



Blockchain Based

DECENTRALIZED LOAN SYSTEM.

Aliasgar Merchant



Decentralized Loan System

Table of Contents

1. ABSTRACT	3
2. INTRODUCTION.....	4
GOALS AND OBJECTIVES	4
PURPOSE OF THE PAPER	4
3. WORKING OF EXISTING SYSTEM	5
ADMINISTRATIVE DEPARTMENT	5
LOAN BORROWER	5
LOAN LENDER.....	5
4. UNDERSTANDING THE SYSTEM.....	6
NEEDY PERSPECTIVE	6
ELITE PERSPECTIVE:	6
ADMIN PERSPECTIVE.....	7
FIG: TRADITIONAL LOAN SYSTEM – CLASS DIAGRAM	8
FIG: TRADITIONAL LOAN SYSTEM – FLOW CHART	9
5. AREA OF IMPROVEMENT	10
NEEDY PERSPECTIVE:	10
ELITE PERSPECTIVE:	10
ADMIN PERSPECTIVE.....	11
6. SOLUTION.....	12
NEEDY PERSPECTIVE	12
ELITE PERSPECTIVE	13
ADMIN PERSPECTIVE.....	13
FIG: DECENTRALIZED LOAN SYSTEM – LOAN REQUEST AND APPROVAL	14
FIG: DECENTRALIZED LOAN SYSTEM – REPAYMENT OF LOAN	15
FIG: DECENTRALIZED LOAN SYSTEM – DONATION	16
7. USE CASE	17
USE CASE DEVELOPMENT METHODOLOGY.....	17
CASE 1: LOAN APPROVED	18
CASE 2: LOAN REJECTED.....	19
CASE 3: LOAN REJECTED – VOTING DEMANDED.....	20
<i>Voting Accepted – Loan Approved</i>	20
<i>Voting Accepted – Loan Rejected</i>	21
CASE 4: LOAN DEFAULTED	22
CASE 5: LOAN DEFAULTED AFTER VOTING	23
CASE 6: DONATION / WITHDRAWAL OF FUNDS.....	24
8. SOFTWARE CODES	25
MAIN CODE – DECENTRALIZED LOAN SYSTEM	25
MORTGAGE CODE (COLLATERAL) – DECENTRALIZED LOAN SYSTEM	29
VOTING CODE – DECENTRALIZED LOAN SYSTEM.....	32
MIGRATION CODE – DECENTRALIZED LOAN SYSTEM	33
FUTURE SCOPE	34
CONCLUSION	36

Decentralized Loan System

AREA OF IMPROVEMENT	36
SOLUTION (FUTURE STEPS)	37
APPENDIX.....	38
DEFINITIONS.....	38
1. <i>Decentralized Loan System</i>	38
2. <i>Existing System</i>	38
3. <i>Administrative Department / Admin</i>	38
4. <i>Loan Borrower / Needy</i>	38
5. <i>Loan Lender / Elite</i>	38
6. <i>Blockchain v/s Other System</i>	38
7. <i>Vote</i>	39
8. <i>Partial Transparency</i>	39
DECENTRALIZED LOAN SYSTEM – GITHUB.	40
WHITE PAPER.....	40
AUDIT SMART CONTRACT	40
COLLATERAL SMART CONTRACT	40
ELECTION SMART CONTRACT	40
LOAN SMART CONTRACT	40
BIBLIOGRAPHY.....	41

Decentralized Loan System

1. Abstract

Decentralized Loan System is designed to address real world problems faced by 'Loan Systems' within a specific institution. The purpose of the paper is to demonstrate the benefits of **Interest Free** loans within a society. While – this kind of system is majorly based on a peer to peer system, we also introduce 'Admin' or 'Administrator' for uninterrupted operations within the institution. There are three major actors within the loan system –

Elite – They are responsible for lending funds in good faith within the institution.

Needy – They request for loan as per need.

Admin – They overlook the operations performed within the institution.

The paper also explains in depth the concept of **transparency** which is sometimes missing within these institutions. **Decentralized Loan System** is based on a **Blockchain** system to address this issue. Blockchain being **immutable** offers the stake holders the ability to *audit* all the transactions.

Also, the paper describes in detail – the **right to vote** which is bestowed upon the stake holders. This empowers them to take decisions as to *who shall be granted loan, what amount shall be granted and what shall be the repayment term*.

While, the loan is based on an **Interest Free** system – however collateral must be required at all times. The paper also explains, if the collateral offered is not sufficient how can the needy be still benefited from the Loan System.

In the end, the paper also talks about the future scope and benefits of using this kind of system.

Major Software Modules:

1. Loan Request / Loan Dispersal / Loan Repayment Smart Contract.
2. Collateral / Mortgage Smart Contract.
3. Voting Smart Contract.
4. Audit Smart Contract.

Decentralized Loan System

2. Introduction

Decentralized Loan System specifically considers example of one particular 'private institution.'

This segment of the paper explicitly tries to explain the working of that institution. Though, it might be presumed at first instance that this is a very specific problem solution catering to very specific audience, but this idea can be well expanded to fit in a number of scenarios.

However, please note given the great use-case of this institution and its ability; it may be used or replicated at a greater extent by any other institution.

Goals and Objectives

- Introduction of paperless system for loan application.
- Introduction of paperless system for loan mortgage.
- Trustless system to distribute loan amount.
- Decentralized system for loan recovery / loan repayment.
- Decentralized voting system for non-collateral loan sanction.
- Trustless system for loan repayment.

Purpose of the paper

The paper is specifically written for a particular institution. However, given the great use case – this paper tends to solve problems which can be bifurcated into different components which can be used individually to solve other problems.

Components:

- Paperless system.
- Trustless authentication.
- Unbiased decisions.
- Transparent function.

Decentralized Loan System

3. Working of existing system

The ‘private institution’ mentioned above – works in the following ways:

There are three major *actors* in the existing loan process constituted by the private institution.

Administrative Department

Actor 1:

Let’s consider this actor to be “*Administrative Department*” also referred as *admin* throughout the document which consist of the following features.

- a. Advertising for donating funds and increasing the pool of funds available.
- b. Managing the funds – that is returning the funds to lending parties as and when required.
- c. Allocating the funds to borrowers depending on the predefined parameters.

Loan Borrower

Actor 2:

Let’s consider this actor to be “*Loan Borrower*” also referred as *needy* throughout the document which consist of the following features.

- a. (S)He will request for certain amount of money depending on the need of the *needy*.
- b. Based on the predefined parameters, the allocation is done by the trustees to *needy*.
- c. The *needy* are subjected to submit collateral as defined by the *admin* department.

Loan Lender

Actor 3:

Let’s consider this actor to be “*Loan Lender*” also referred as *elite* throughout the document which consist of the following features.

- a. (S)He will lend the funds and agree upon fund withdrawal rules.
- b. (S)He will request for the sum of the money as and when required by the *elite*.

Decentralized Loan System

4. Understanding the system

Needy Perspective

The *needy* will request for certain amount of money – depending upon the nature of the need. The *needy* will then submit all the necessary documents which demonstrate the ‘actual need’ of the *needy*. Then the *needy* is required to submit a detailed plan of how (s)he plans to return the money to the institution within given frame of time. Apart from all these details – the *needy* is required to submit a collateral of the loan. This collateral should be such, that it exceeds the value of the loan being issued. If, for any reason, the *needy* is unable to repay the loan; the collateral would be responsible for loan repayment.

Also, an important perspective to consider – the loan issued by the institution is **100% interest free**. Also, there are no “service” or other “hidden cost.” Hence, the repayment of the entire loan amount will be same as the amount borrowed. Also, if the *needy* defaults the loan amount – steps are taken as described by the guidelines of the institution.

Elite Perspective:

The *elite* will willingly give or offer certain amount of money to the pool to facilitate the institution to lend the money to people in need. While doing so the *elite* agrees to the general conditions laid down by the institution. The *elite* may hold the money with the institution for indefinite amount of time. If needed, the *elite* can request the money from the institution. The institution promises to repay the full amount requested within 3 to 7 business days. Also, since this is no interest scheme, the *elite* gets only the amount he had submitted initially. Hence, this should not be considered as a means of making profits or earning money; but purely as social cause done in good faith. Also, there are provisions to offer money in this pool recursively every month.

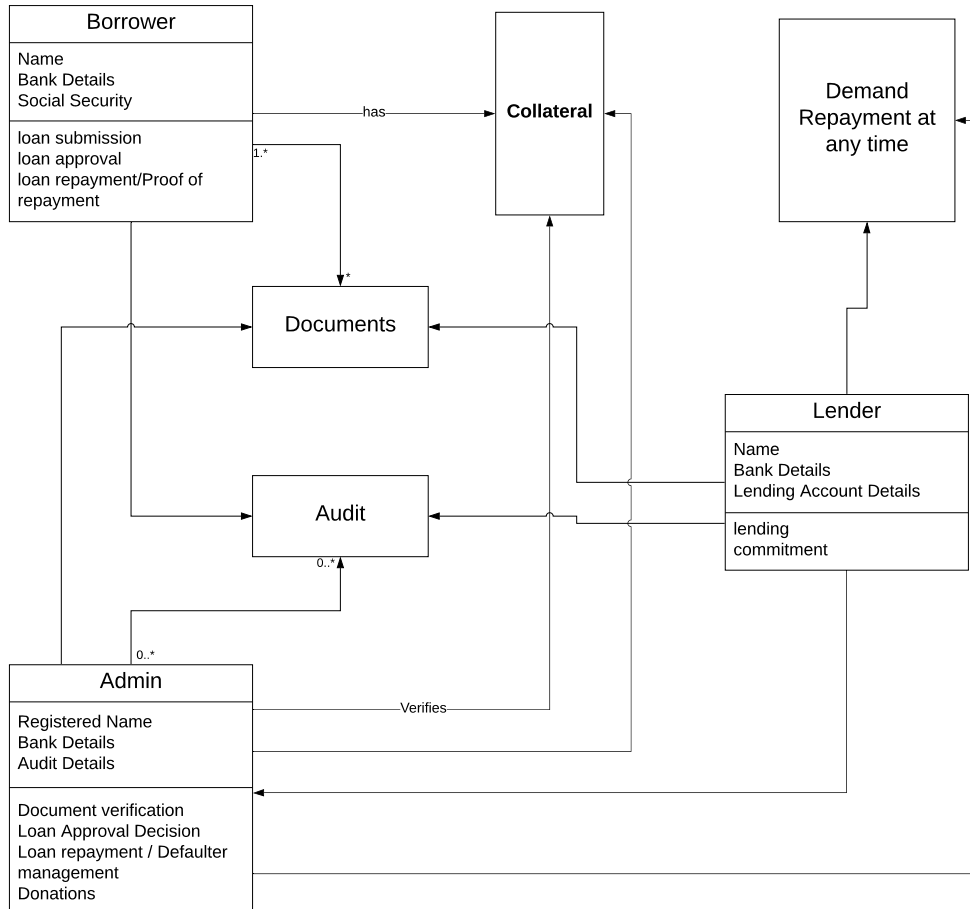
Decentralized Loan System

Admin Perspective

They are responsible for advertising the ideology so more and more people offer money in the pool. Once *needy* request for money, they check all the documents and necessary credentials. This is similar to running a complete credit check. They also manage timely repayment to *elite* as and when they demand. Also, they allocate the funds based on their discretion and availability of collateral.

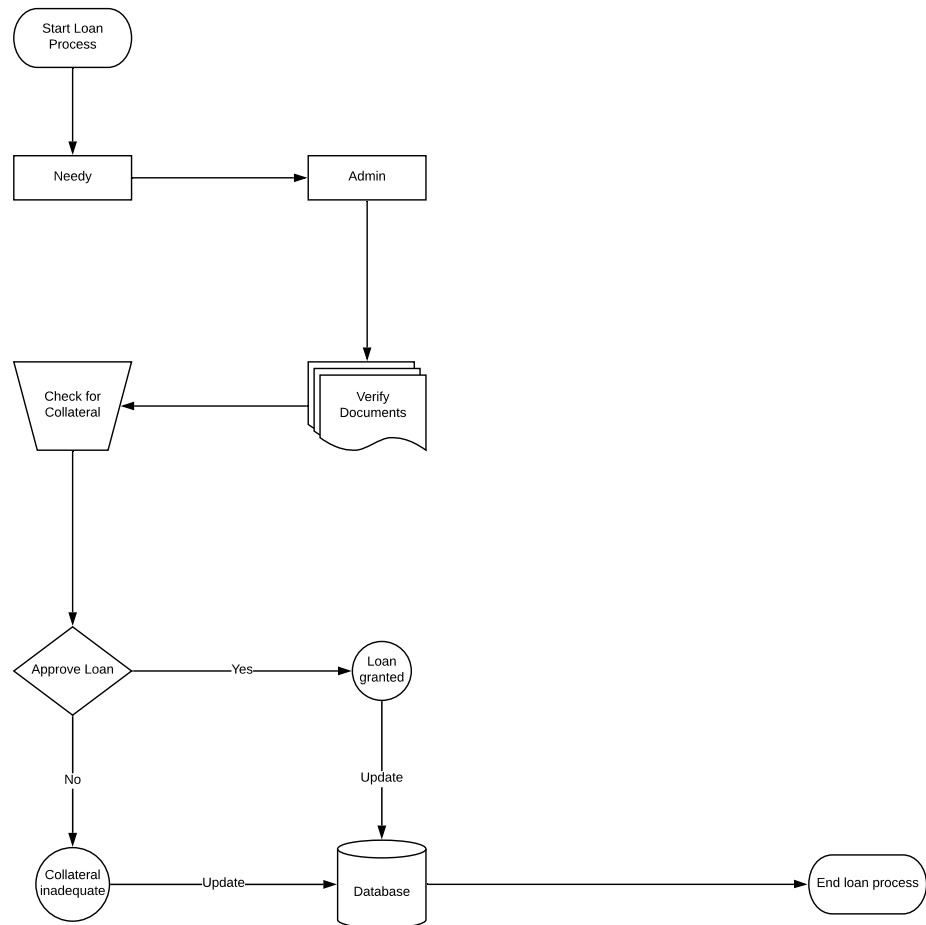
Decentralized Loan System

Fig: Traditional loan system – Class Diagram



Decentralized Loan System

Fig: Traditional loan system – Flow chart



Decentralized Loan System

5. Area of Improvement

This is a very crucial part of the paper – as most of the parts described here will serve as the **‘solution’** to **‘special problem.’**

Before we define – the areas where the model can be improved let’s consider the factors which influence these ‘area of improvement.’ We will define two perspective that is – one of the *elite* and other of *needy*. These serve as the foundation to the solution proposed in this paper. However, it should also be noted that – these perspectives only serve as a guideline or suggestion and not all ideas can be implemented in the solution.

Needy Perspective:

The entire process should be more transparent – especially if the *needy* is being denied the loan amount.

They should be given the option of ‘no collateral’ loan option – only if the stake holders agree to it.

Elite Perspective:

They should have ‘on demand’ transparency. The *elite* should have a right to ‘vote’ as to where and how the funds are being used. Note – The right to vote is a very subjective feature. Later parts of the paper explain in depth the concept of voting.

Also, the *elite* are able to see on demand every request made to the institution and other details. The transparency is subjective and not everything is revealed to the elite. Note – The paper also explains the reason for subjective transparency in later parts.

Decentralized Loan System

Admin Perspective

The funds collected in existing system is stored, managed and distributed locally. The decision made by committee to allocate these funds have little or no influence from central authority. We try to implement a way in which existing resources can be pooled – especially in same geographic location as country. This will make efficient use of more resources. However, this ‘use’ is strictly defined using guidelines and standards explained in later parts of paper.

Decentralized Loan System

6. Solution

In this part of the paper we focus on solving the existing problems defined in 'area of improvement.'

We distribute the definition of solution as described in area of improvement.

The solution we offer is majorly focused on 'Blockchain based solution.' This segment of the paper evaluates how Blockchain can solve all existing concerns raised by respective actors. A supplement to this segment is comparison between Blockchain technology and other technologies and why Blockchain is better apt for this situation.

Needy Perspective

Blockchain in general makes the entire transaction more transparent and open. One of the major uses at this stage – is for auditing which solves one of the concerns of the needy.

Once – the needy is rejected a loan, he/she can ask for a non-collateral loan from the stake holders.

Here – the stake holders (typically 5 – depends largely on the amount to be dispersed) can vote for dispersing the loan to the needy. Note: if the amount exceeds the defined amount for loan disperse without a collateral - then the Chairman of the committee may be needed to invoke.

If the needy fails to repay the said amount in said time – he/she may be permanently or temporarily banned from using the loan service.

Also, the stake holders who voted for the respective needy may lose the credibility of voting and would have to pay the amount as defined by the management.

Decentralized Loan System

Elite Perspective

Blockchain solution makes the entire transaction transparent- readily available to view at any given point of time.

Elite has a right to vote – however they might need to hold responsibility for the same. This feature is again subjective – depending on the credibility of user.

(Credibility can be defined by member since how long, amount invested, etc.)

The elite can also see – all the request which were made by needy – which were approved, and which were rejected. However, personal information such as name or SSN are secured.

The elite may also demand – a in detail reason as to why a certain loan was rejected or certain amount was approved.

Admin Perspective.

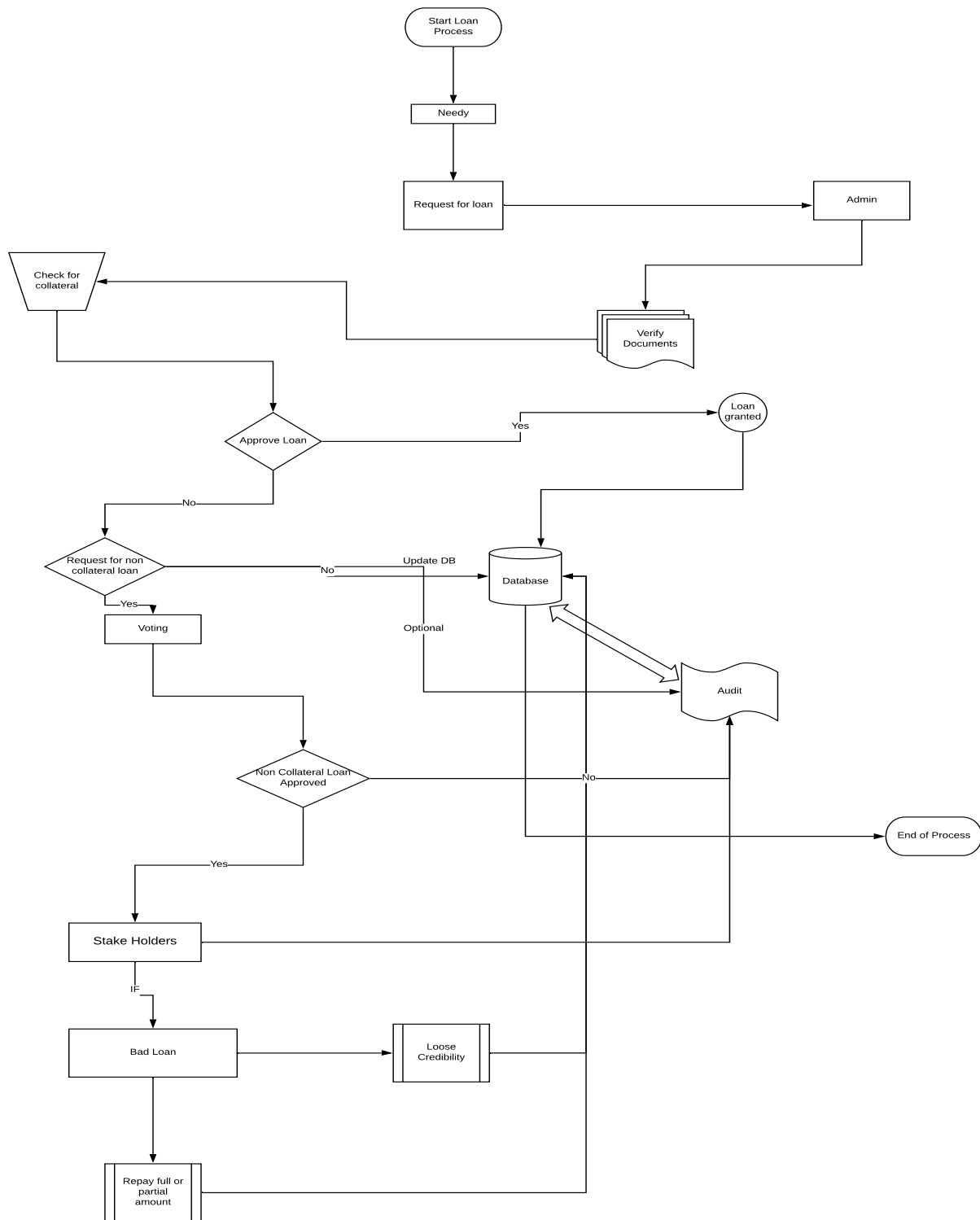
The admin may have time to time events in order to promote this kind of system.

Also, it may request money from other pools within a geographical location or may also donate to other pools if necessary. This requires necessary approvals.

Also, in certain cases it may need approval from central authority.

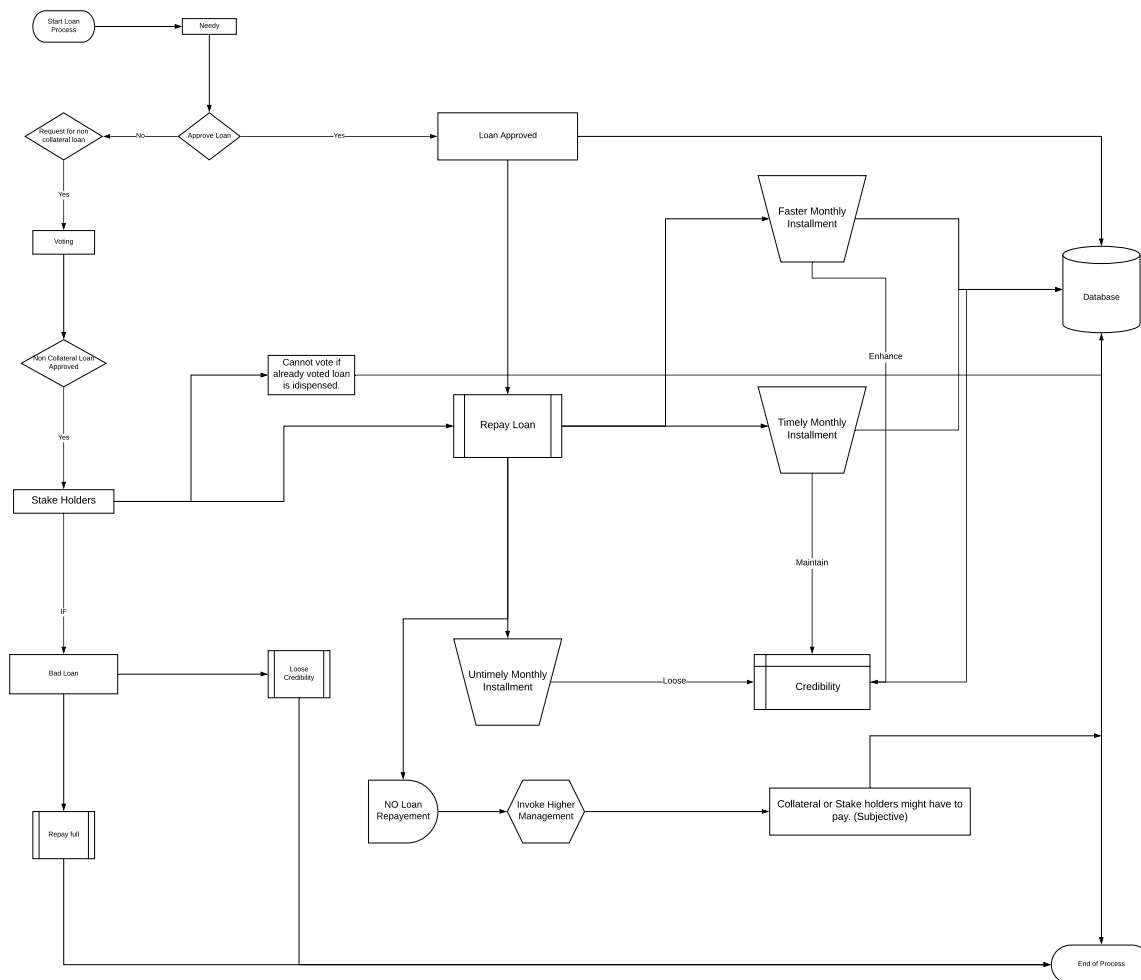
Decentralized Loan System

Fig: Decentralized loan system – Loan request and approval



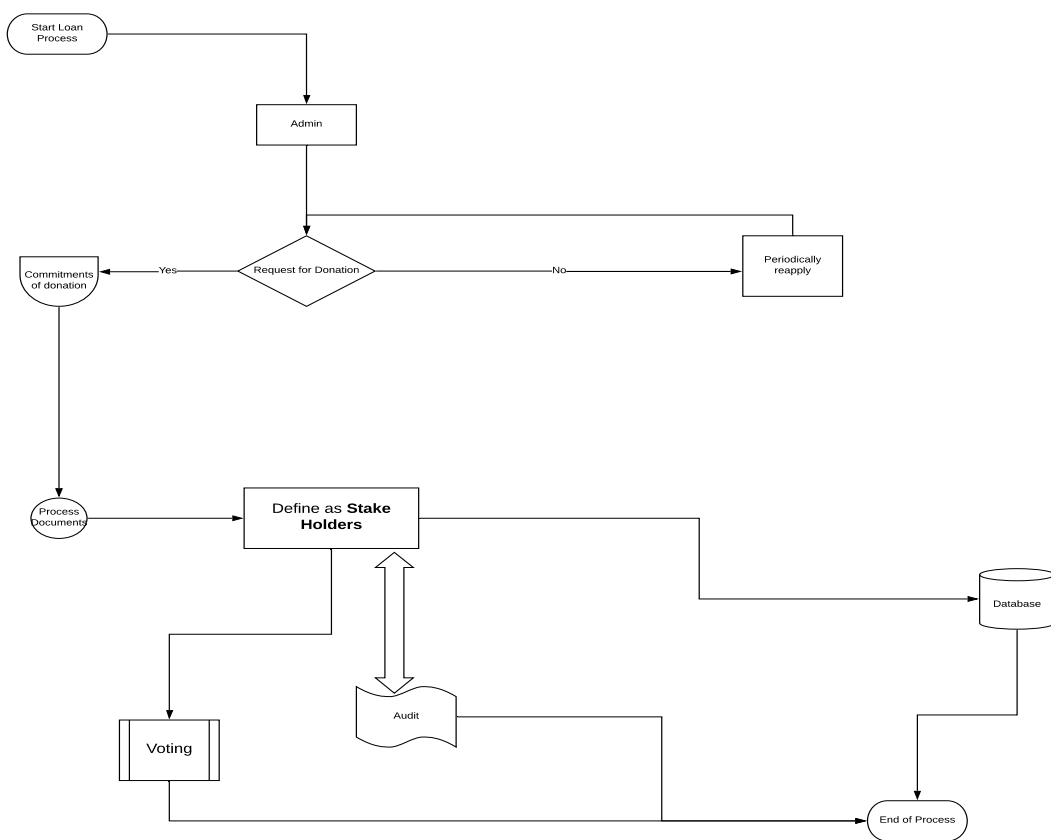
Decentralized Loan System

Fig: Decentralized loan system – Repayment of loan



Decentralized Loan System

Fig: Decentralized loan system – Donation



Decentralized Loan System

7. Use Case

Use case development methodology

The paper describes the use cases of Decentralized Loan System in the following way.

Loan Approved / Loan Rejected:

The idea behind developing these use cases is very simple – based on the documents like collateral and credit check the admin can decide to grant or reject loan. Consider the loan is approved – however, it may be less than what was anticipated. Hence, it may also be treated as loan rejected.

When a loan is rejected (or less than anticipated) the needy can demand a detailed analysis of why the loan was rejected. In certain cases, the needy may demand for a voting.

Loan Rejected – Voting:

As mentioned in previous use case, if the loan is rejected the needy may demand a voting. Depending on various circumstances – for e.g.: nature of the loan, previous loan history, etc. the stake holders may reject or approve the request of voting.

Loan Defaulted:

Another perspective which comes into picture is what happens when the loan is defaulted. In certain situations (depending upon the consensus of the stake holders), the loan installment may be relaxed for certain period. Also, the stake holders may unanimously decide to waive loan or a part of loan.

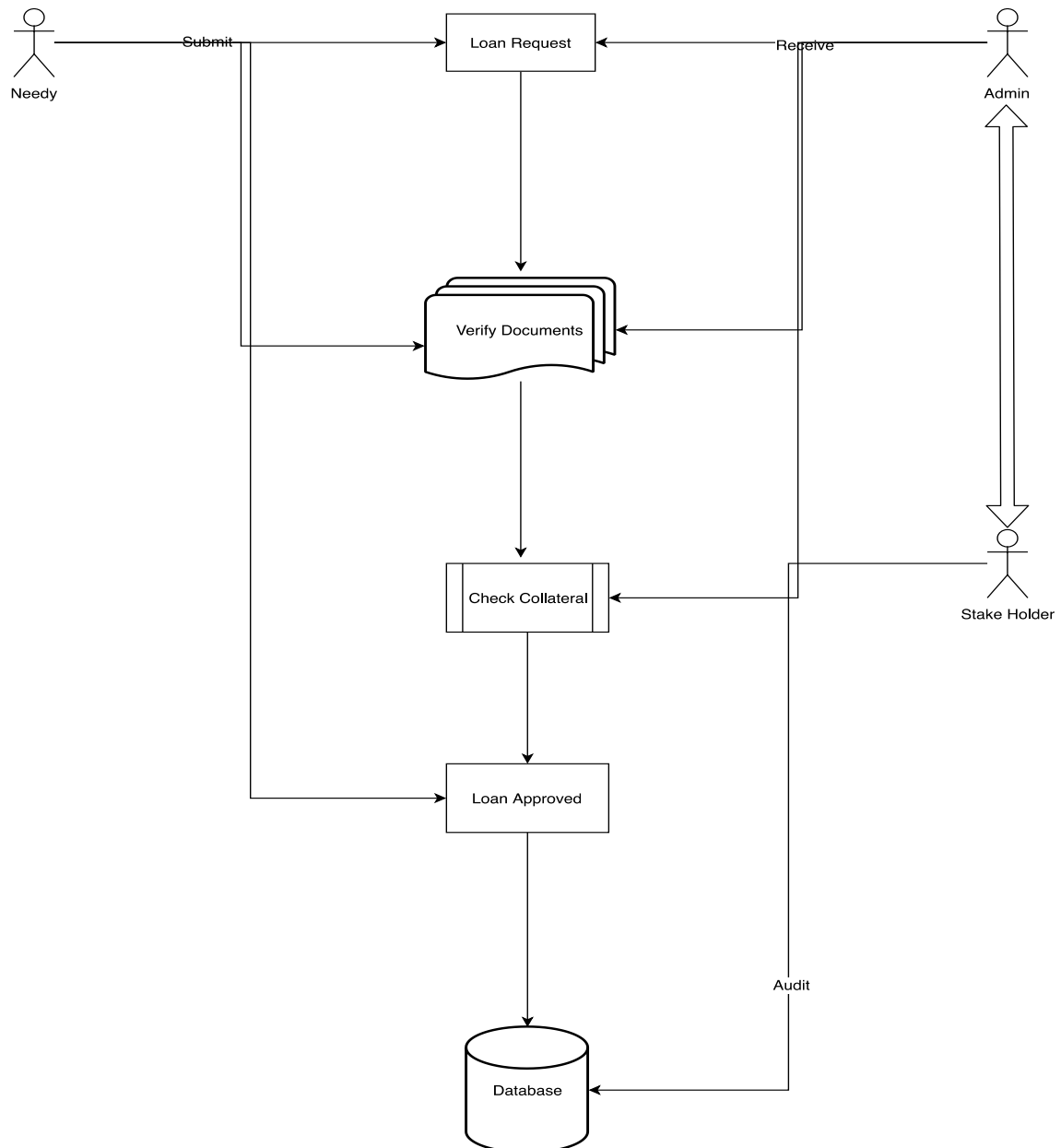
Additionally, if the stake holders had approved a non-collateral loan – who should be held responsible for the default.

Fund Withdrawal:

This is the last use case which defines how the stake holders may withdraw funds from the institution. However, the institution may promote various methodology to hold more funds for a longer period within the institution.

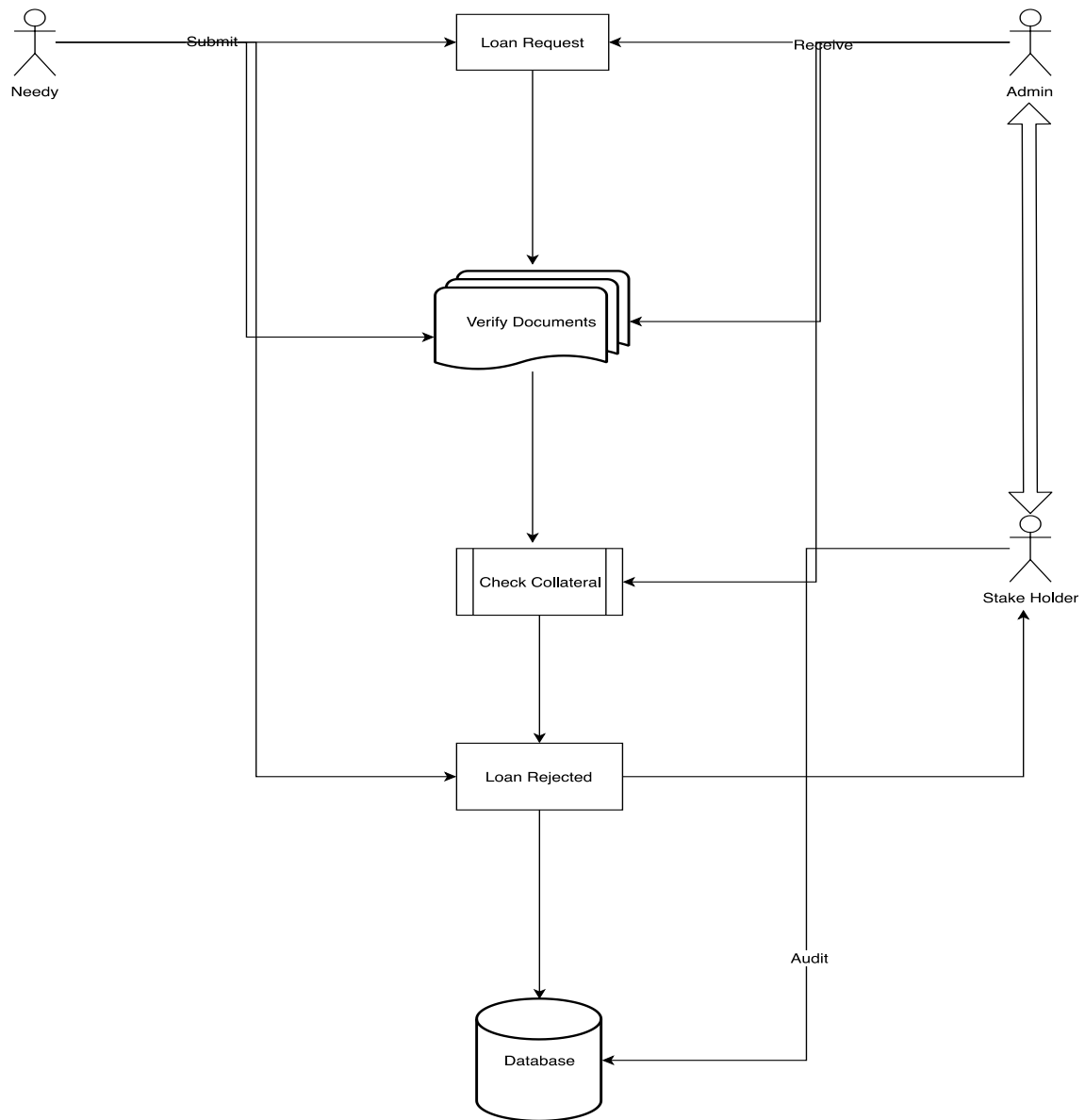
Decentralized Loan System

Case 1: Loan Approved



Decentralized Loan System

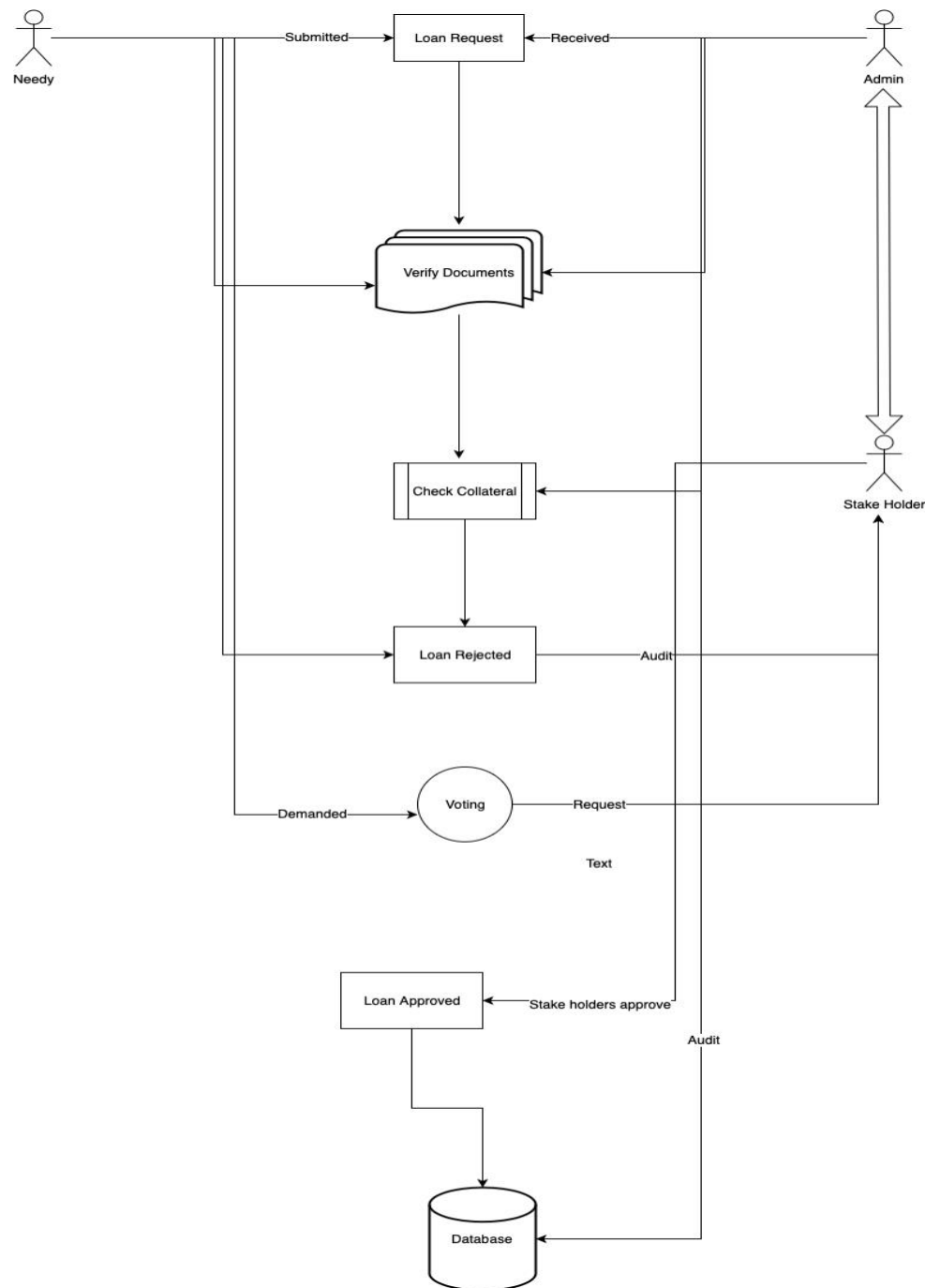
Case 2: Loan Rejected



Decentralized Loan System

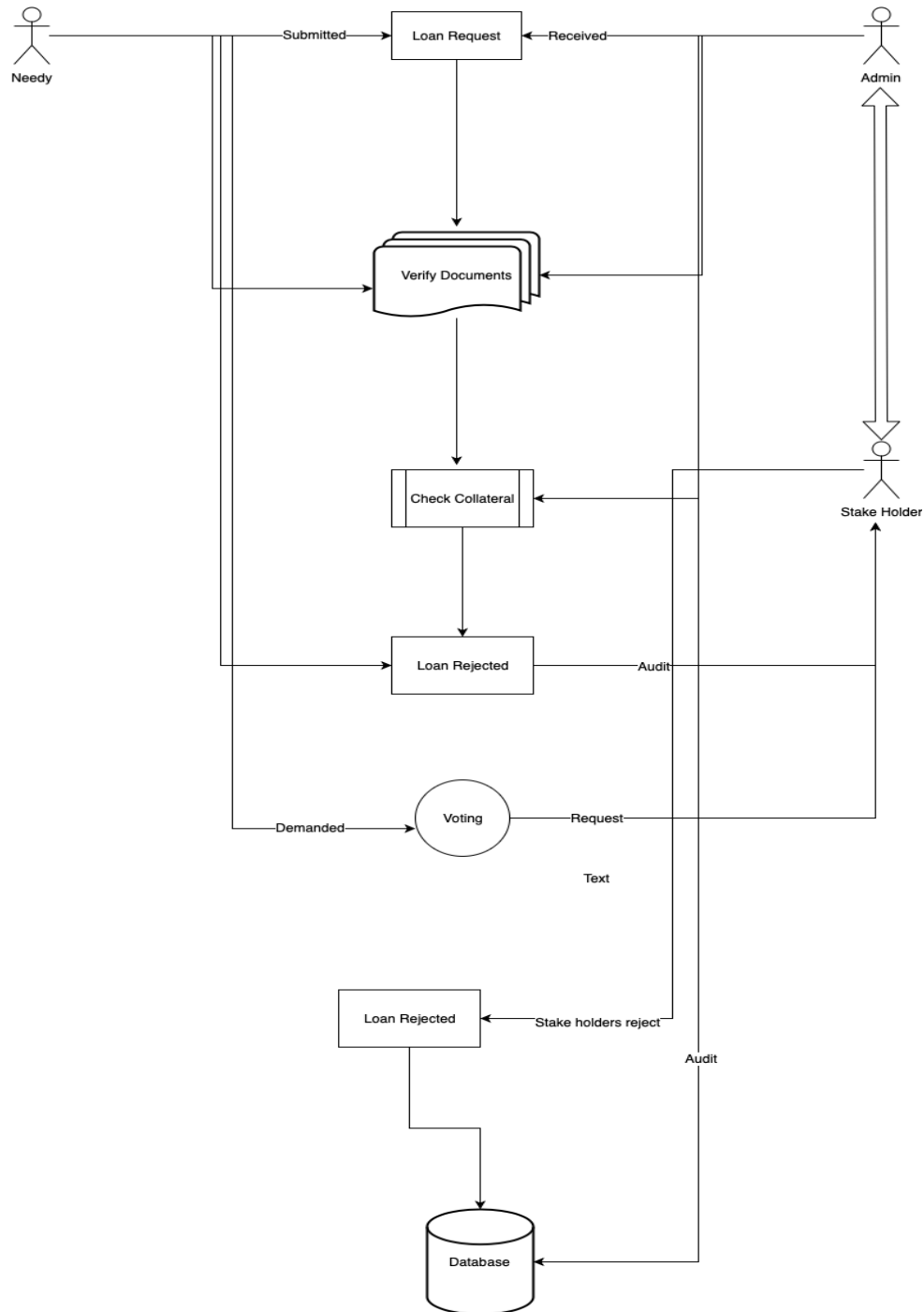
Case 3: Loan Rejected – Voting Demanded

Voting Accepted – Loan Approved



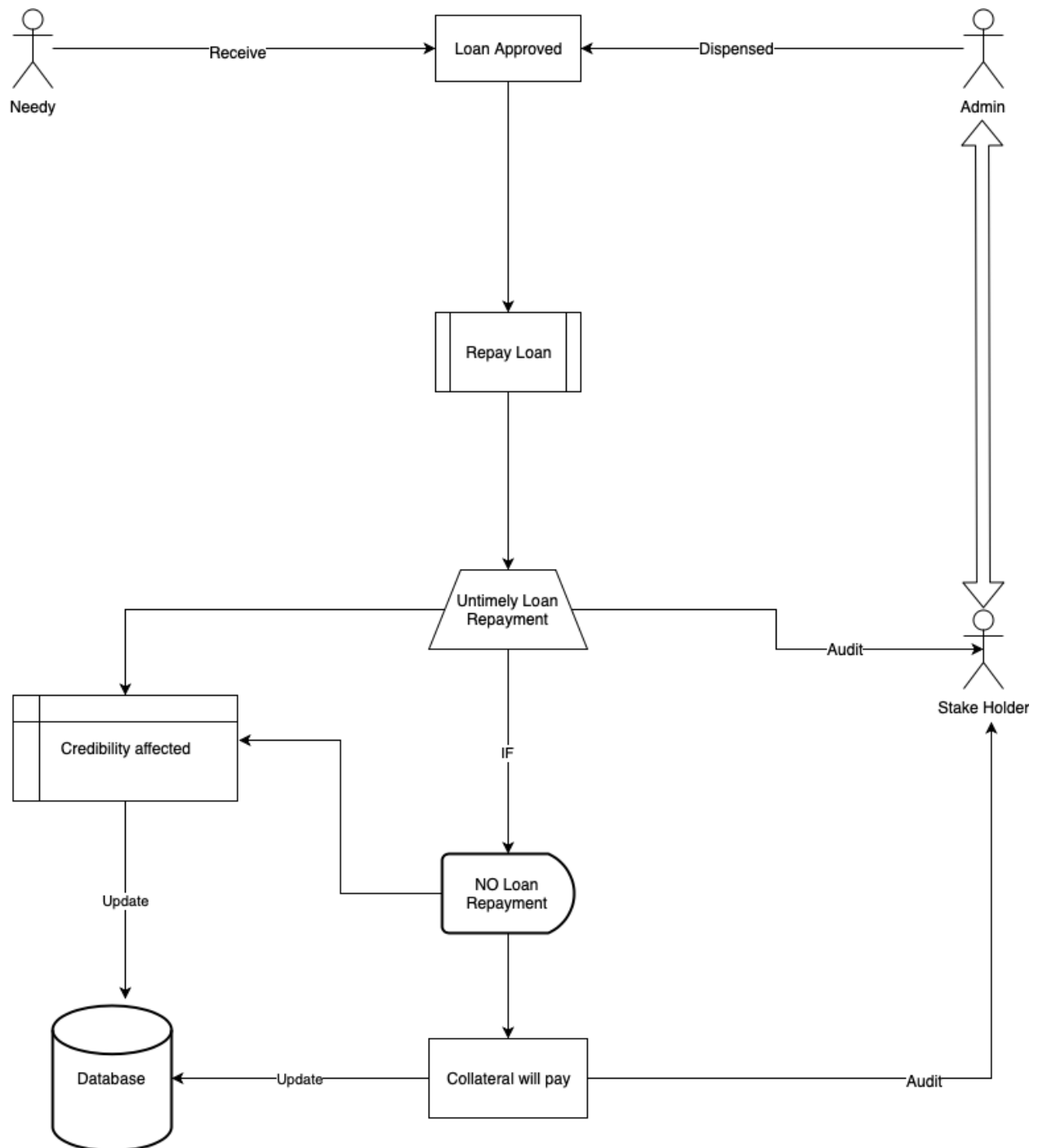
Decentralized Loan System

Voting Accepted – Loan Rejected



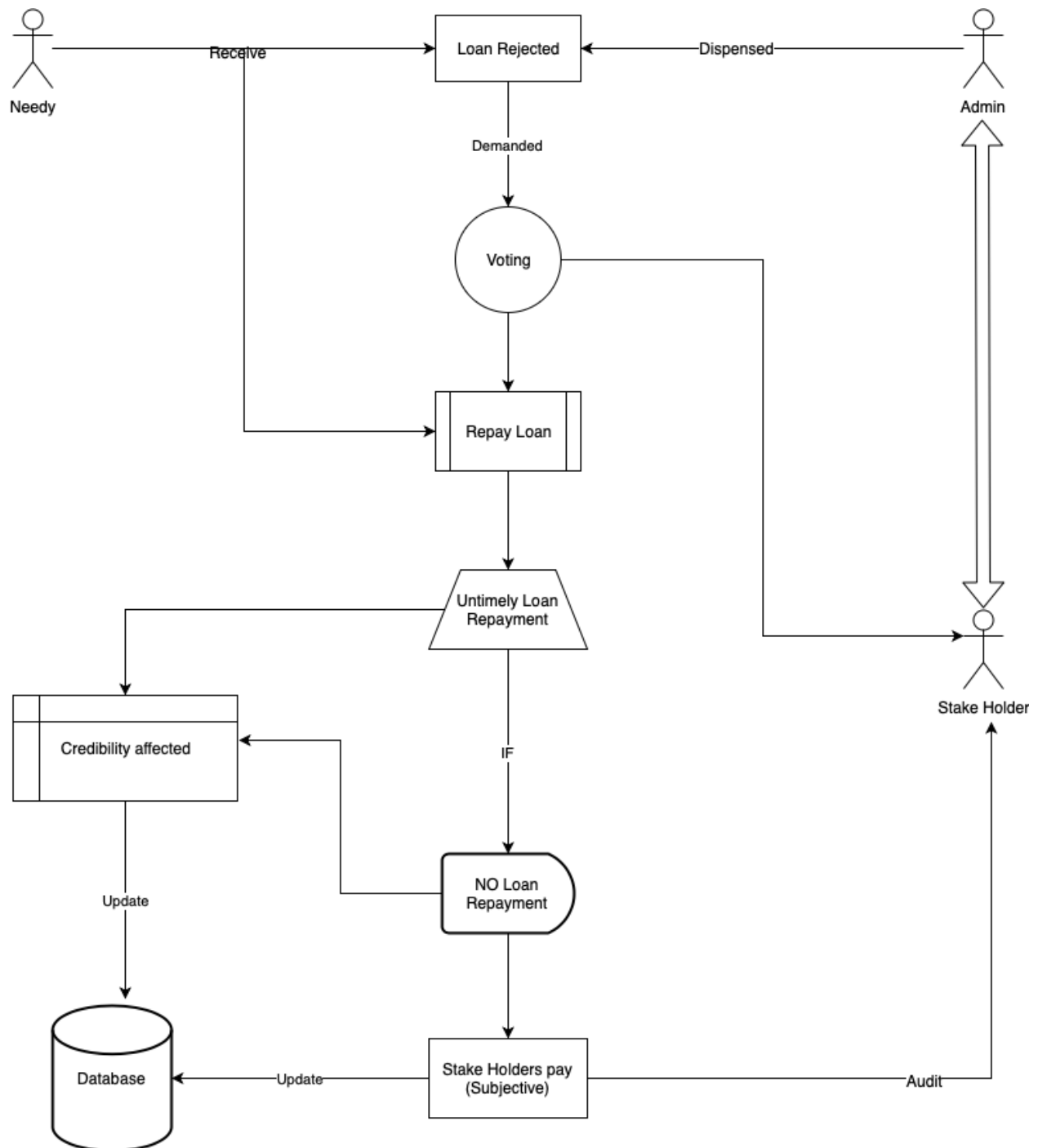
Decentralized Loan System

Case 4: Loan Defaulted



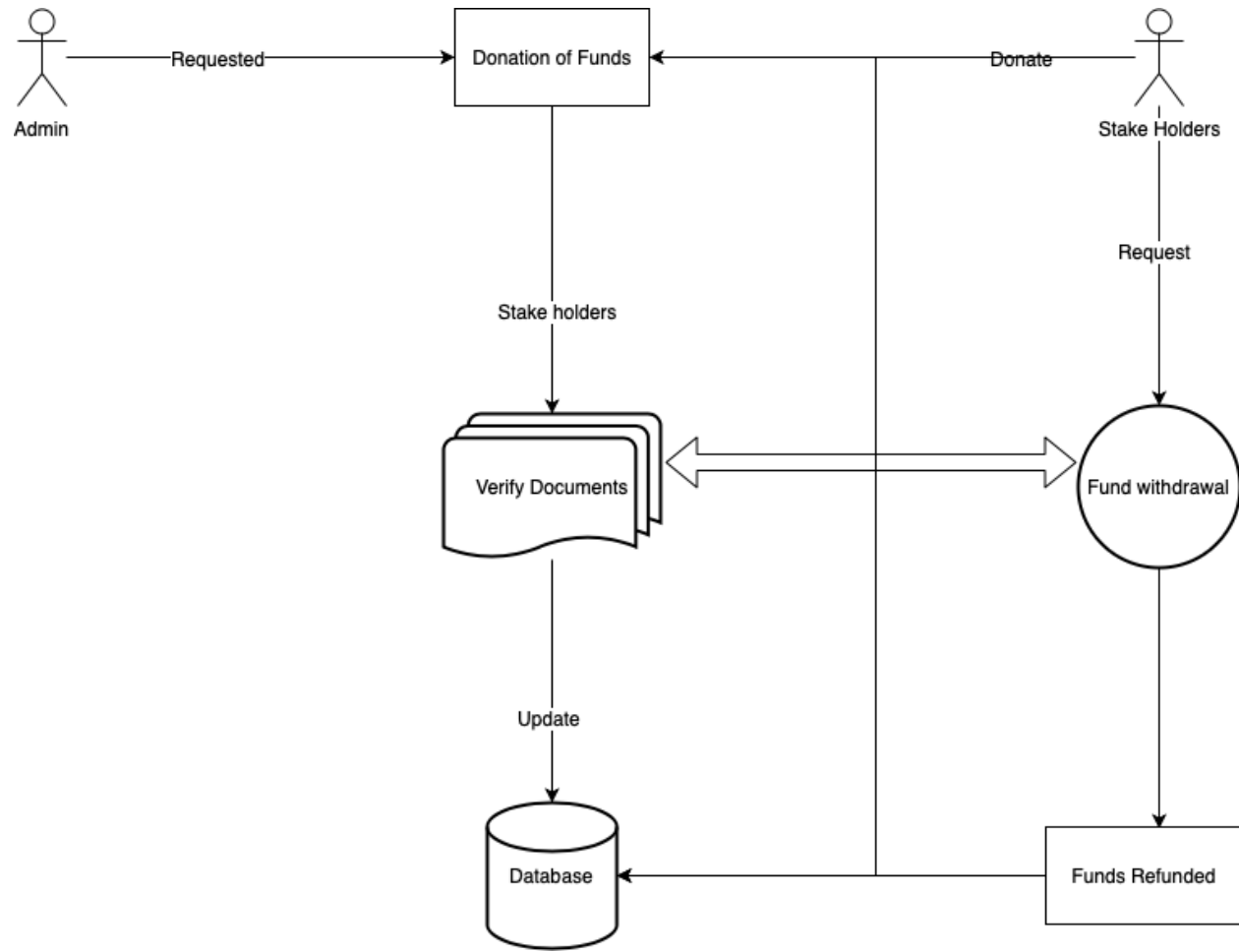
Decentralized Loan System

Case 5: Loan Defaulted after voting



Decentralized Loan System

Case 6: Donation / Withdrawal of funds



Decentralized Loan System

8. Software Codes

Main Code – Decentralized Loan System

```
1  pragma solidity ^0.4.0;
2
3  contract driver {
4      uint TimeStart; //time stamp of the block
5      //constructor
6      function driver() public payable {
7          TimeStart=now;
8      }
9
10
11     mapping(uint=>address)usr_social;
12
13     //structure of member
14     struct member {
15
16         uint addtime;
17         uint counter;
18
19         address member_address;
20         uint social;
21         //for references of the member
22         address ref_1;
23         address ref_2;
24         address ref_3;
25         address ref_4;
26     }
27
28     //maps address to the member structure
29     mapping (address => member) link;
30
31     uint count=1;
32
33     address var1;
34     address var2;
35     address var3;
36     address var4;
37
38
39     //trigered when member is added
40     event SomeoneTriedToAddSomeone(address personWhoTried,address personWhoWasAdded);
41     //trigered when money is deposited
42     event SomeoneAddedMoneyToThePool(address personWhoSent,uint moneyHeSent);
43     //trigered when requested for loan
44     event SomeoneRequestedForMoney(address personWhoRequested,uint requestedM);
45
46     //resets counter for new member
47     function onlynew(address newadd){
```

Decentralized Loan System

```
main.sol
47 function onlynew(address newadd){
48
49     if(link[newadd].ref_1==0x0)
50         count=1;
51
52 }
53
54 uint currttime;
55
56 //check eligibility of member for payments
57 modifier check_eligibility_of_payments(address _check_address) {
58
59     if(link[_check_address].counter < 4){
60         throw;
61     }
62     else{
63         _;
64     }
65 }
66
67
68 uint init_member_counter = 1;
69
70 //assigns initial members
71 function init_members(uint social) {
72     usr_social[social]=msg.sender;
73     if(init_member_counter < 5){
74         link[msg.sender]=member(now,4,msg.sender,social,0x1,0x2,0x3,0x4);
75         init_member_counter++;
76     }
77     else{
78         throw;
79     }
80 }
81
82
83 //validates new member by references
84 function add_Member(address _req_member,uint __aadhaar) check_eligibility_of_payments(msg.sender) {
85
86     onlynew(_req_member);
87
88     if(count==1)
89     {usr_social[__social]=_req_member;
90     var1=msg.sender;
91     link[_req_member]=member(now,count,_req_member,__social,var1,0,0,0);
92     }
93     else if (count==2)
```

```
main.sol
93     else if (count==2)
94     {
95         link[_req_member].ref_2=msg.sender;
96     }
97     else if (count==3)
98     {
99         link[_req_member].ref_3=msg.sender;
100     }
101
102     else if (count==4)
103     {
104         link[_req_member].ref_4=msg.sender;
105     }
106
107     count++;
108
109     SomeoneTriedToAddSomeone(msg.sender,_req_member);
110 }
111
112 //show social references of a member
113 function list_references(address _master_address) constant returns (uint,uint,uint,uint) {
114
115     return (link[link[_master_address].ref_1].social,link[link[_master_address].ref_2].social,link[link[_master_address].ref_3].social,link[link[_master_address].ref_4].social);
116 }
117
118 //shows the money in the pool
119 function getPoolMoney() constant returns (uint){
120
121     return this.balance;
122 }
123
124 //deposit money in the pool
125 function pool(uint __amount) payable {
126
127     this.transfer(__amount);
128     SomeoneAddedMoneyToThePool(msg.sender,__amount);
129 }
130
131 uint[] public amounts;
132 //requested money mapped to member address
133 mapping (uint => address) amount_map;
134
135 modifier onlyafter6()
```

Decentralized Loan System

```
138
139
140 modifier onlyafter6()
141 {
142     uint memtime=link[msg.sender].addtime;
143     if(memtime==0)
144     {
145         throw;
146     }
147     uint nowtime=now;
148     uint _days=(nowtime-memtime)/(24*60*60);
149     if(_days >= 180)
150     {
151         _;
152     }
153     else
154     {
155         throw;
156     }
157 }
158
159 //Checks if the member is valid
160 modifier onlymember()
161 {
162     uint memcount=link[msg.sender].counter;
163     if(memcount >= 4)
164     {
165         _;
166     }
167     else
168     {
169         throw;
170     }
171 }
172 //To request money from the pool
173 function req_Money(uint _amount_) onlymember {
174     amounts.push(_amount_);
175     amount_map[_amount_] = msg.sender;
176     SomeoneRequestedForMoney(msg.sender,_amount_);
177 }
178
179 uint temp;
180
181 function bubble_sort(){
182
```

```
182
183 function bubble_sort(){
184     for(uint j=0;j<amounts.length-1;j++){
185         for(uint k=0;k<amounts.length-j-1;k++){
186             if(amounts[k]>amounts[k+1]){
187                 temp = amounts[k];
188                 amounts[k] = amounts[k+1];
189                 amounts[k+1] = temp;
190             }
191         }
192     }
193 }
194
195 uint sum;
196
197 uint t;
198
199 uint counter_sum=0;
200 //Total distributable money from the pool
201 function assign_loan_amount_from_pool() constant returns (uint){
202     sum = 0;
203     for(t=0;t<amounts.length;t++){
204         if(sum<=amounts[t]){
205             sum=sum+amounts[t];
206             counter_sum = t;
207         }
208     }
209     return sum;
210 }
211
212 function check_time(address ad1) constant returns(uint)
213 {
214     return(link[ad1].addtime);
215 }
216
217 //Address of members who will receive loan
218
```

Decentralized Loan System

```
main.sol
229 function displayAllowedForLoan() constant returns(address[]){
230
231     uint length = amounts.length;
232     address[] memory addr = new address[](length);
233
234     for(uint q=0; q <= counter_sum; q++){
235         addr[q] = amount_map[amounts[q]];
236     }
237
238     return addr ;
239 }
240
241
242
243 address temp_address;
244 //Check if the month is end of three months cycle
245 modifier every_3_months {
246
247     uint months=(now-TimeStart)/(24*60*60*30);
248     if(months%3==0)
249     {
250         _;
251     }
252     else
253     {
254         throw;
255     }
256 }
257
258 //Pay the members the requested loan amount
259 function pay_loan() every_3_months {
260
261     for(uint w=0; w <= counter_sum; w++){
262
263         temp_address = amount_map[amounts[w]];
264         temp_address.transfer(amounts[w]);
265     }
266 }
267
268
269 function getcurtime() constant returns(uint)
270 {
271     curtime=now;
272     return curtime;
273 }
274
275
```

Decentralized Loan System

Mortgage Code (Collateral) – Decentralized Loan System

```
mortgage.sol
1 pragma solidity ^0.4.4;
2
3 contract Mortgage{
4
5     /* This is the constructor which will deploy the contract on the blockchain.
6     We will initialize with the loan status as 'Initiated' and for test purposes
7     we will initialize the loan applicant's balance to 1000000. Note since solidity
8     does not support float or double at this point, we will store the actual value
9     multiplied by 100 in the contract, and will be divide the value retrieved from
10    the contract by 100, when we have to represent the balance in the UI
11    */
12    function Mortgage()
13    {
14        loanApplicant = msg.sender;
15        loan.status = STATUS_INITIATED;
16        balances[msg.sender] = 100000000;
17    }
18
19    /* address of the loan applicant */
20    address loanApplicant;
21
22    // Events - publicize actions to external listeners
23    event LienReleased(address _owner);
24    event LienTrasferred (address _owner);
25    event LoanStatus (int _status);
26
27    int constant STATUS_INITIATED = 0;
28    int constant STATUS_SUBMITTED = 1;
29    int constant STATUS_APPROVED = 2;
30    int constant STATUS_REJECTED = 3;
31
32
33    /* struct datatype to store the property details */
34    struct Property {
35        bytes32 addressOfProperty;
36        uint32 purchasePrice;
37        address owner;
38    }
39
40    /* struct datatype to store the loan terms */
41    struct LoanTerms{
42        uint32 term;
43        uint32 interest;
44        uint32 loanAmount;
45        uint32 annualTax;
46        uint32 annualInsurance;
47    }
```

Decentralized Loan System

```
mortgage.sol
49  /* struct datatype to store the monthly payment structure */
50  struct MonthlyPayment{
51      uint32 pi;
52      uint32 tax;
53      uint32 insurance;
54  }
55
56  /* struct datatype to store the details of the loan contract */
57  struct Loan {
58      LoanTerms loanTerms;
59      Property property;
60      MonthlyPayment monthlyPayment;
61      ActorAccounts actorAccounts;
62      int status; // values: SUBMITTED, APPROVED, REJECTED
63  }
64
65  struct ActorAccounts {
66      address mortgageHolder;
67      address insurer;
68      address irs;
69  }
70
71  Loan loan;
72  LoanTerms loanTerms;
73  Property property;
74  MonthlyPayment monthlyPayment;
75  ActorAccounts actorAccounts;
76
77  /* mapping is equivalent to an associate array or hash
78  Maps addresses of the actors in the mortgage contract with their balances
79  */
80  mapping (address => uint256) public balances;
81
82  /* This means that if the mortgage holder calls this function, the
83  function is executed and otherwise, an exception is thrown */
84  modifier bankOnly {
85      if(msg.sender != loan.actorAccounts.mortgageHolder) {
86          throw;
87      }
88      _;
89  }
90
91  /* deposit into actor accounts and will return the balance of the user
92  after the deposit is made */
93  function deposit(address receiver, uint amount) returns(uint256) {
94      if (balances[msg.sender] < amount) return;
95      balances[msg.sender] += amount;
```

```
mortgage.sol
100
101  /* 'constant' prevents function from editing state variables; */
102  function getBalance(address receiver) constant returns(uint256){
103      return balances[receiver];
104  }
105
106  /* check if mortgage payment if complete, if complete, then release the property
107  lien to the homeowner */
108  function checkMortgagePayoff(){
109      if(balances[loan.actorAccounts.mortgageHolder]
110         == loan.monthlyPayment.pi*12*loan.loanTerms.term &&
111         balances[loan.actorAccounts.insurer]
112         == loan.monthlyPayment.tax*12*loan.loanTerms.term &&
113         balances[loan.actorAccounts.irs]
114         == loan.monthlyPayment.insurance*12*loan.loanTerms.term
115      ){
116          loan.property.owner = loanApplicant;
117          LienReleased(loan.property.owner);
118      }
119  }
120
121
122  /* Add loan details into the contract */
123  function submitLoan(
124      bytes32 _addressOfProperty,
125      uint32 _purchasePrice,
126      uint32 _term,
127      uint32 _interest,
128      uint32 _loanAmount,
129      uint32 _annualTax,
130      uint32 _annualInsurance,
131      uint32 _monthlyPi,
132      uint32 _monthlyTax,
133      uint32 _monthlyInsurance,
134      address _mortgageHolder,
135      address _insurer,
136      address _irs
137  ){
138      loan.property.addressOfProperty = _addressOfProperty;
139      loan.property.purchasePrice = _purchasePrice;
140      loan.loanTerms.term=_term;
141      loan.loanTerms.interest=_interest;
142      loan.loanTerms.loanAmount=_loanAmount;
143      loan.loanTerms.annualTax=_annualTax;
144      loan.loanTerms.annualInsurance=_annualInsurance;
145      loan.monthlyPayment.pi=_monthlyPi;
146      loan.monthlyPayment.tax=_monthlyTax;
```

Decentralized Loan System

```
mortgage.sol
148     loan.actorAccounts.mortgageHolder = _mortgageHolder;
149     loan.actorAccounts.insurer = _insurer;
150     loan.actorAccounts.irs = _irs;
151     loan.status = STATUS_SUBMITTED;
152 }
153
154 /* Gets loan details from the contract */
155 function getLoanData() constant returns (
156     bytes32 _addressOfProperty,
157     uint32 _purchasePrice,
158     uint32 _term,
159     uint32 _interest,
160     uint32 _loanAmount,
161     uint32 _annualTax,
162     uint32 _annualInsurance,
163     int _status,
164     uint32 _monthlyPi,
165     uint32 _monthlyTax,
166     uint32 _monthlyInsurance)
167 {
168     _addressOfProperty = loan.property.addressOfProperty;
169     _purchasePrice = loan.property.purchasePrice;
170     _term = loan.loanTerms.term;
171     _interest = loan.loanTerms.interest;
172     _loanAmount = loan.loanTerms.loanAmount;
173     _annualTax = loan.loanTerms.annualTax;
174     _annualInsurance = loan.loanTerms.annualInsurance;
175     _monthlyPi = loan.monthlyPayment.pi;
176     _monthlyTax = loan.monthlyPayment.tax;
177     _monthlyInsurance = loan.monthlyPayment.insurance;
178     _status = loan.status;
179 }
180
181 /* Approve or reject loan */
182 function approveRejectLoan(int _status) bankOnly {
183     //if(msg.sender == loanApplicant) throw;
184     loan.status = _status;
185     /* if status is approved, transfer the lien of the property
186     to the mortgage holder */
187     if(_status == STATUS_APPROVED)
188     {
189         loan.property.owner = msg.sender;
190         LienTrasferred(loan.property.owner);
191     }
192     LoanStatus(loan.status);
193 }
194 }
```

Line 194, Column 2 Spaces: 4 Solidity

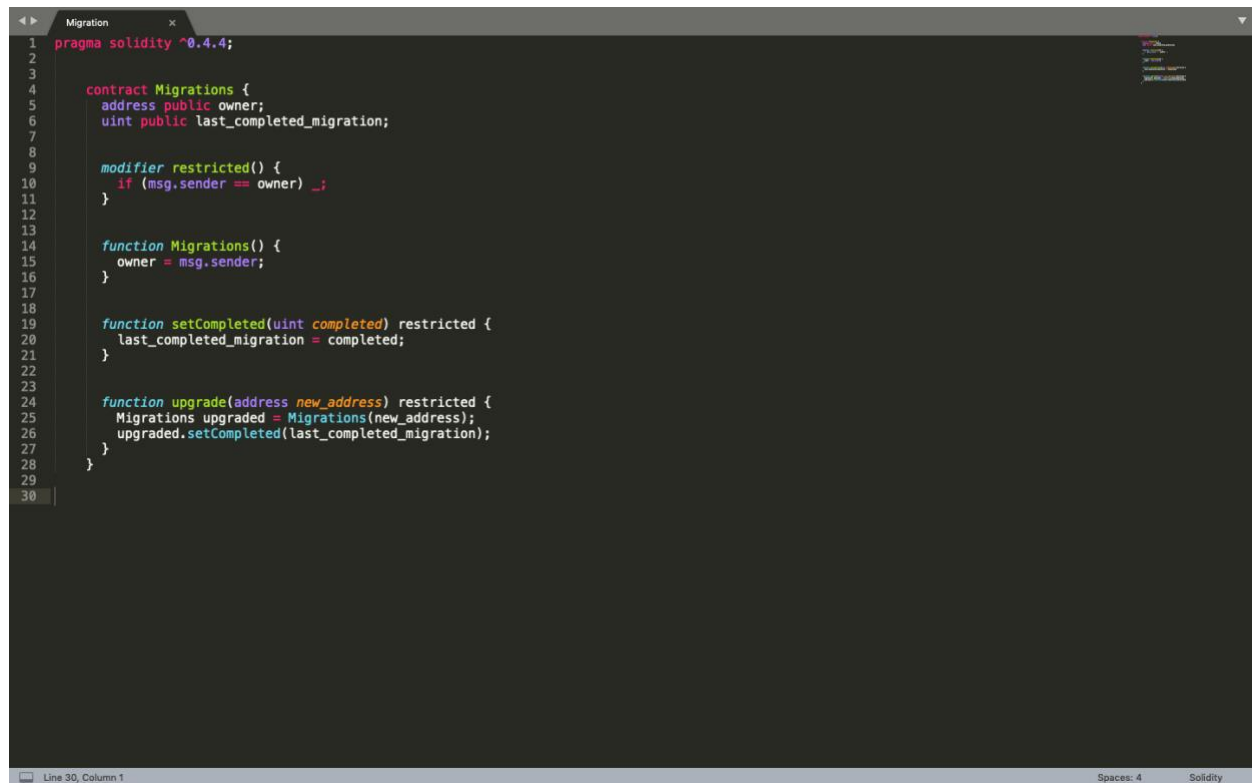
Decentralized Loan System

Voting Code – Decentralized Loan System

```
1 pragma solidity ^0.4.2;
2
3
4 contract Election {
5     // Model a Candidate
6     struct Candidate {
7         uint id;
8         string name;
9         uint voteCount;
10    }
11
12
13    // Store accounts that have voted
14    mapping(address => bool) public voters;
15    // Store Candidates
16    // Fetch Candidate
17    mapping(uint => Candidate) public candidates;
18    // Store Candidates Count
19    uint public candidatesCount;
20
21
22    // voted event
23    event votedEvent (
24        uint indexed _candidateId
25    );
26
27
28    function Election () public {
29        addCandidate("Candidate 1");
30        addCandidate("Candidate 2");
31    }
32
33
34    function addCandidate (string _name) private {
35        candidatesCount ++;
36        candidates[candidatesCount] = Candidate(candidatesCount, _name, 0);
37    }
38
39
40    function vote (uint _candidateId) public {
41        // require that they haven't voted before
42        require(!voters[msg.sender]);
43
44
45        // require a valid candidate
46        require(_candidateId > 0 && _candidateId <= candidatesCount);
47    }
```

Decentralized Loan System

Migration Code – Decentralized Loan System



```
1 pragma solidity ^0.4.4;
2
3
4 contract Migrations {
5     address public owner;
6     uint public last_completed_migration;
7
8
9     modifier restricted() {
10         if (msg.sender == owner) _;
11     }
12
13     function Migrations() {
14         owner = msg.sender;
15     }
16
17     function setCompleted(uint completed) restricted {
18         last_completed_migration = completed;
19     }
20
21     function upgrade(address new_address) restricted {
22         Migrations upgraded = Migrations(new_address);
23         upgraded.setCompleted(last_completed_migration);
24     }
25 }
26
27
28
29
30
```

The screenshot shows a Solidity code editor with a dark theme. The code defines a contract named 'Migrations'. It includes a pragma statement for Solidity version ^0.4.4. The contract has two public variables: 'owner' of type 'address' and 'last_completed_migration' of type 'uint'. It features a 'restricted' modifier that checks if the sender is the owner. The contract has four functions: a constructor 'Migrations()' that sets the owner to the sender; 'setCompleted(uint completed)' which updates the last completed migration; and 'upgrade(address new_address)' which creates a new instance of the contract at a new address and transfers the last completed migration state. The editor interface includes a file explorer on the right, a line number column on the left, and a status bar at the bottom indicating 'Line 30, Column 1', 'Spaces: 4', and 'Solidity'.

Decentralized Loan System

Future Scope

There are three improvements which can be made in the near future which will facilitate more and more participants to take benefit of the system.

1. Increase the trust between stake holders so that more and more participants contribute money into the pool.

2. Stake holders should be incentivized so that they keep money for much longer period of time.

3. Formation of a central pool depending upon geographical location and or other parameters.

Consider a scenario which may be implemented in future.

If person A – donates money in the pool recursively over a long period of time. At the end of the tenure the amount is now \$1,00,000. Now person A requires the said amount for his child (B's) education. However instead of withdrawing the money from the pool – person A takes a loan of the said amount from organization. Now, the said amount of person A acts like a collateral for the loan. However, while the loan is not completely repaid – person A does not enjoy the right to vote. After B completes his study – he can repay the loan as defined by the guidelines of the institution. After the completion of the tenure – the lock upon A's amount is withdrawn and A can enjoy his amount.

Hence, at the end – B is able to pursue education without utilizing his father's money.

This example can be applied to number of different scenarios. Loan against the existing amount within the system may serve to be a great way of retaining funds and utilizing the money.

Decentralized Loan System

The central pool mentioned above might work in the following way:

The institutions within a geographical location may maintain a pool of funds. The members of this central pool are mandated to lend funds to the central pool. This central pool might be useful if – another institution within the pool may lack money. The funds lent to the central pool may differ from institution to institution.

Different parameters of fund lending may be:

- a. The ratio of funds lent by elite to the funds being allocated to the needy.
- b. The duration for which the funds are being held by the institution.

The central pool may allocate funds depending upon the priority – for e.g.: an institution within the central pool which has to repay a stake holder might be given priority over an institution which just needs funds for maintaining its minimum fund balance.

The stake holders for the central pool maybe admins from the member institution and also elite members – depending upon their score.

Decentralized Loan System

Conclusion

The paper describes the problem statement and working of existing system. Then the paper lays down the limitations and area of improvement within the system. Based on these limitations – the paper proposes a solution with the help of class diagrams, use cases and software code. In the end, the paper describes the future scope of the project.

Also, each of the modules described in the paper – for e.g.: Paperless system, Trustless authentication, Unbiased decisions and Transparent function can be implemented as individual system or can be integrated with other existing system.

Area of Improvement

- Once stake holders reject the request of loan and/or voting can a need reapply – if yes how and what is the criteria.
- Another major concern is – how can the stake holders be put in a controlled environment so that they cannot abuse their scope of voting or decision making. For e.g.: If a group of people amongst the stake holders have inclination towards certain individuals and/or organizations they might approve/waive the loan without adequate inquiry or collateral. This expression of personal interest might make the authenticity of loan process dubious.
- Once a loan has been defaulted (non-collateral) and or waived off who is responsible for the repayment of the loan amount. Stake holders might be held accountable for these activities.

Decentralized Loan System

Solution (Future Steps)

- The institution also has the facility of donation within the system. This amount can be exclusively used for repaying loan defaults and loan waivers. However, it should be considered that such activities might promote self-interest and hence abuse the trust within the institution.
- A score is maintained within the institution which defines the credit worthiness of an individual. This score might be directly affected in case a loan is defaulted or installment is missed. Also, the stake holders may be incentivized and penalized for good voting and bad voting respectively.

Additionally, once a stake holder votes – his voting capacity might be restricted.

- The institution can promote methodologies to retain the funds within the institution for a longer period of time. This can be achieved by a number of ways which may include – incentivizing stake holders with a good score for maintaining the amount or if the stake holder needs the money – (s)he may be given a loan instead of complete withdrawal of funds.

Decentralized Loan System

Appendix

Definitions

1. Decentralized Loan System

This is a Blockchain based application of the loan system which tries to migrate traditional systems on to Blockchain.

2. Existing System

This typically defines the 'non-profit' organization for which this system was defined.

3. Administrative Department / Admin

They are responsible for allocating loan, managing funds and requesting funds.

4. Loan Borrower / Needy

They are the people who request for the loan from the system.

5. Loan Lender / Elite

They are the people who donate funds into the system.

6. Blockchain v/s Other System

The paper makes it very much clear as to why Blockchain based solution is used as compared to any other technology. Following are the most notable reasons –

- a. Blockchain is immutable – hence cannot be altered or manipulated.
- b. There is on demand transparency – which can be leveraged as and when required.
- c. Stake holders can vote for or against approval of a loan.
- d. Managing of funds is relatively easy.

Decentralized Loan System

e. There is a trust among the stake holders – which attracts more donations.

7. Vote

Stake holders may enjoy the power of voting. However, it has certain restrictions which are defined here.

- a. The power of the vote is directly proportional to the amount of funds currently secured with the system by the stake holder.
- b. The power of the vote is also dependent upon – credibility. If there is a bad loan or delayed repayments – the credibility of the stake holder is affected.
- c. Once a stake holder cast his vote – he cannot re-vote until the said amount has been completely repaid. (Certain exceptions applied.)

Note: The feature of vote may be superseded by the chairman of the committee at any time.

8. Partial Transparency

Although the participants of this system enjoy transparency at all times – it should be noted that personal information which may reveal the identity of the participant will be not be shared with any member. However, a few members of the admin department may be allowed to view the identity of a person. However, it is mandatory on them to not disclose the information.

Decentralized Loan System

Decentralized Loan System – [GitHub](#).

[White Paper](#)

[Audit Smart Contract](#)

[Collateral Smart Contract](#)

[Election Smart Contract](#)

[Loan Smart Contract](#)

Decentralized Loan System

Bibliography

- a. <https://medium.com/trivial-co/lending-and-borrowing-on-the-blockchain-should-banks-be-scared-e0a01c857c43>
- b. <https://hackernoon.com/decentralized-credit-lending-on-the-blockchain-fa988b1ec7b4>
- c. <https://lendoit.com>
- d. <https://github.com/bhaumik-choksi/Loan>
- e. <https://github.com/ripio/rcn-network>
- f. <https://github.com/SatoshiNextTechLab/0xSHG>
- g. <https://github.com/rajivjc/mortgage-blockchain-demo>
- h. <https://github.com/anshulshah96/CrowdBank>
- i. <https://github.com/piy0999/CreditSense>