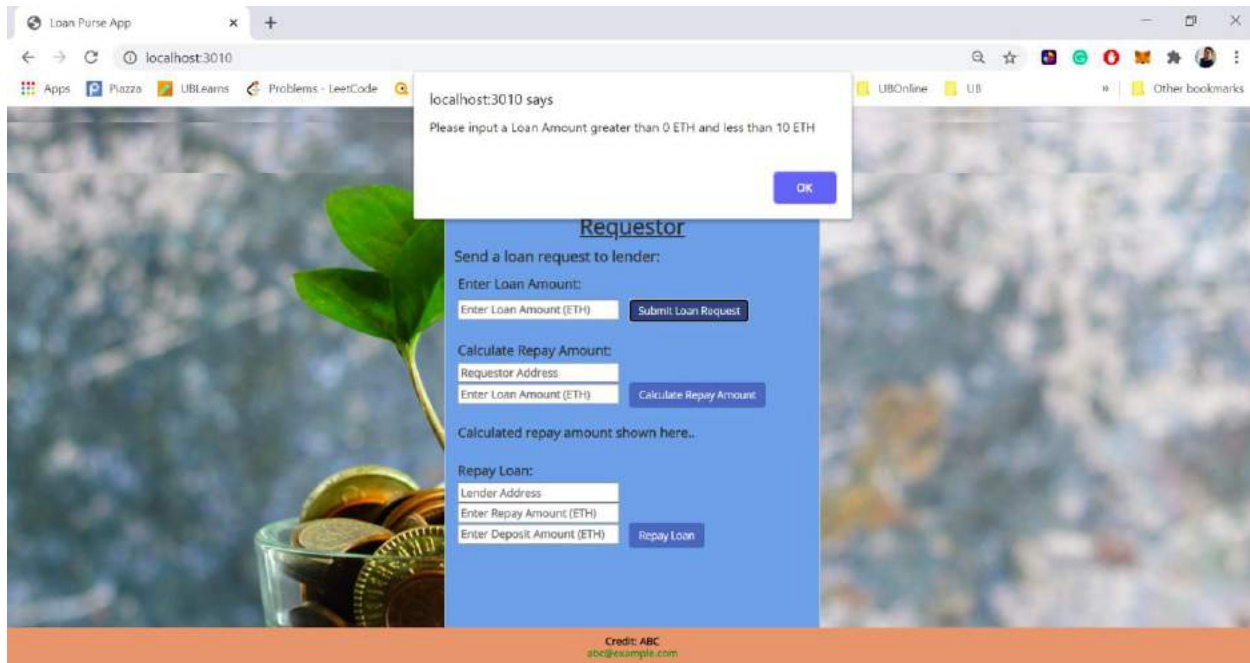
Lender Address

Enter Repay Amount (ETH)

Enter Deposit Amount (ETH)

Credit: ABC
abc@example.com

Validation of blank fields:



This screenshot is identical to the previous one, but with a white modal dialog box overlaid in the center. The dialog box contains the text "localhost:3010 says" and "Please input a Loan Amount greater than 0 ETH and less than 10 ETH", with an "OK" button at the bottom right.

localhost:3010 says

Please input a Loan Amount greater than 0 ETH and less than 10 ETH

LoanPurse Dapp

Requestor

Send a loan request to lender:

Enter Loan Amount:

Enter Loan Amount (ETH)

Calculate Repay Amount:

Requestor Address

Enter Loan Amount (ETH)

Calculated repay amount shown here..

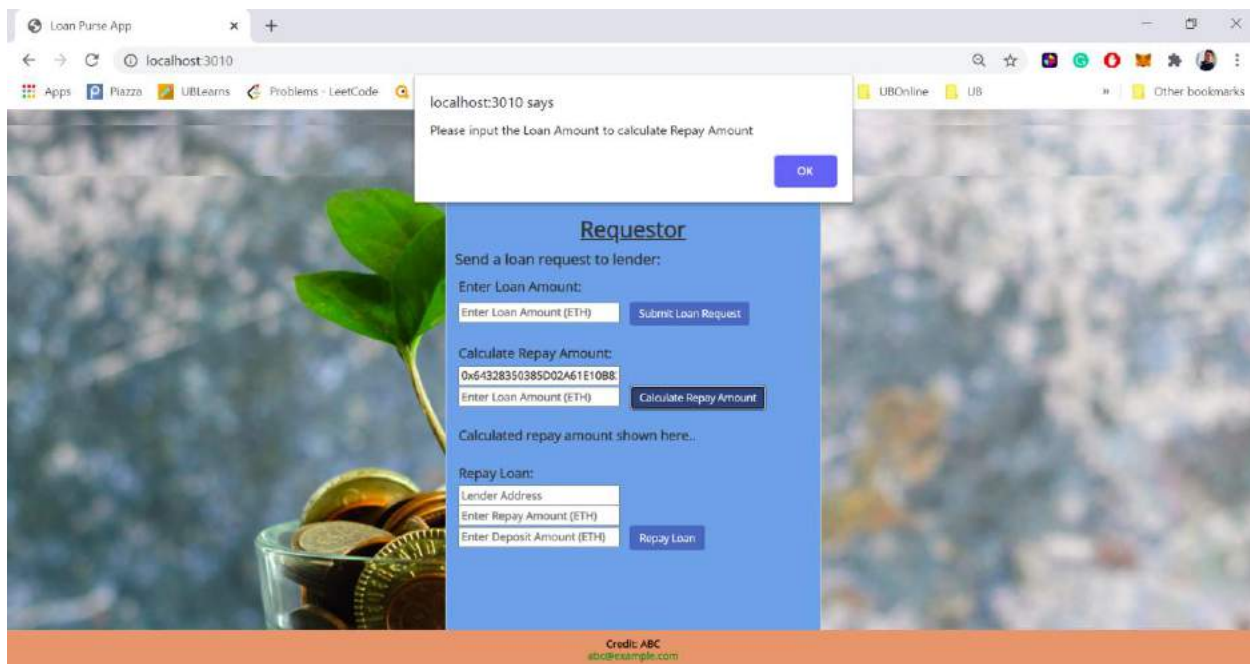
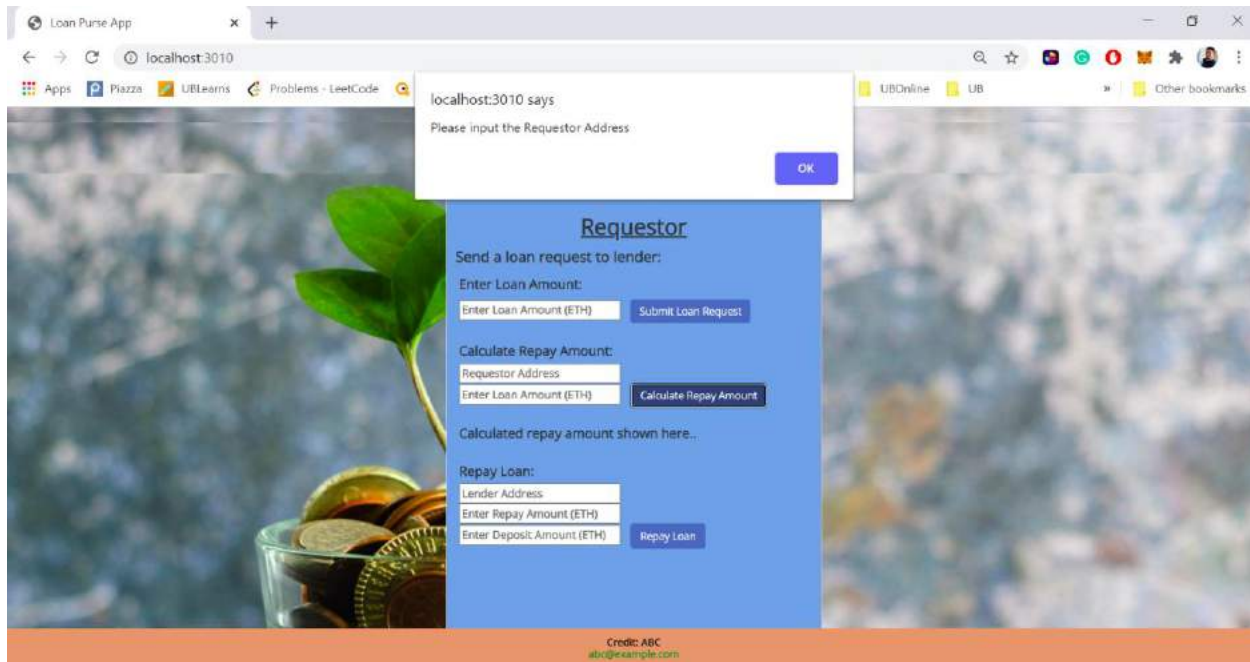
Repay Loan:

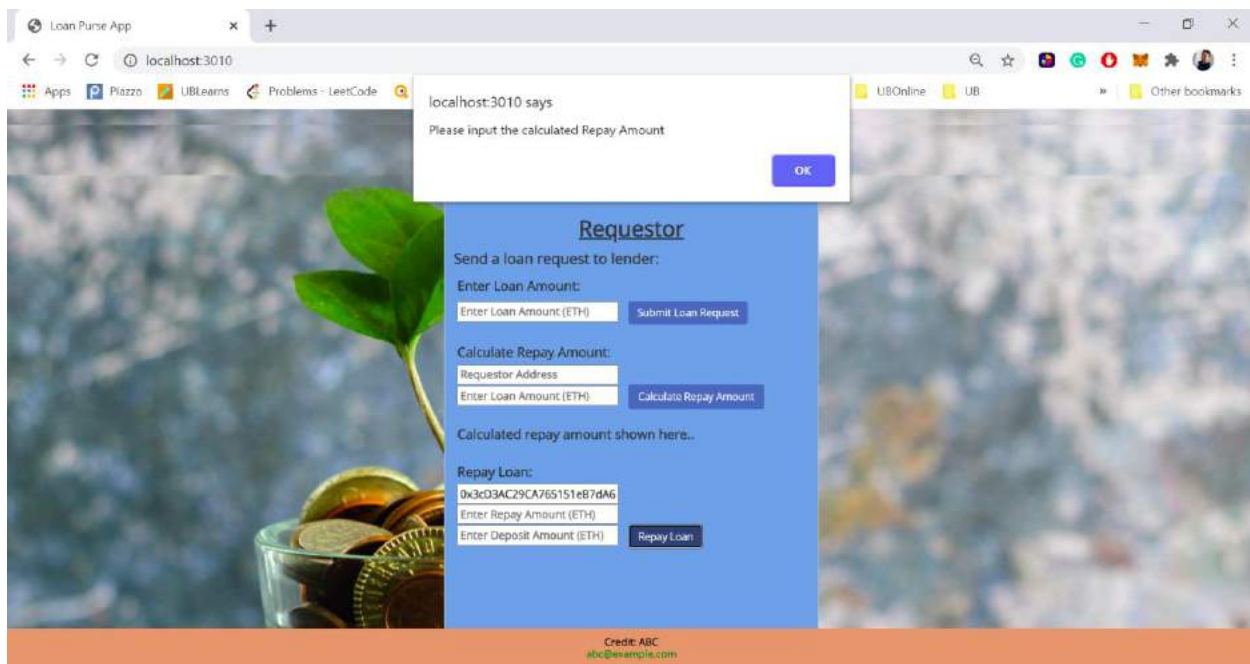
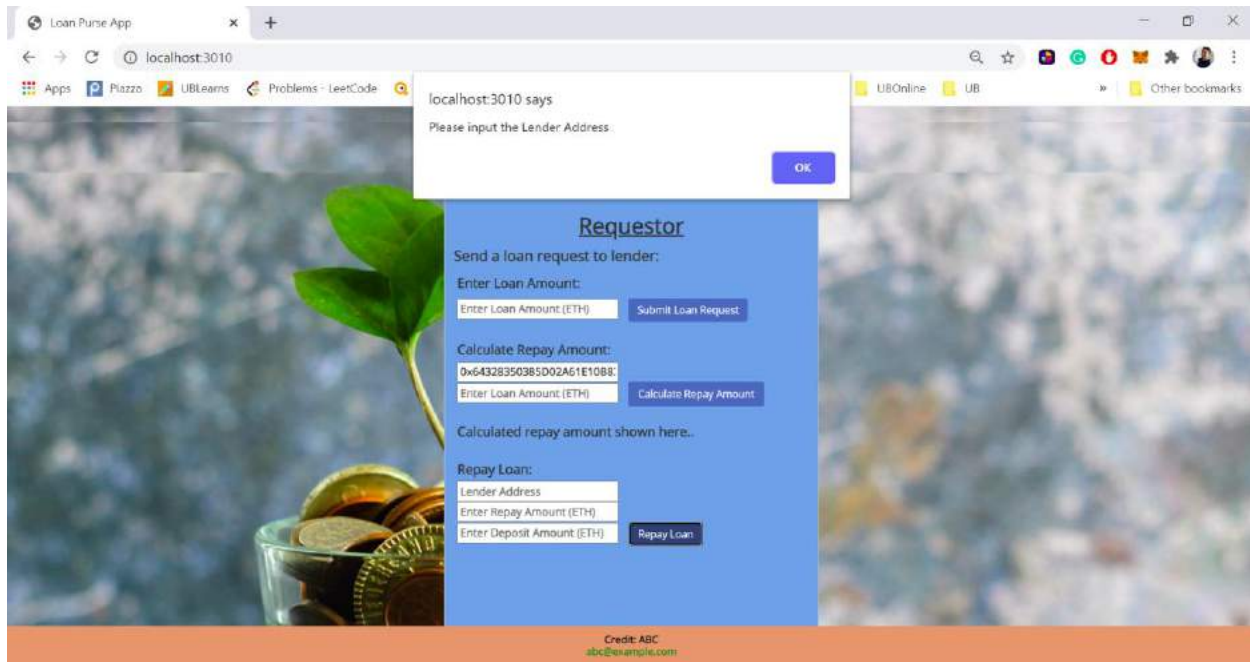
Lender Address

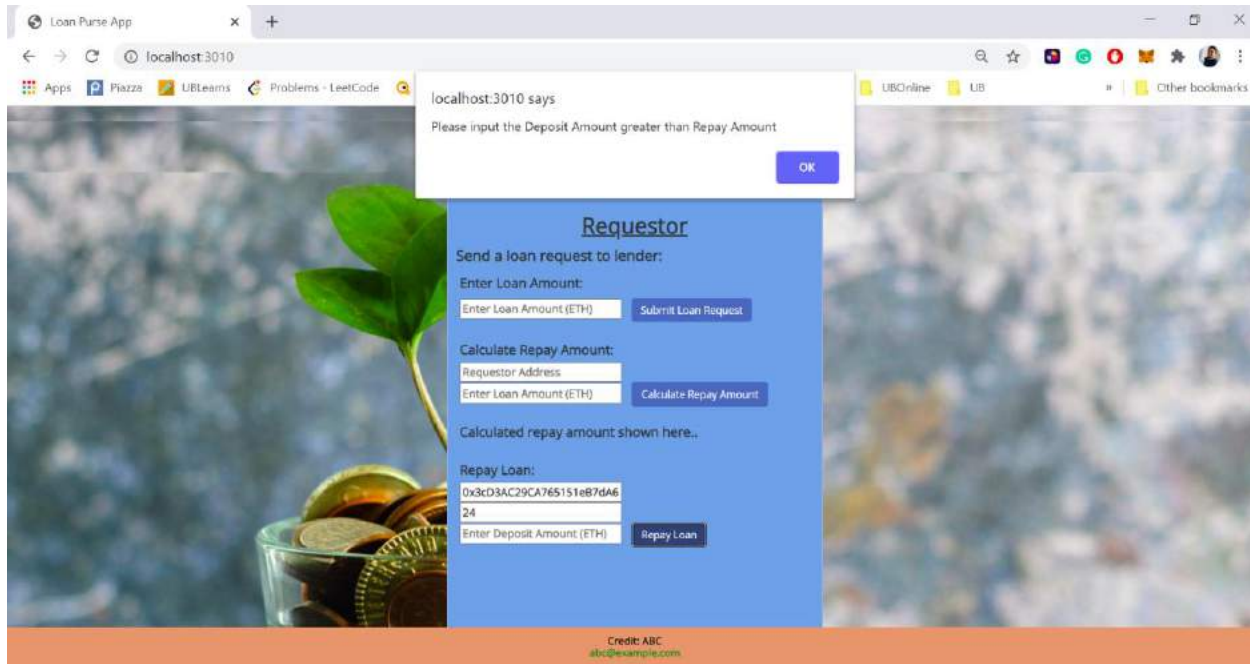
Enter Repay Amount (ETH)

Enter Deposit Amount (ETH)

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.