Microfinance dApp
System Analysis Level 1

MH6808

Blockchain-based System

Requirements Gathering

User Roles / Stakeholders

Investor: Investors push funds to the entire ecosystem through an ICO.

Bank: The Bank holds the rights to a finite supply of the new currency or token that is created and universally agreed on by all stakeholders as a medium for the exchange of goods or services.

Broker: The stakeholder who is connecting the Bank and the Borrower. The Bank and the Borrower communicate through a Broker.

Borrower: Borrowers use the new tokens for their businesses. E.g.: farmers, non-banking people; those who are not familiar with banks.

Vendor: The party that accepts the new tokens and provides services and goods to the Borrower’s business.

Microfinance Operations

1. The Bank creates the finite supply of ERC20 tokens.
2. The Bank publishes loan plans.
3. The Bank identifies Brokers and Borrowers and registers them in the blockchain and assign digital identities to them.
4. The Broker requests a loan provided by the Bank with a specific amount, interest rate, and period on behalf of the Borrower.
5. The Borrower signs the loan contract requested by Broker to the Bank.
6. The Bank approves the loan.
7. The Bank transfers tokens to the Borrower.
8. After the Borrower signs the contract, the Broker gets some commission in the form of tokens from the Bank.
9. To pay back the loan, the Borrower transfers the initially agreed tokens back to the Bank.
10. The Bank marks the loan as “Defaulted” if the Borrower is not able to pay back the borrowed tokens.

User Stories

The Bank

Creates and holds the finite supply of ERC20 tokens.

The Bank publishes loan plans, modifies and changes the availability of plans.

The Bank identifies the Brokers and Borrowers, registers them in the blockchain, and assigns digital identities to them.

The Bank approves or rejects loan requests from the Broker.

After the Borrower signs the loan request,

- the Bank transfers tokens to the Borrower (loan).

- the Bank transfers tokens to the Broker (commission).

Marks loans as Defaulted or not.

The Broker

The Broker requests a loan provided by the Bank with a particular amount, interest rate, and period on behalf of the Borrower.

The Borrower

The Borrower signs the loan contract.

The Borrower transfers the initially agreed tokens to the Bank

Functional Requirements

The Bank

Publish, modify, and change the availability of plans.

Register Brokers and Borrowers on the blockchain.

Transfer tokens to Borrowers.

Transfer tokens to Brokers.

The Broker

Apply for Loans on behalf of the Borrower.

The Borrower

Sign the loan contract.

Transfer tokens to the Vendors.

Transfer tokens to the Bank.

Non-functional Requirements

Confidentiality

Privacy

Availability

performance (latency, throughput)

Modifiability

Usability

immutability,

Non-repudiation,

Integrity

Transparency

Trust

Data privacy and scalability

Cost effective (minimum gas fees)

Data Requirements

User identities.

Loan plan information.

Loan information.

User Interface Requirements

Web UI for Bank users to interact with bank functions (services).

Web UI for Brokers to interact with the bank functions (services).

Web UI for Borrowers to interact with the bank functions. (services).

Smart Contract Requirements

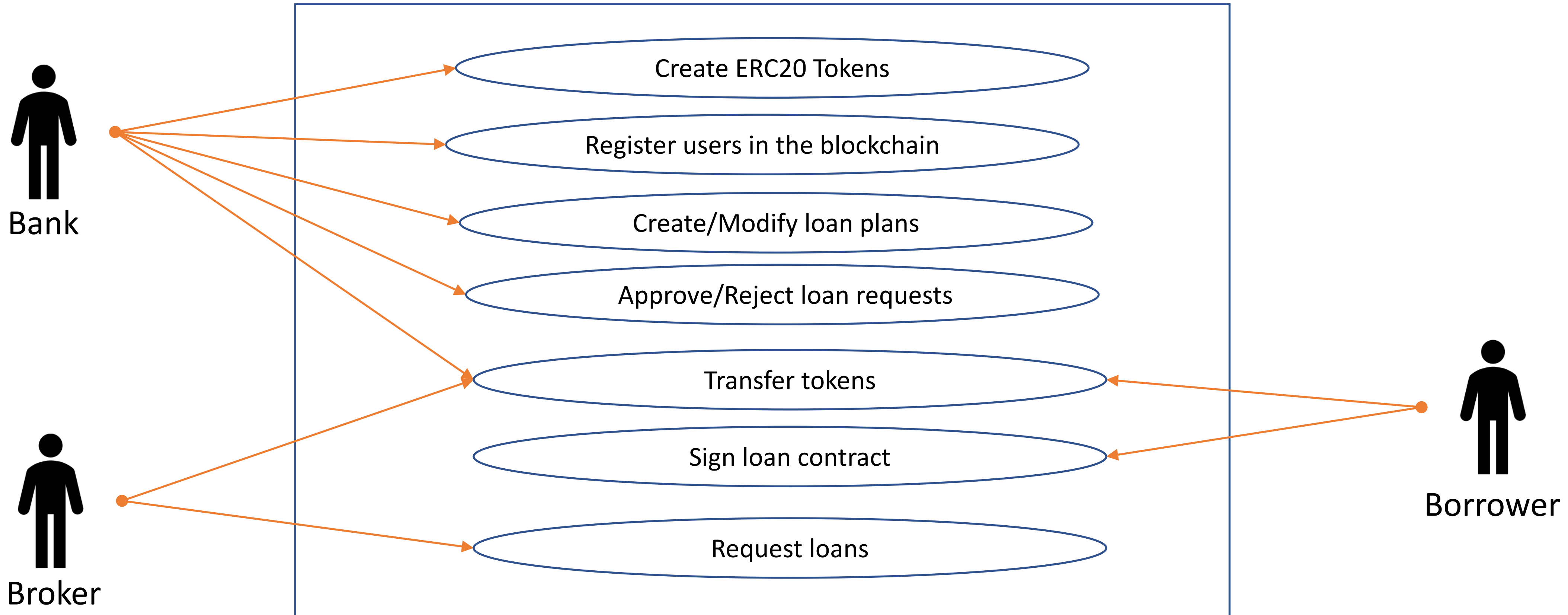
ERC 20 token smart contract.

Need to track user details.

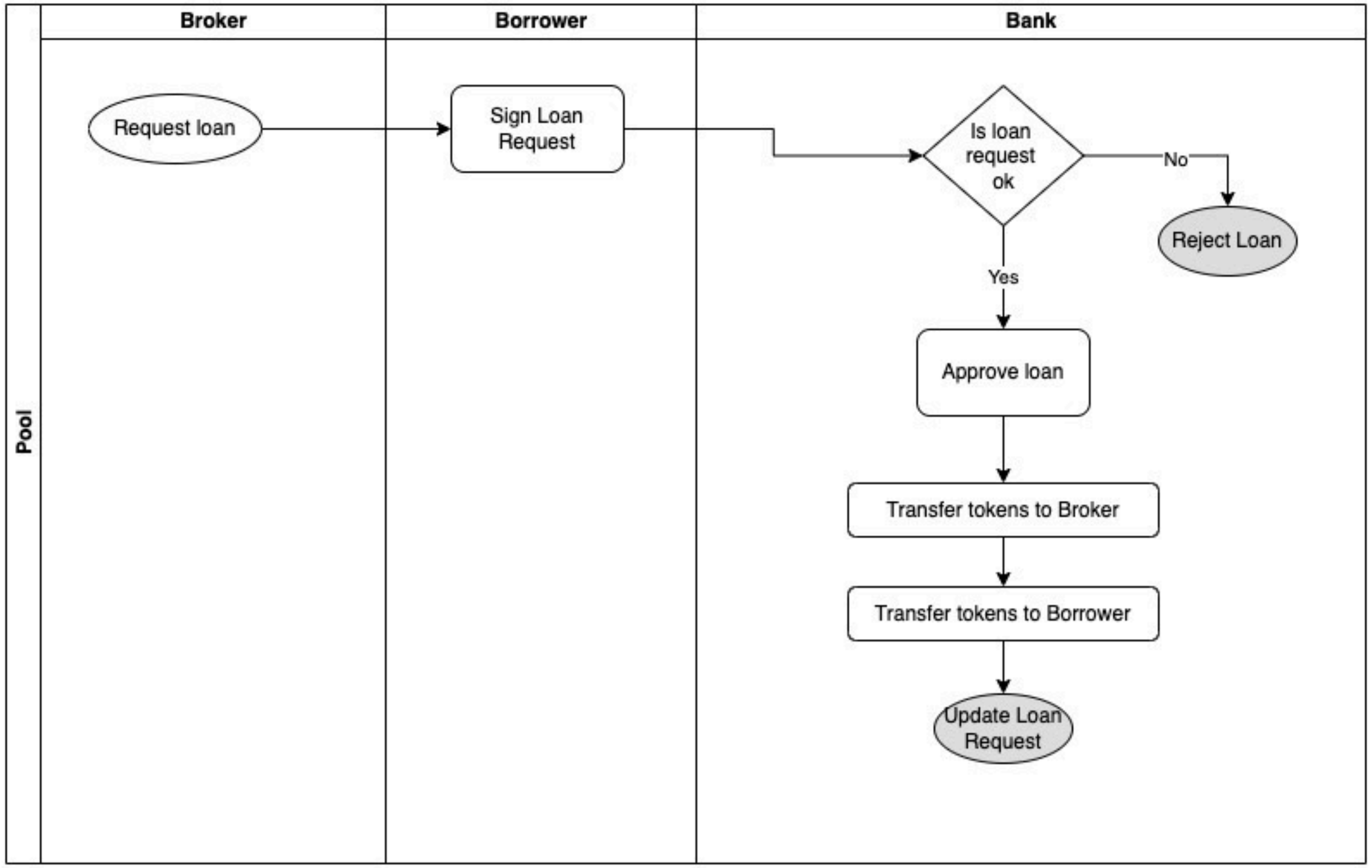
Need to keep track of various stages of a loan.

Requirements Analysis

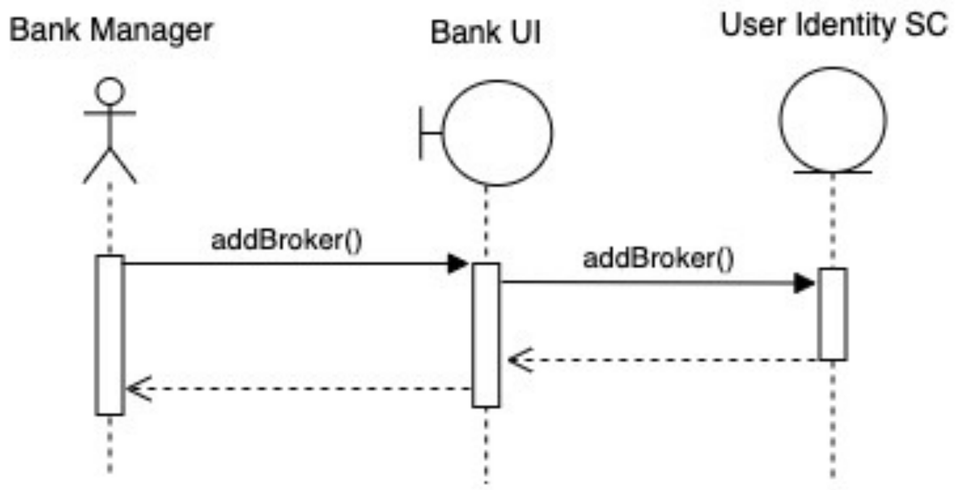
Use Case Diagram

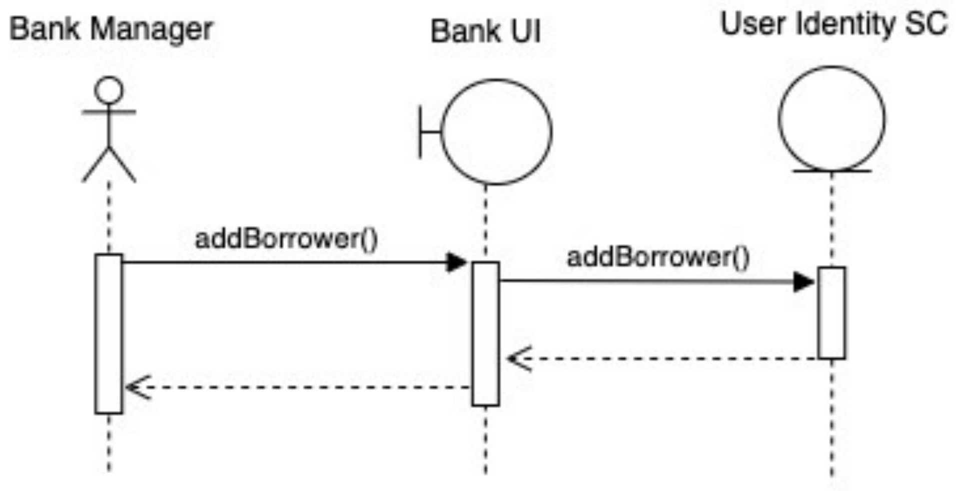


Swimlane Diagram – Get Loan

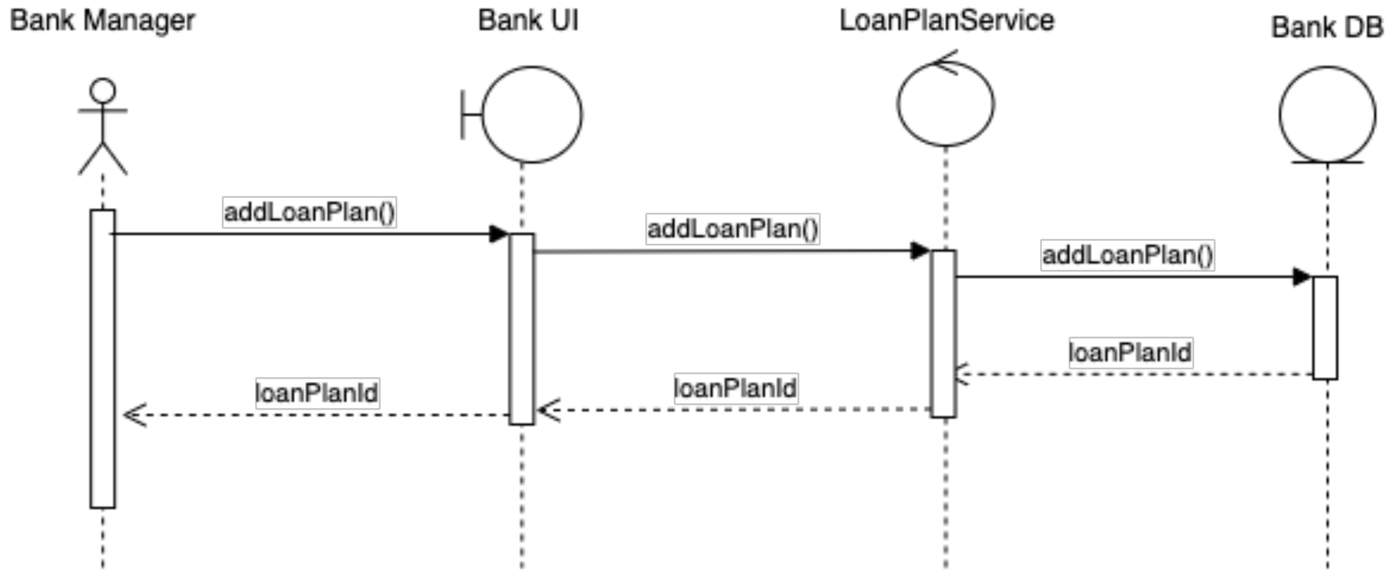


Sequence Diagram – Broker/Borrower Registration

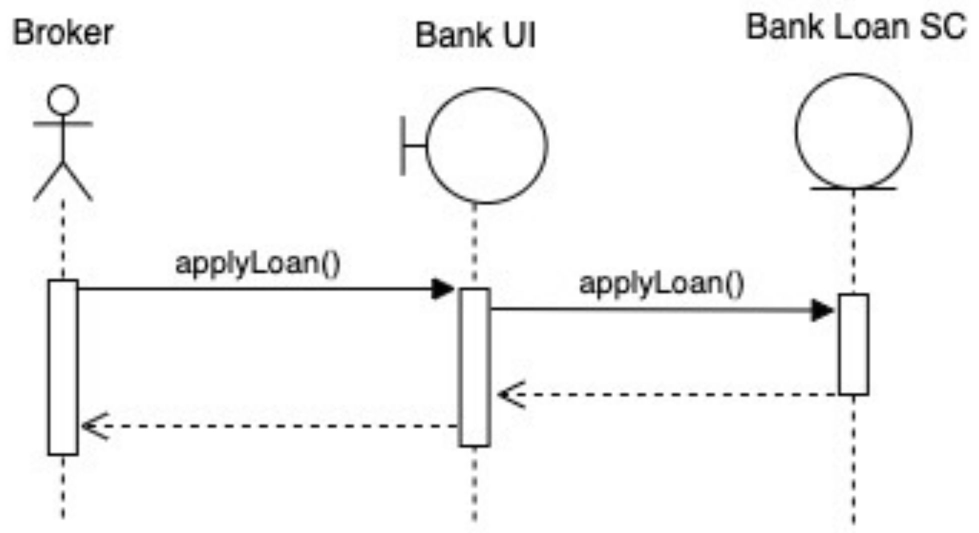




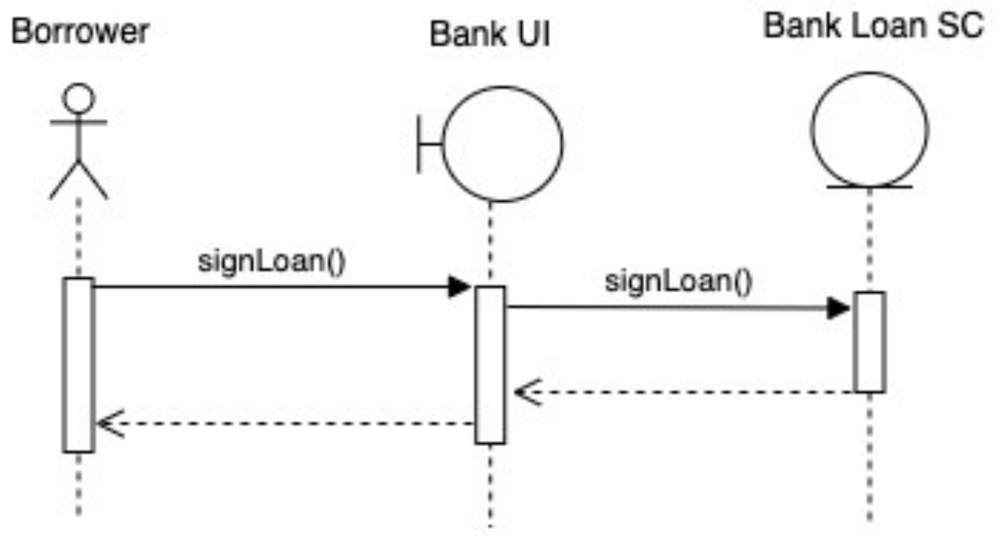
Sequence Diagram – Add Loan Plans



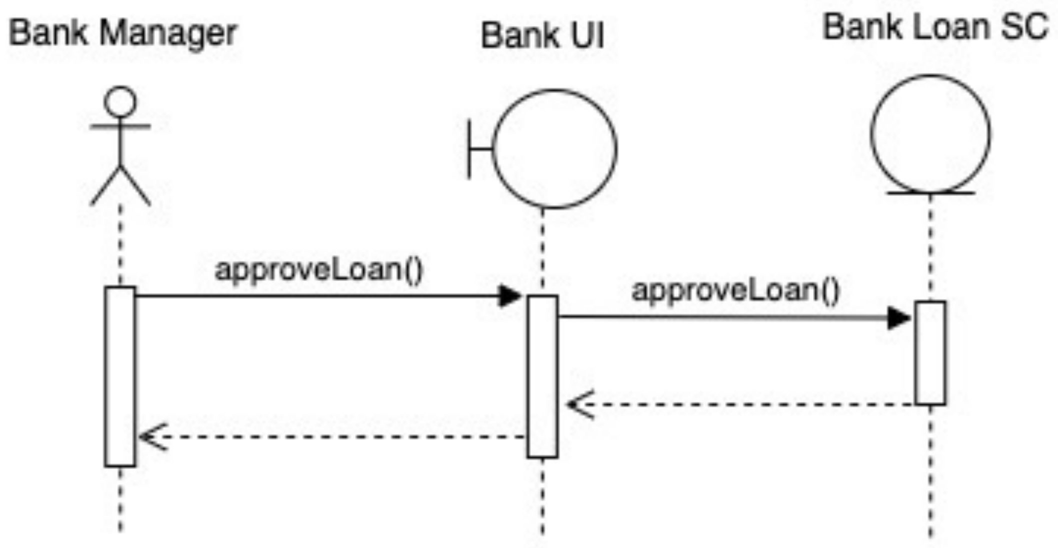
Sequence Diagram – Apply Loan



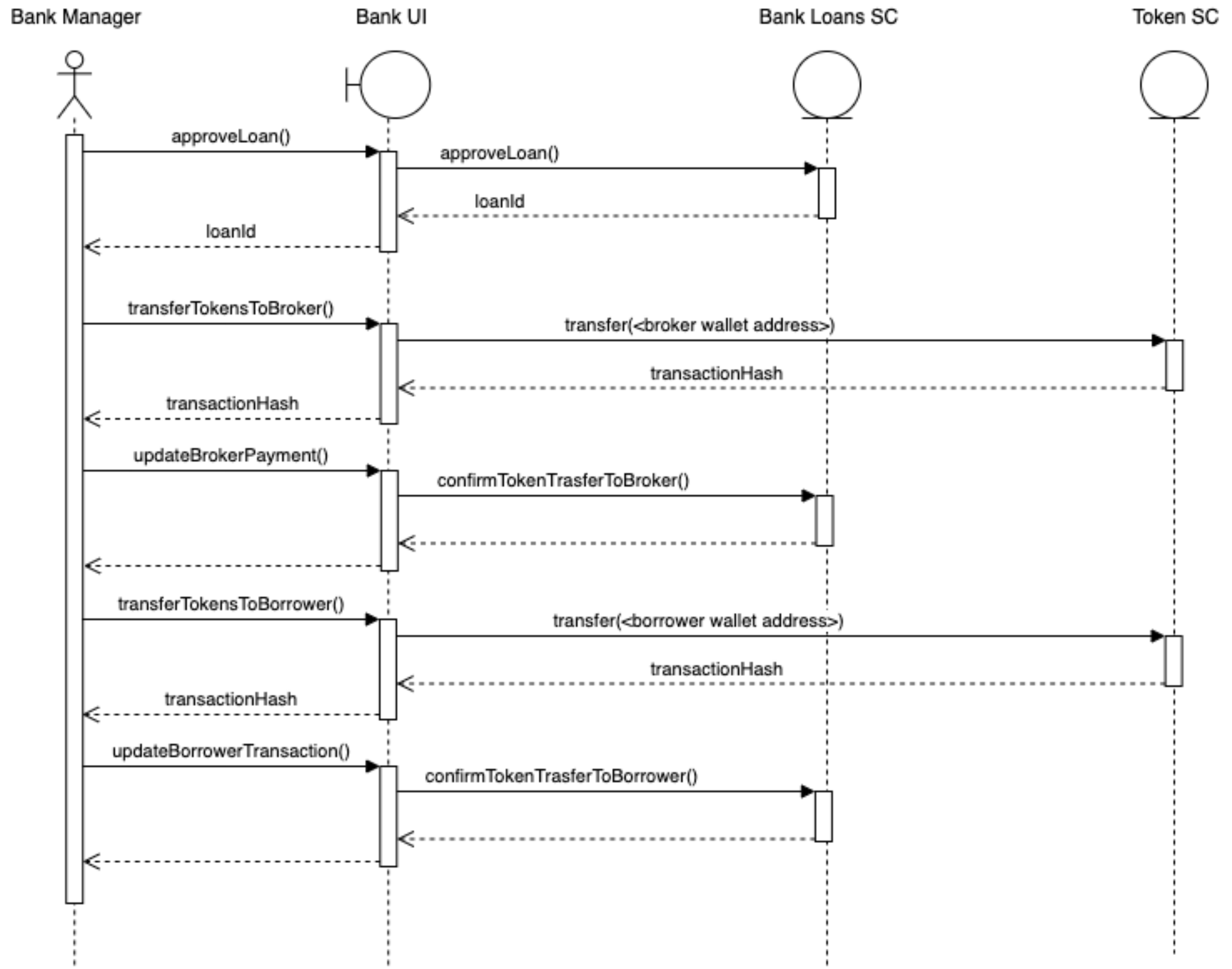
Sequence Diagram – Sign Loan



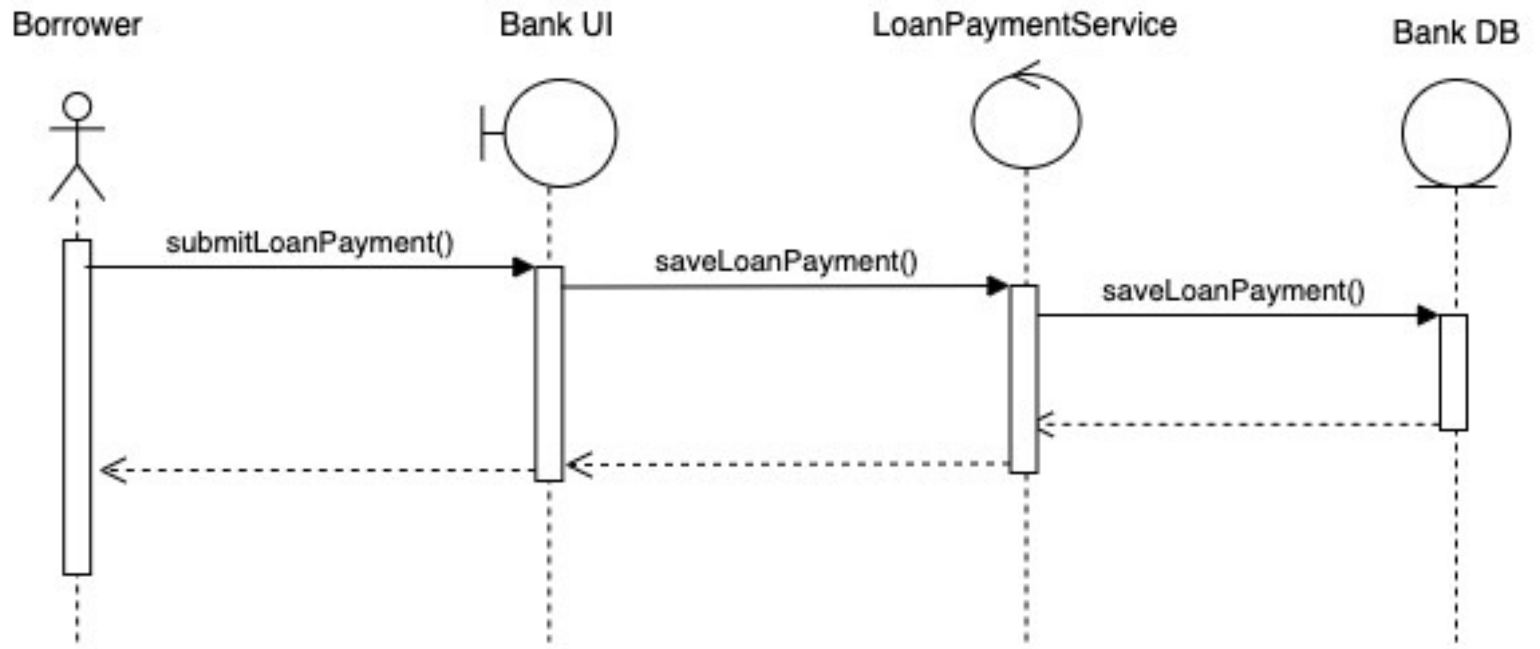
Sequence Diagram – Approve Loan



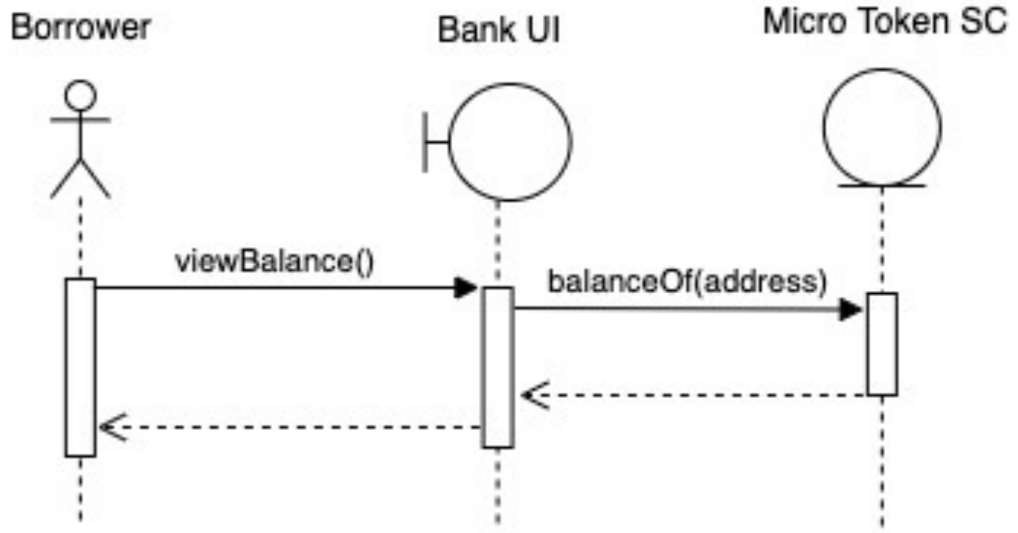
Sequence Diagram – Approve Loan and Transfer Tokens



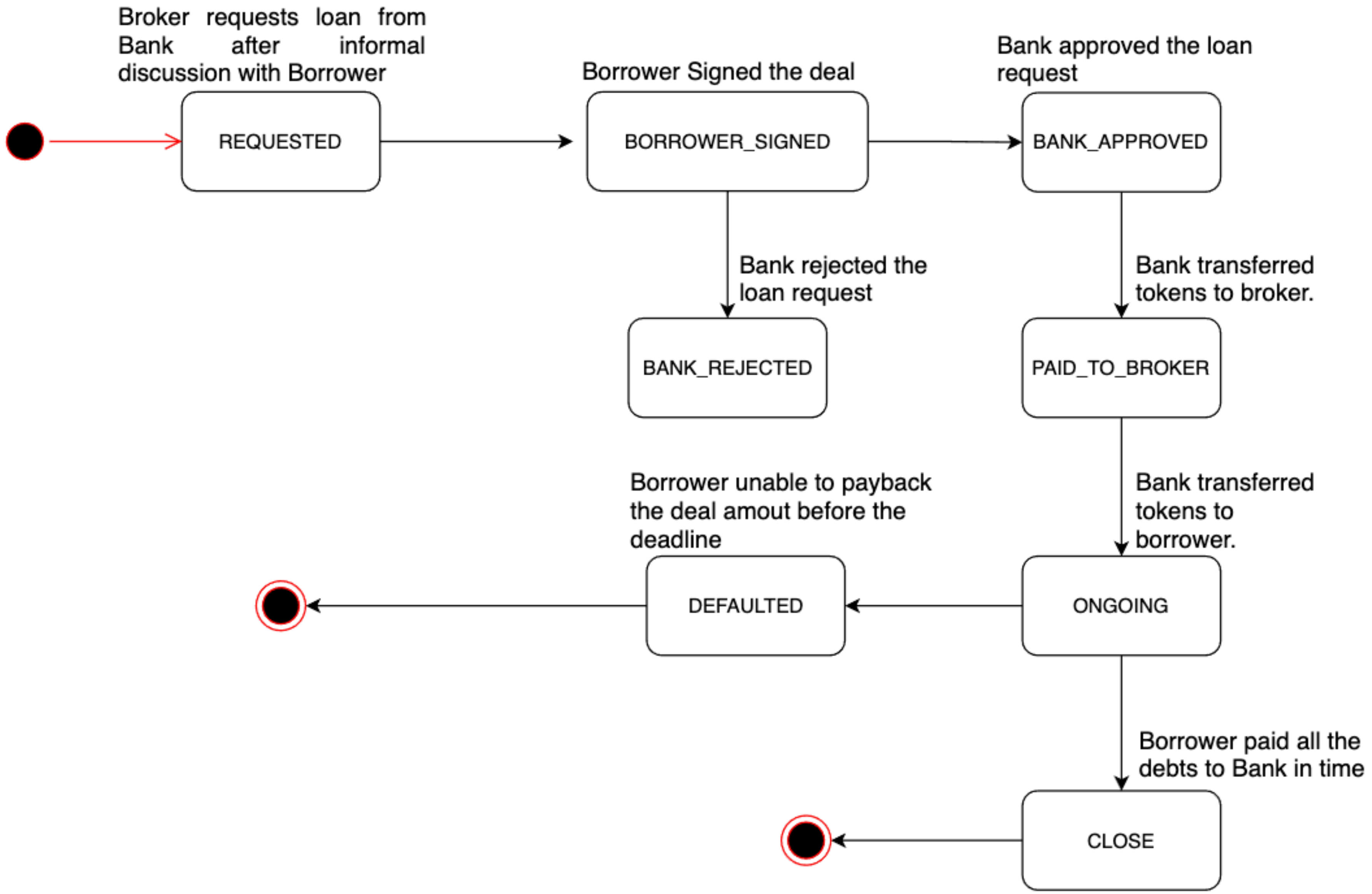
Sequence Diagram – Save Loan Payment



Sequence Diagram – View Token Balance

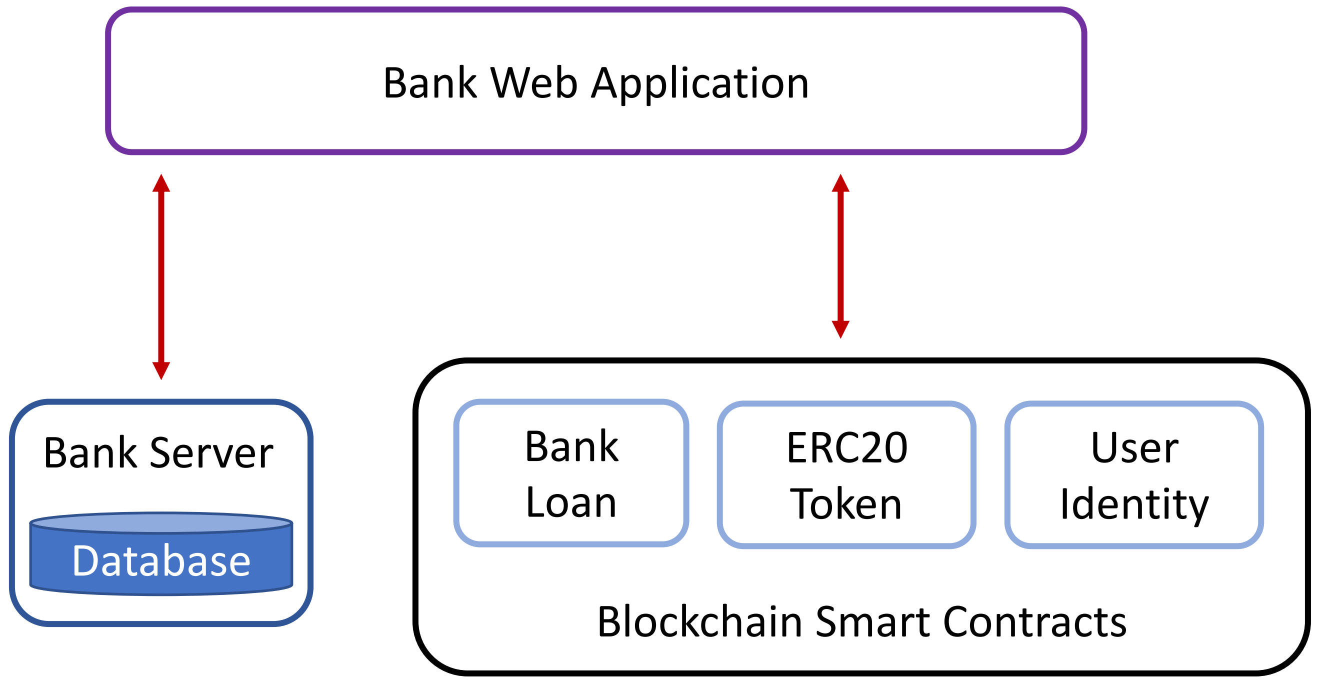


State Transition Diagram for Loan


Broker requests loan from Bank after informal discussion with Borrower
REQUESTED
Borrower Signed the deal
BORROWER_SIGNED
Bank approved the loan request
BANK_APPROVED
Bank rejected the loan request
BANK_REJECTED
Bank transferred tokens to broker.
PAID_TO_BROKER
Borrower unable to payback the deal amout before the deadline
DEFAULTED
Bank transferred tokens to borrower.
ONGOING
Borrower paid all the debts to Bank in time
CLOSE

System Design

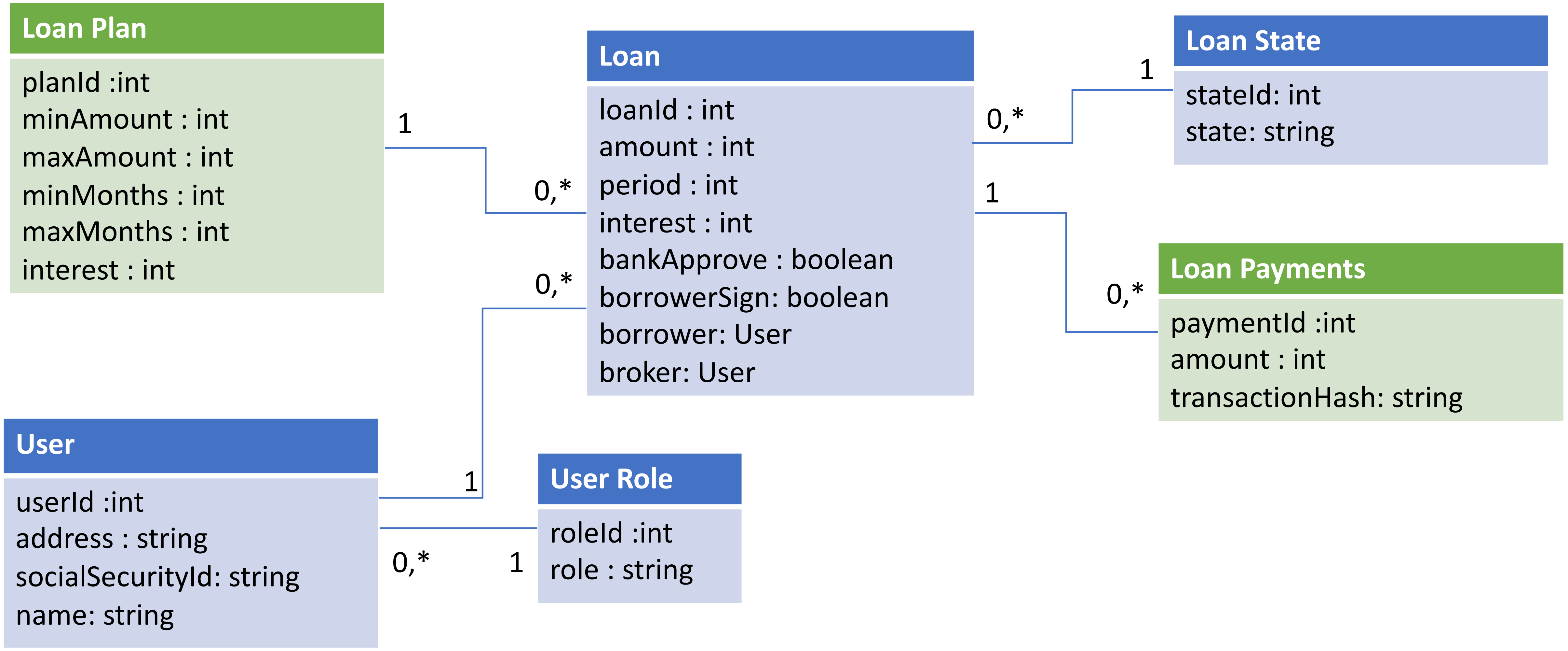
Layered Architecture



Green tables are in MongoDB

Blue tables are in blockchain

ER / Database Design - Bank



Smart Contract Designs

The Following slides describe the smart contract designs.

- **structs** required in each smart contract
- **ENUMs** required in each smart contract
- **modifiers** required in each smart contract functions
- **events** required in each smart contract
- **attributes** of each smart contract
- **functions** required to implement in each smart contract

Function structure

```
<Function Name>(<parameters>)
      : <Return type>
      : <[Optional Function modifiers]>
      : <[Optional Events]>
```

MicroToken.sol

```
contract MicroToken is IERC20 {

      string public constant symbol = "MFT";
      string public constant name = "Microfinance Token";
      uint8 public constant decimals = 0;
      uint private constant __totalSupply = 1000;
      mapping (address => uint) private __balanceOf;
      mapping (address => mapping (address => uint)) private __allowances;

      constructor() {
          __balanceOf[msg.sender] = __totalSupply;
      }

      function totalSupply() public pure override returns (uint256);
      function balanceOf(address _addr) public view override returns (uint balance);
      function transfer(address _to, uint _value) public override returns (bool success);
      function transferFrom(address _from, address _to, uint _value) public override returns (bool success);
      function approve(address _spender, uint _value) public override returns (bool success);
      function allowance(address _owner, address _spender) public view override returns (uint remaining)
}
```

UserIdentity.sol - Structs

```
//User struct to store Broker and Borrower details
struct User{
      uint id;
      string socialSecurityId; //unique social security id
      address userAddress;
      string name;
      Role role;
}
```

UserIdentity.sol - ENUMs

```
//User roles for the users.
enum Role {
         GUEST, // Default user role
         BROKER,
         BORROWER
}
```

UserIdentity.sol - Modifiers

```
modifier isAdmin()
{
          // Checks _address is the smart contract admin’s(Bank) address.
          require(admin == msg.sender, 'Admin Only’);
          _;
}
```

UserIdentity.sol – Attributes and Functions

- brokers :mapping(address -> User) //Stores Brokers’ data
- borrowers : mapping(address -> User) //Stores Borrowers’ data
- admin : address //Stores smart contract deployer’s(Bank) address

```
addBroker(User): void : isAdmin(_msg.sender)
addBorrower(User) : void : isAdmin(msg.sender)
verifyBroker(): bool : public
verifyBorrower() : bool : public
getAllBrokers() : User[] : public
getAllBorrowers() : User[] : public
```

BankLoan.sol – Structs

```
// Loan struct to store Loan details
struct Loan
{
      uint id;
      uint amount;
      uint months;
      uint interest;
      uint planId;
      LoanState state;
      address broker;
      address borrower;
      bool bankApprove;
      bool isBorrowerSigned;
}
```

BankLoan.sol – ENUMs

```
// Loan State Enum to store Loan states
enum LoanState{
     REQUESTED,
     BORROWER_SIGNED,
     BANK_APPROVED,
     BANK_REJECTED,
     PAID_TO_BROKER,
     ONGOING,
     DEFAULT,
     CLOSE
}
```

BankLoan.sol – Events

```
event loanRequest(// This event will emit when Broker creates a loan request.
      uint id,
      uint amount,
      uint months,
      uint interest,
      uint planId,
      LoanState state,
      address broker,
      address borrower,
      bool bankApprove,
      bool isBorrowerSigned,
);
```

BankLoan.sol – Modifiers

```
modifier isAdmin()
{
          // Checks function caller is the smart contract admin’s address.
}
modifier isBroker()
{
          // Checks function caller’s address registered as a Broker.
}
modifier isLoanBorrower(uint _loanId)
{
          // Checks function caller borrowed the loan.
}
modifier isValidLoan(uint _loanId)
{
         // Checks the _loanId exists in the system
}
modifier isLoanIn(uint _loanId, LoanState _state)
{
         // Checks the loan is in _state
}
```

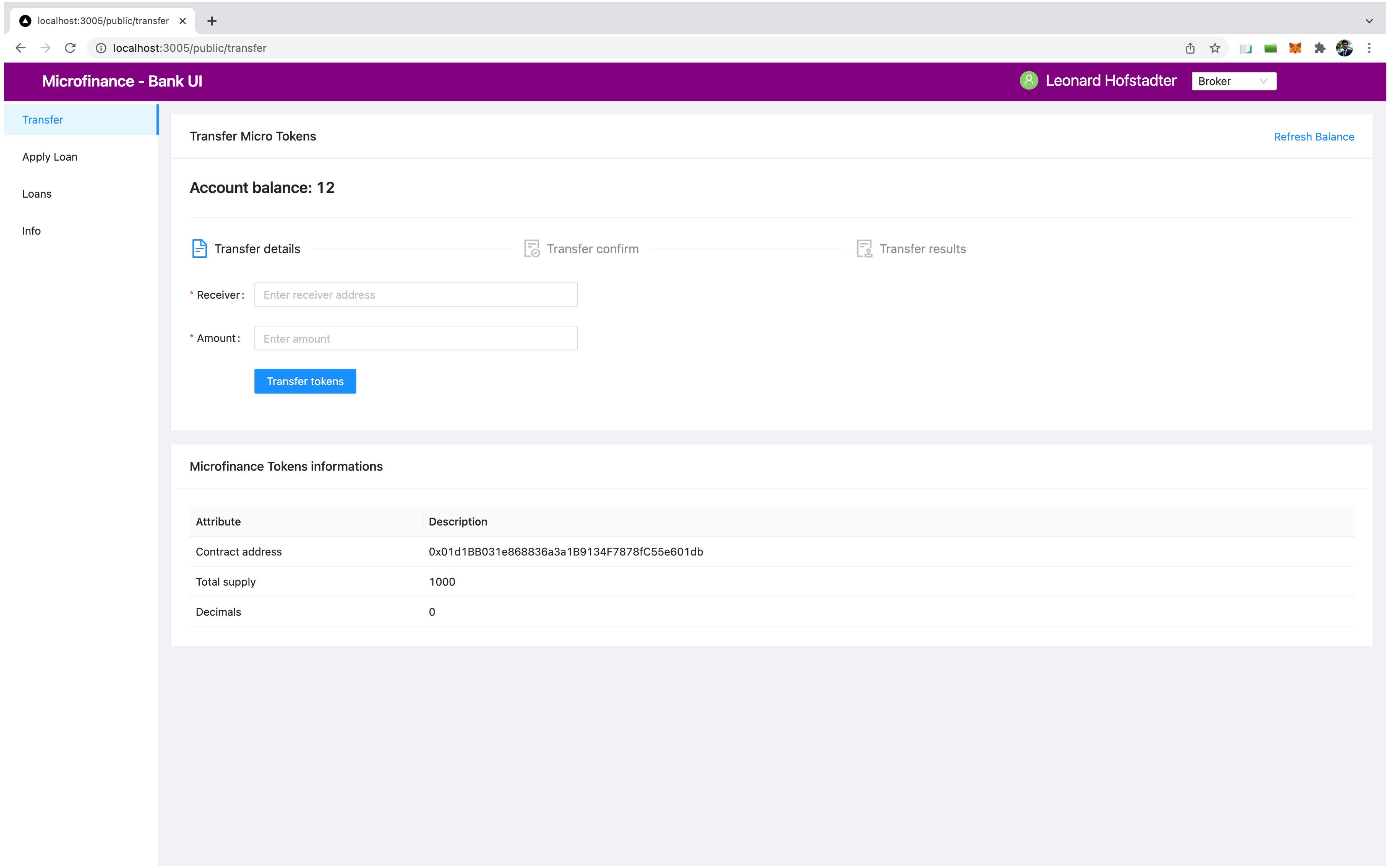
BankLoan.sol – Attributes and Functions

```
- identitySC : UserIdentity // Stores UserIdentity smart contract object
- admin : address //Store smart contract deployer’s address
- Loan[] loans // Stores loan data

applyLoan(amount, months, interest, planId, borrower) : void : isBroker() : loanRequest
signByBorrower (loanId): void : isLoanBorrower() isValidLoan(_loanId) isLoanIn(_loanId, REQUESTED)
approveLoan(loanId): void : isAdmin(), isValidLoan(_loanId), isLoanIn(_loanId, BORROWER_SIGNED)
rejectLoan(loanId): void : isAdmin(), isValidLoan(_loanId), isLoanIn(_loanId, BORROWER_SIGNED)
confirmTokenTrasferToBroker(loanId)
         : void
         : isAdmin(), isValidLoan(_loanId), isLoanIn(_loanId, BANK_APPROVED)
confirmTokenTrasferToBorrower(loanId): void : isAdmin(), isValidLoan(_loanId), isLoanIn(_loanId,
PAID_TO_BROKER)
closeLoan(loanId): void: isAdmin() isValidLoan(_loanId) isLoanIn(_loanId, ONGOING)
markAsDefaulted(loanId): void : isAdmin() isValidLoan(_loanId) isLoanIn(_loanId, ONGOING)
viewLoan (loanId): Loan : public
getLoans (): Loan[] : public
```

User Interface Design

Transfer Tokens UI


localhost:3005/public/transfer

Microfinance - Bank UI

Leonard Hofstadter

Broker

- Transfer
- Apply Loan
- Loans
- Info

Transfer Micro Tokens

Refresh Balance

Account balance: 12

Transfer details — Transfer confirm — Transfer results

* Receiver: Enter receiver address

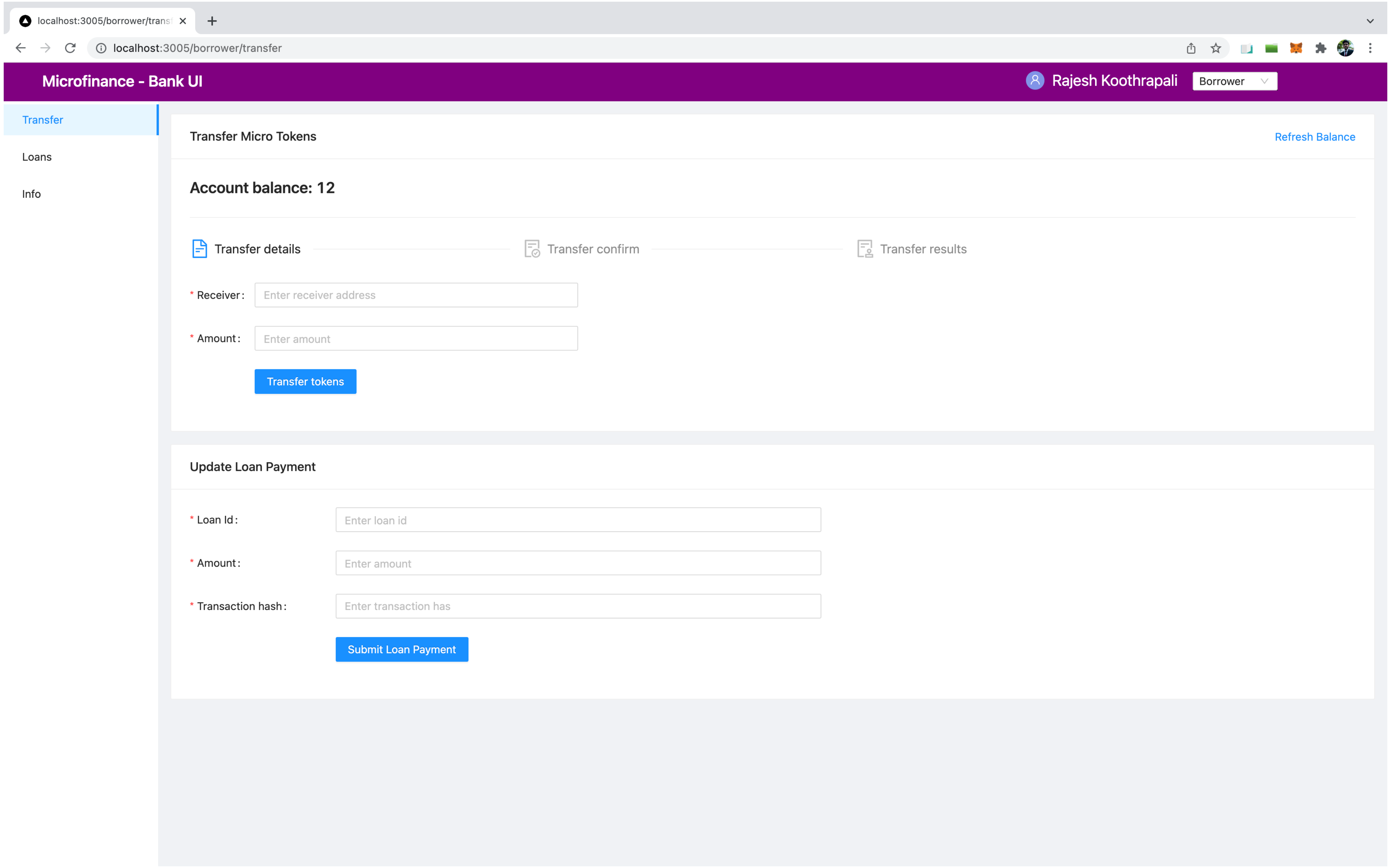
* Amount: Enter amount

Transfer tokens

Microfinance Tokens informations

Attribute	Description
Contract address	0x01d1BB031e868836a3a1B9134F7878fC55e601db
Total supply	1000
Decimals	0

Transfer Tokens UI for Borrower



localhost:3005/borrower/transfer

Microfinance - Bank UI

Rajesh Koothrapali Borrower

Transfer

Loans

Info

Transfer Micro Tokens

Refresh Balance

Account balance: 12

Transfer details — Transfer confirm — Transfer results

* Receiver: Enter receiver address

* Amount: Enter amount

Transfer tokens

Update Loan Payment

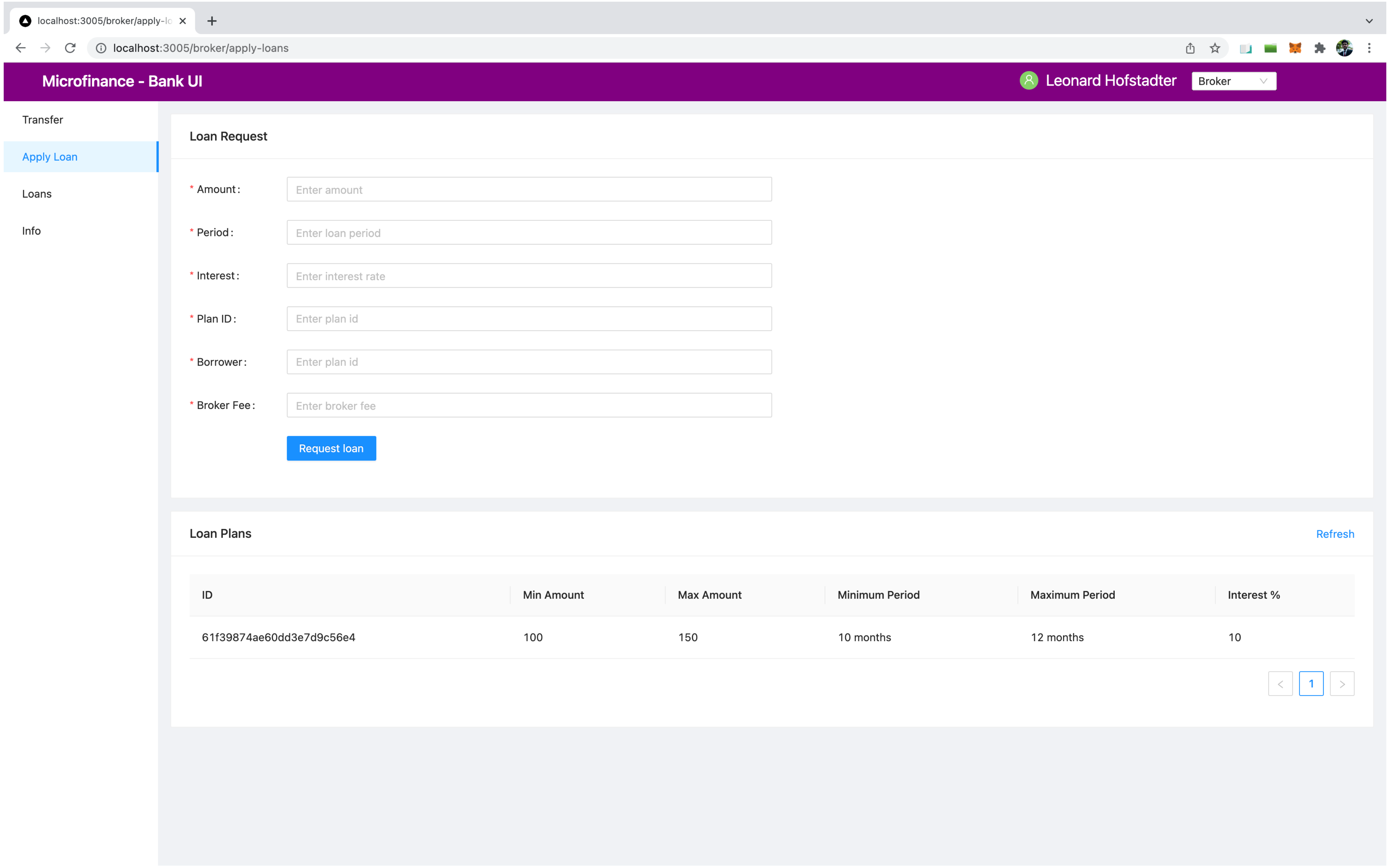
* Loan Id: Enter loan id

* Amount: Enter amount

* Transaction hash: Enter transaction has

Submit Loan Payment

Apply Loan UI



Loan Request

* Amount :	Enter amount
* Period :	Enter loan period
* Interest :	Enter interest rate
* Plan ID :	Enter plan id
* Borrower :	Enter plan id
* Broker Fee :	Enter broker fee

Request loan

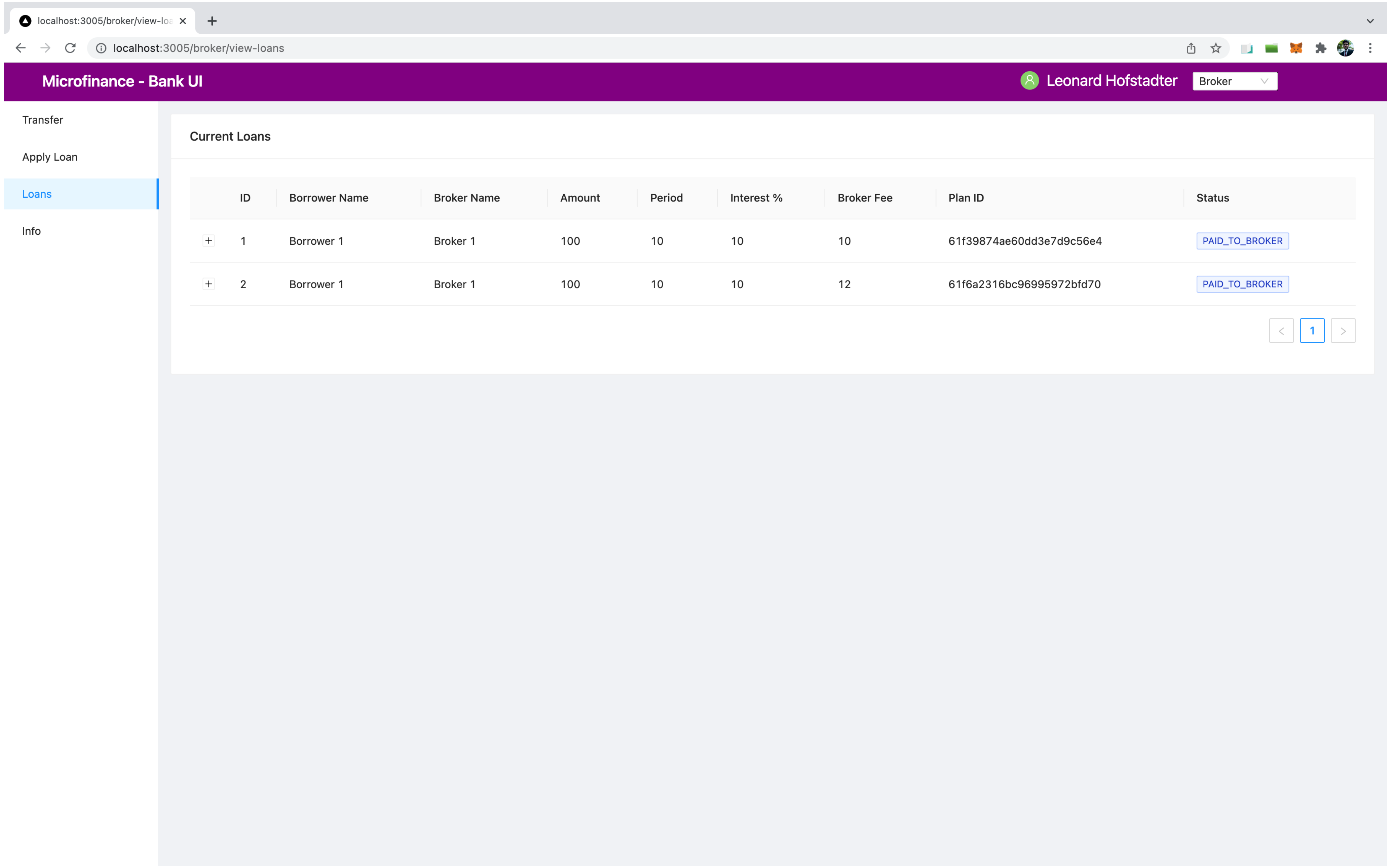
Loan Plans

Refresh

ID	Min Amount	Max Amount	Minimum Period	Maximum Period	Interest %
61f39874ae60dd3e7d9c56e4	100	150	10 months	12 months	10

< 1 >

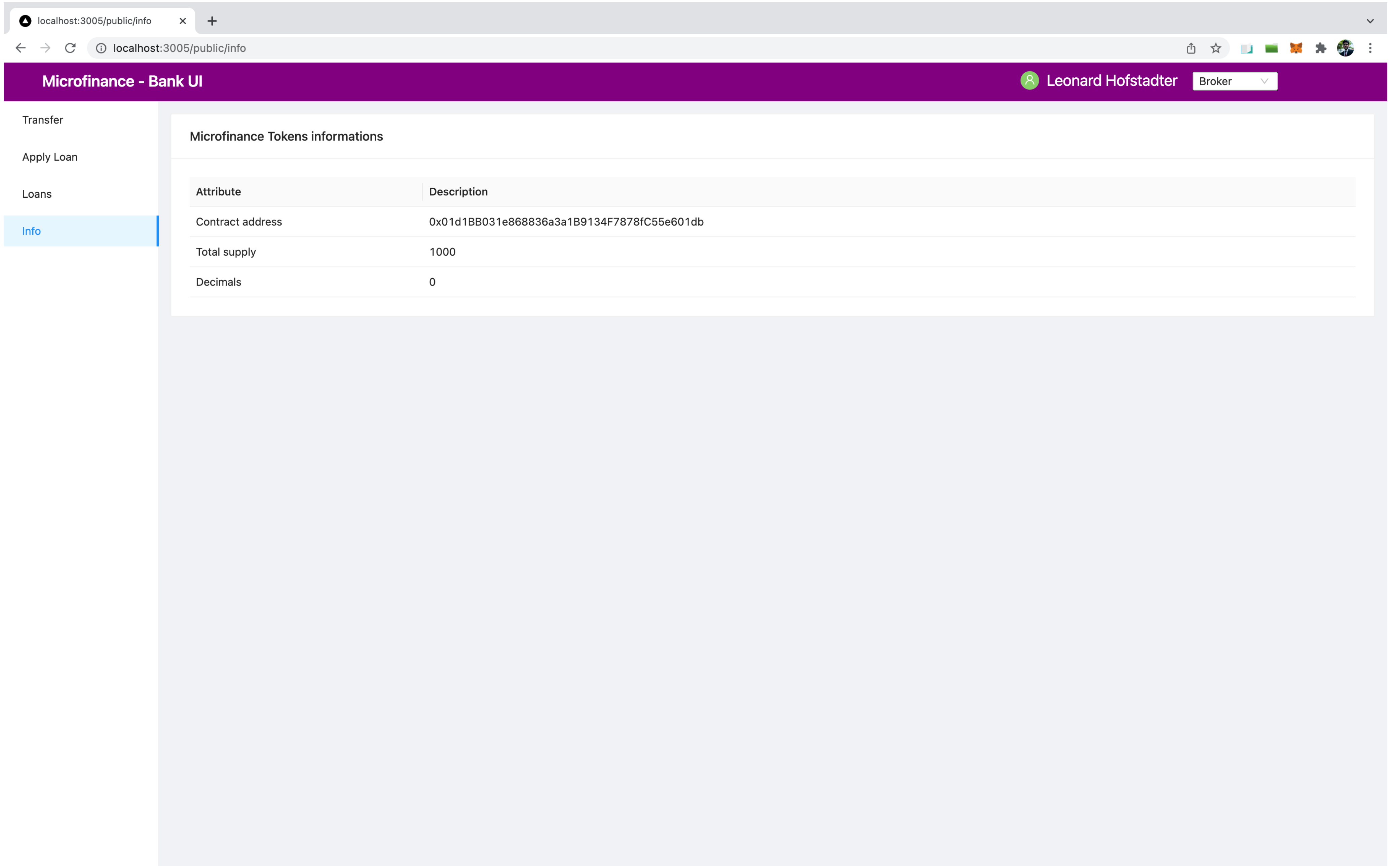
Loans Table UI



Current Loans

	ID	Borrower Name	Broker Name	Amount	Period	Interest %	Broker Fee	Plan ID	Status
+	1	Borrower 1	Broker 1	100	10	10	10	61f39874ae60dd3e7d9c56e4	PAID_TO_BROKER
+	2	Borrower 1	Broker 1	100	10	10	12	61f6a2316bc96995972bfd70	PAID_TO_BROKER

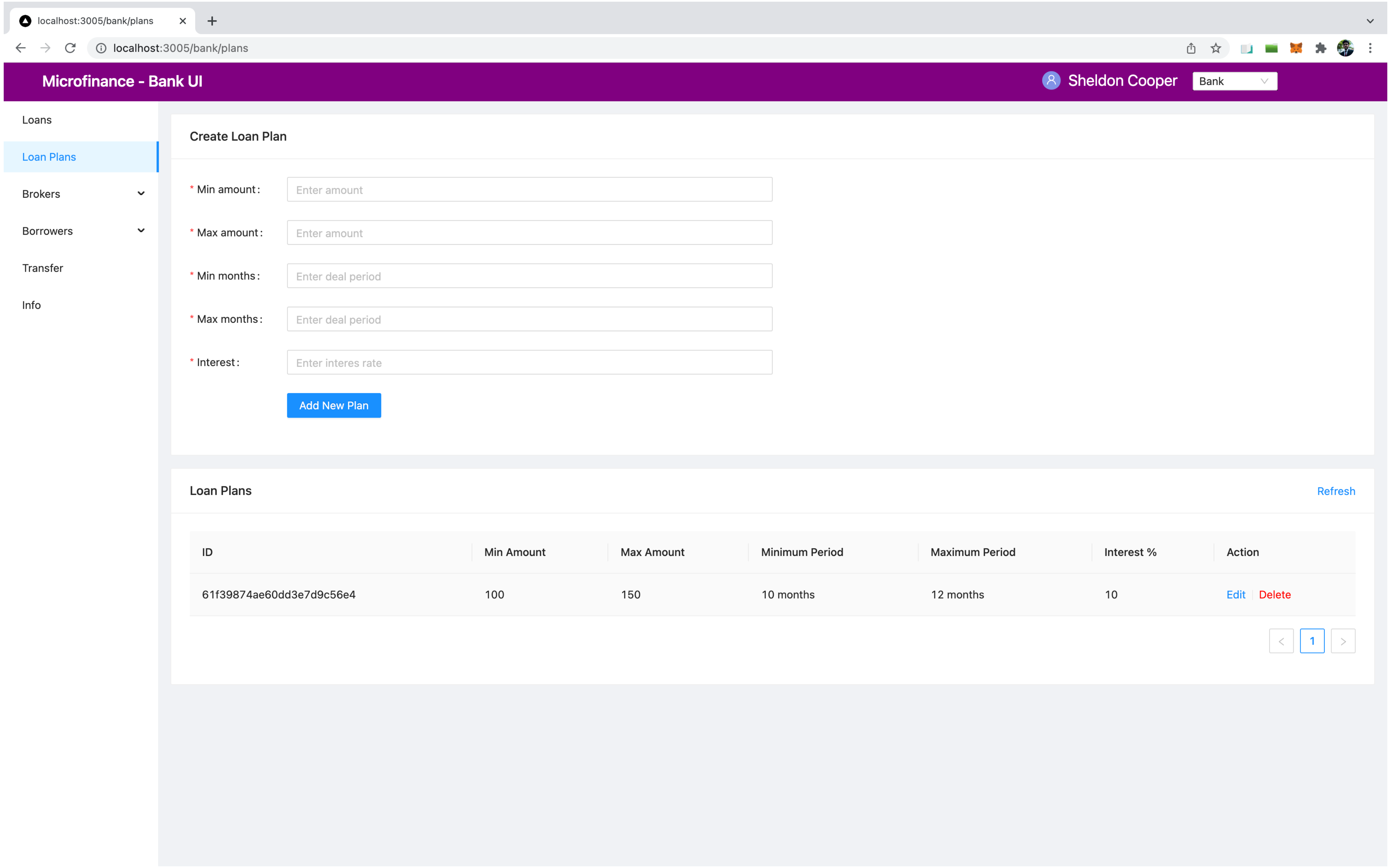
Token Information UI



Microfinance Tokens informations

Attribute	Description
Contract address	0x01d1BB031e868836a3a1B9134F7878fC55e601db
Total supply	1000
Decimals	0

Loan Plans UI



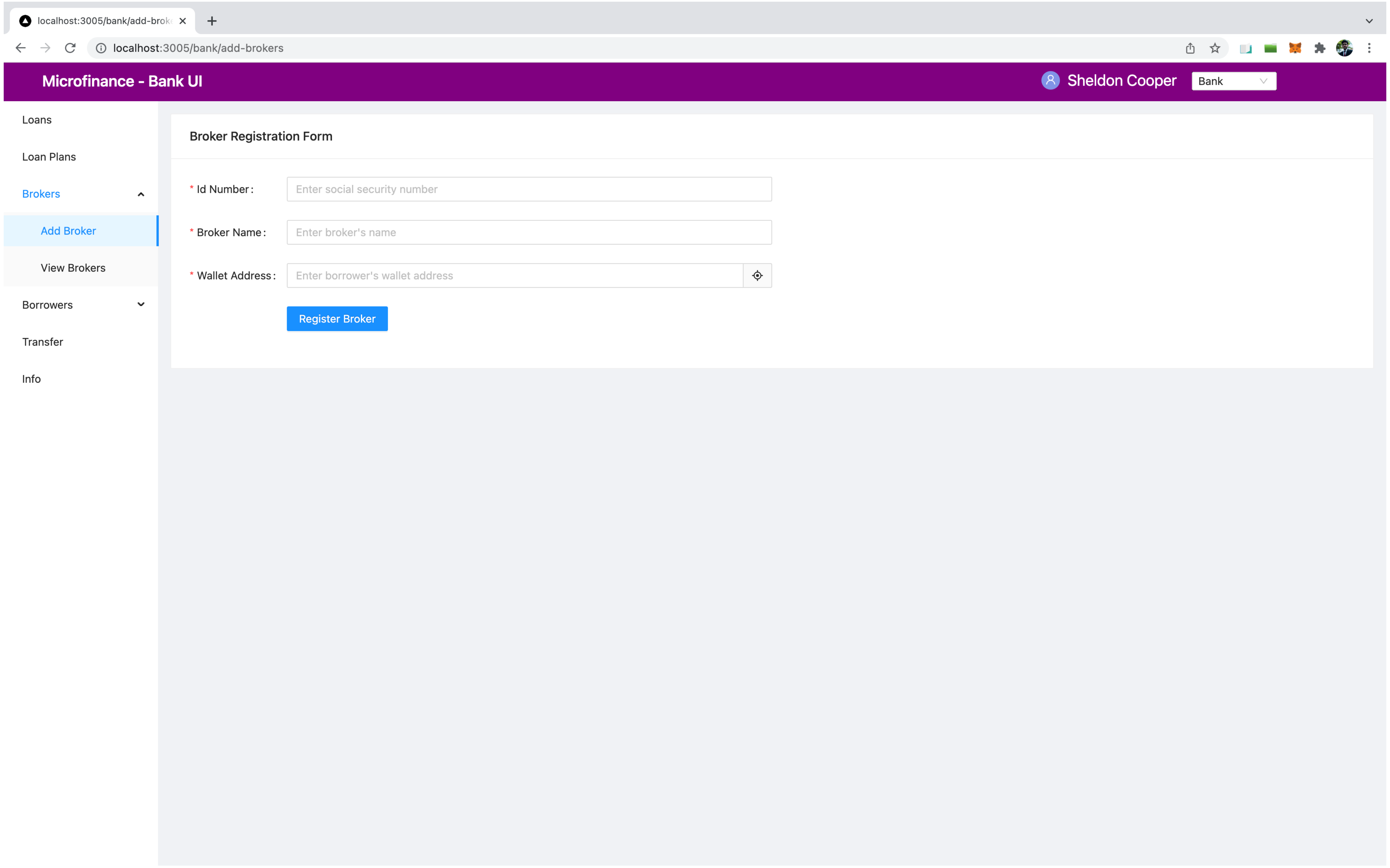
Create Loan Plan

* Min amount :	Enter amount
* Max amount :	Enter amount
* Min months :	Enter deal period
* Max months :	Enter deal period
* Interest :	Enter interes rate

Loan Plans

ID	Min Amount	Max Amount	Minimum Period	Maximum Period	Interest %	Action
61f39874ae60dd3e7d9c56e4	100	150	10 months	12 months	10	Edit Delete

Add Broker UI



localhost:3005/bank/add-brokers

Microfinance - Bank UI

Sheldon Cooper

Bank

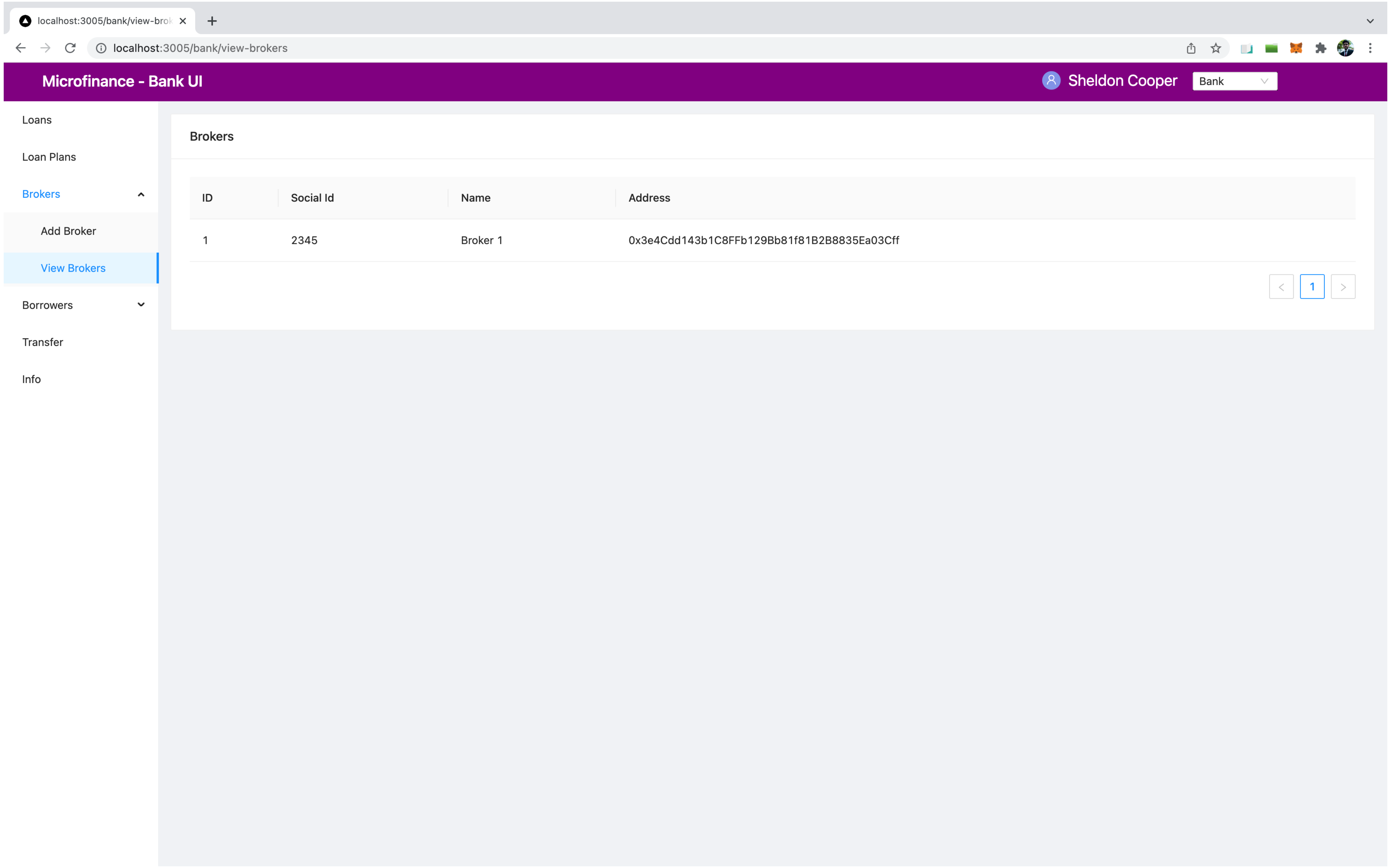
- Loans
- Loan Plans
- Brokers
  - Add Broker
  - View Brokers
- Borrowers
- Transfer
- Info

Broker Registration Form

* Id Number :	Enter social security number
* Broker Name :	Enter broker's name
* Wallet Address :	Enter borrower's wallet address

Register Broker

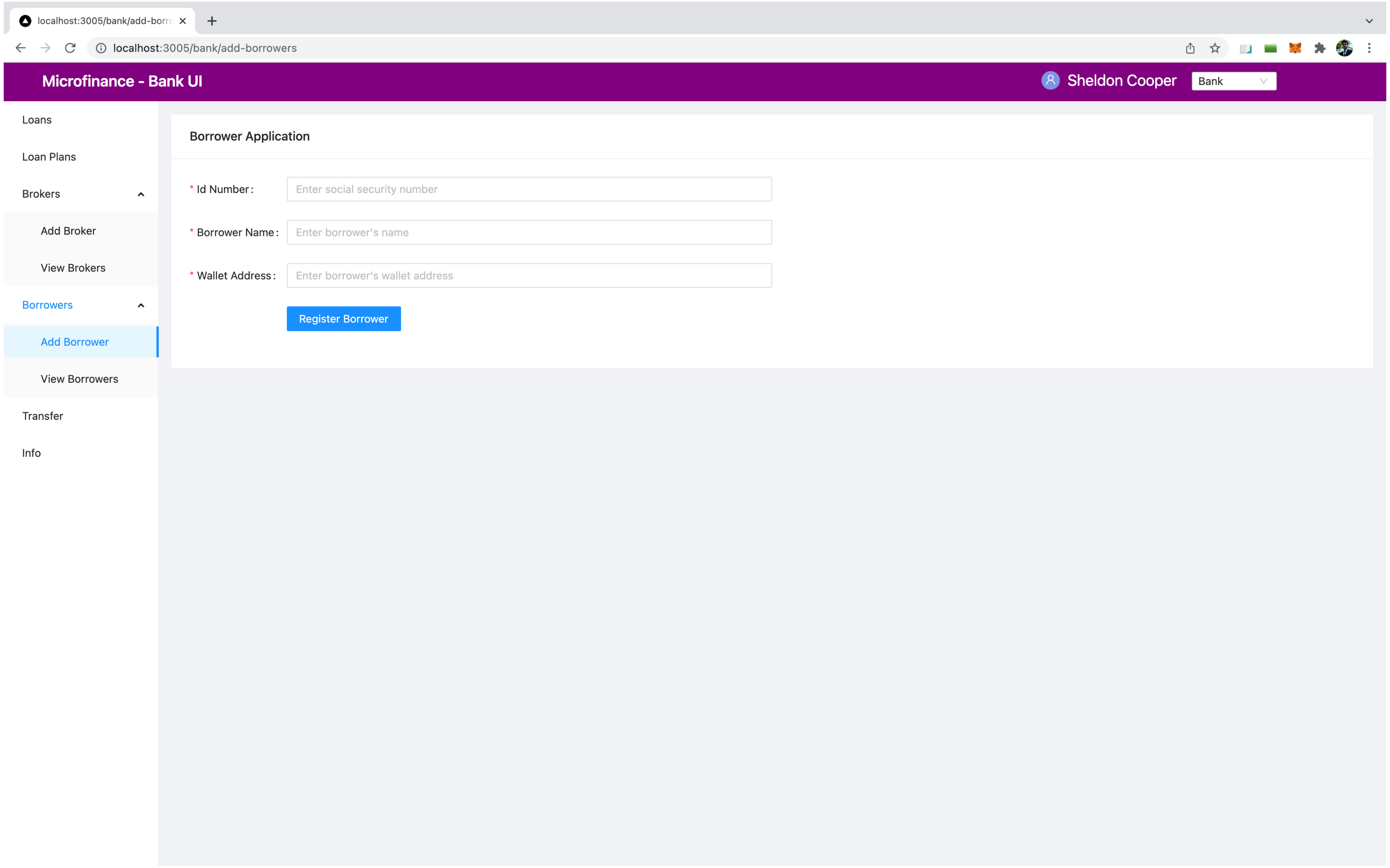
View Brokers UI



Brokers

ID	Social Id	Name	Address
1	2345	Broker 1	0x3e4Cdd143b1C8FFb129Bb81f81B2B8835Ea03Cff

Add Borrower UI



Borrower Application

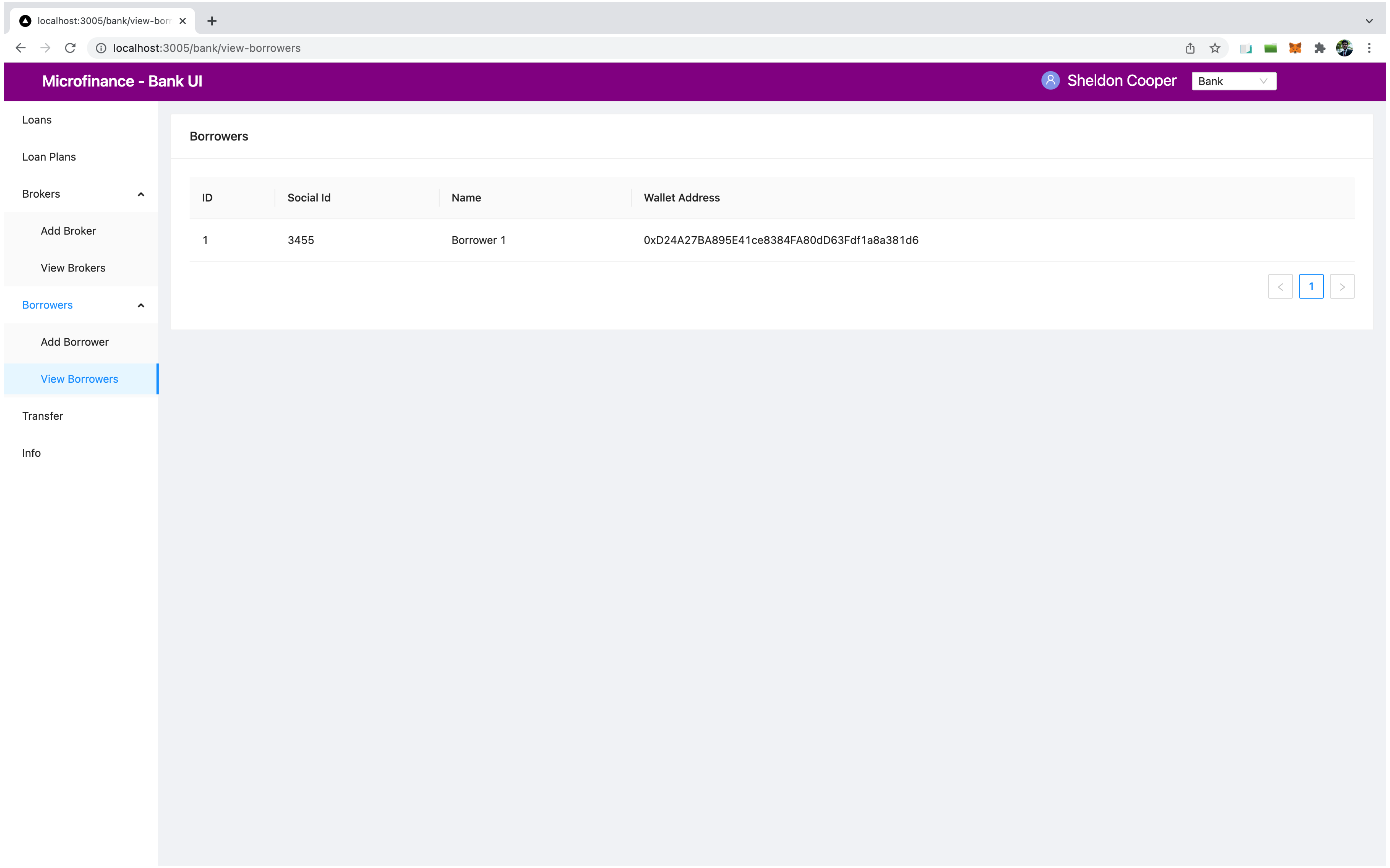
* Id Number : Enter social security number

* Borrower Name : Enter borrower's name

* Wallet Address : Enter borrower's wallet address

Register Borrower

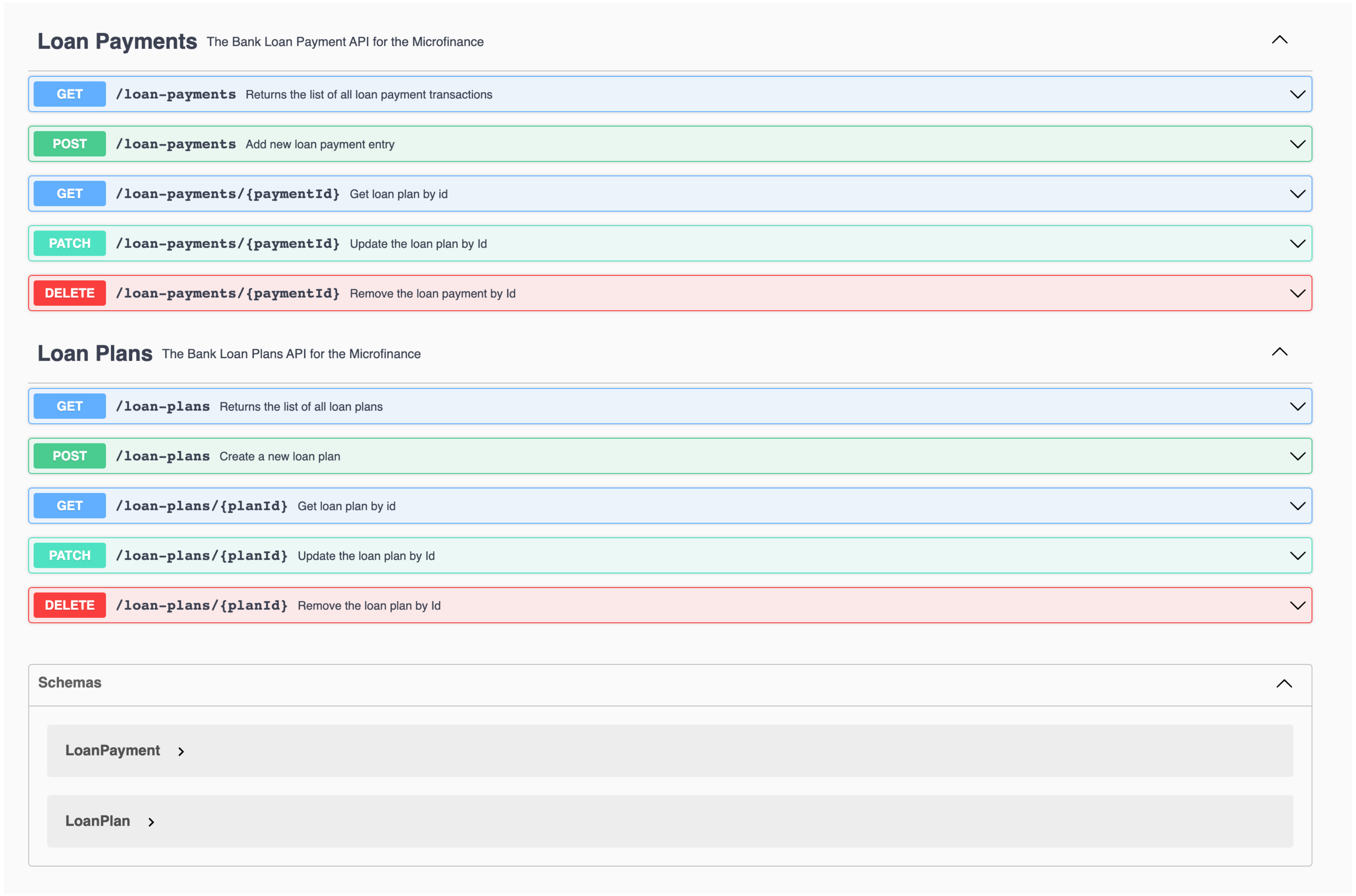
View Borrowers UI



Borrowers

ID	Social Id	Name	Wallet Address
1	3455	Borrower 1	0xD24A27BA895E41ce8384FA80dD63Fdf1a8a381d6

Bank Web Server API



Loan Payments The Bank Loan Payment API for the Microfinance

GET `/loan-payments` Returns the list of all loan payment transactions

POST `/loan-payments` Add new loan payment entry

GET `/loan-payments/{paymentId}` Get loan plan by id

PATCH `/loan-payments/{paymentId}` Update the loan plan by Id

DELETE `/loan-payments/{paymentId}` Remove the loan payment by Id

Loan Plans The Bank Loan Plans API for the Microfinance

GET `/loan-plans` Returns the list of all loan plans

POST `/loan-plans` Create a new loan plan

GET `/loan-plans/{planId}` Get loan plan by id

PATCH `/loan-plans/{planId}` Update the loan plan by Id

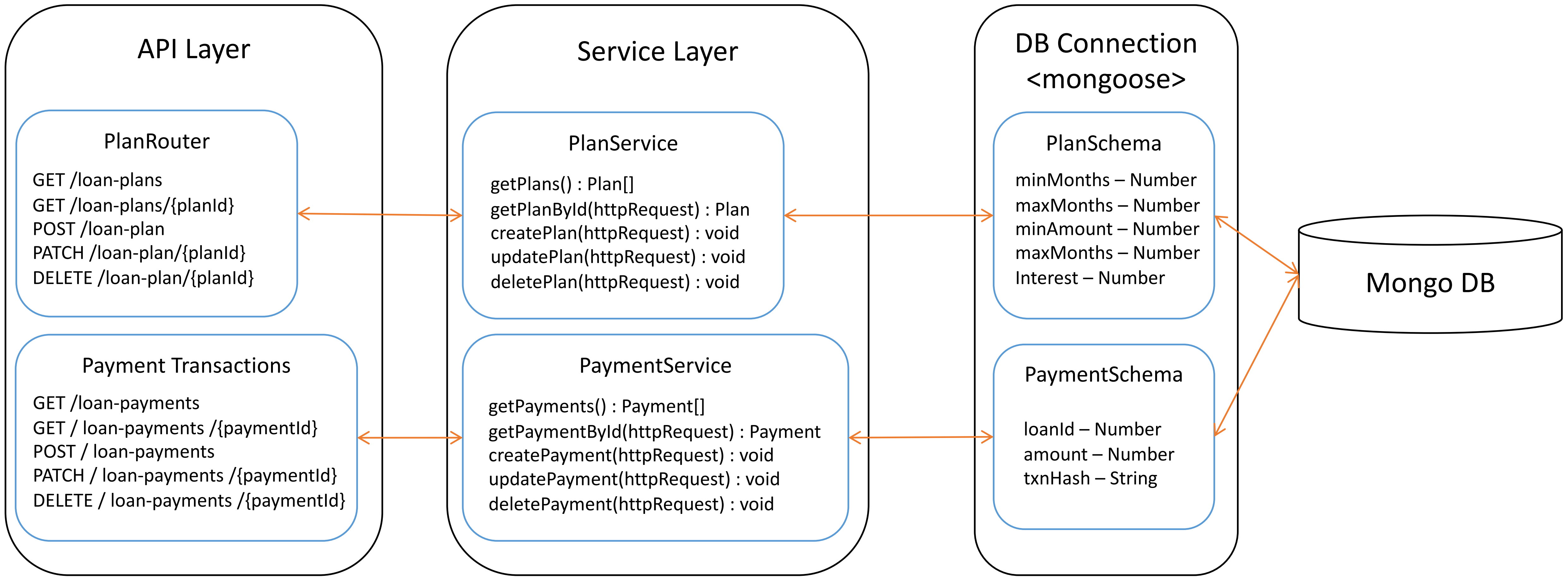
DELETE `/loan-plans/{planId}` Remove the loan plan by Id

Schemas

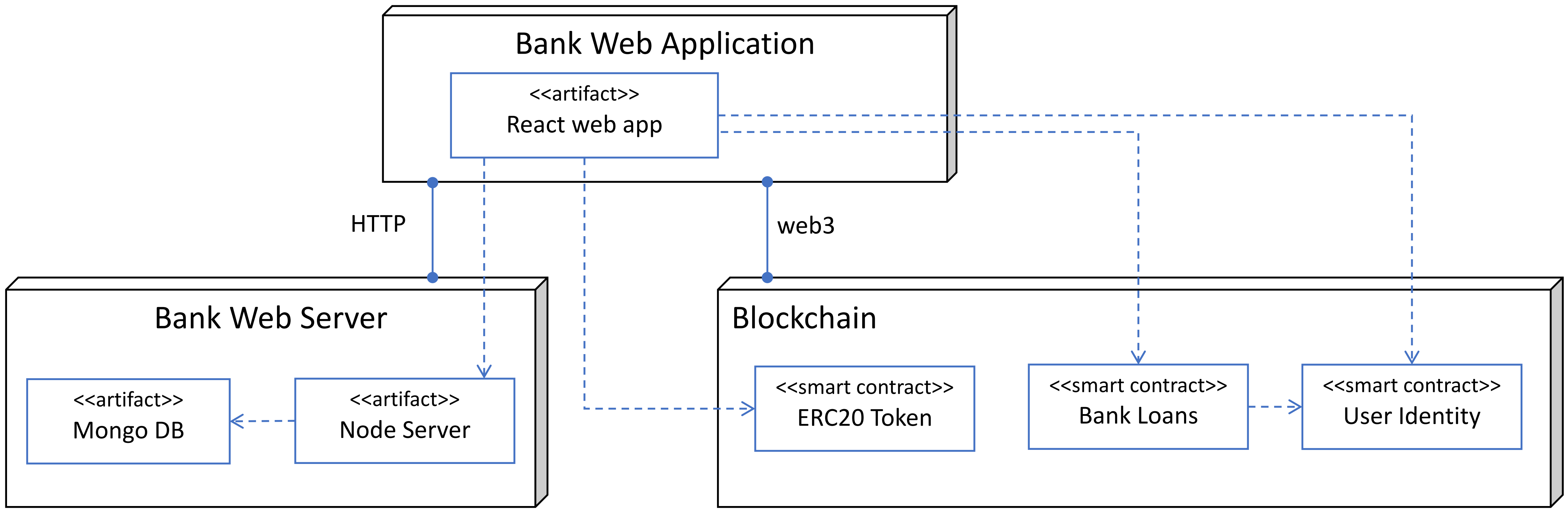
LoanPayment

LoanPlan

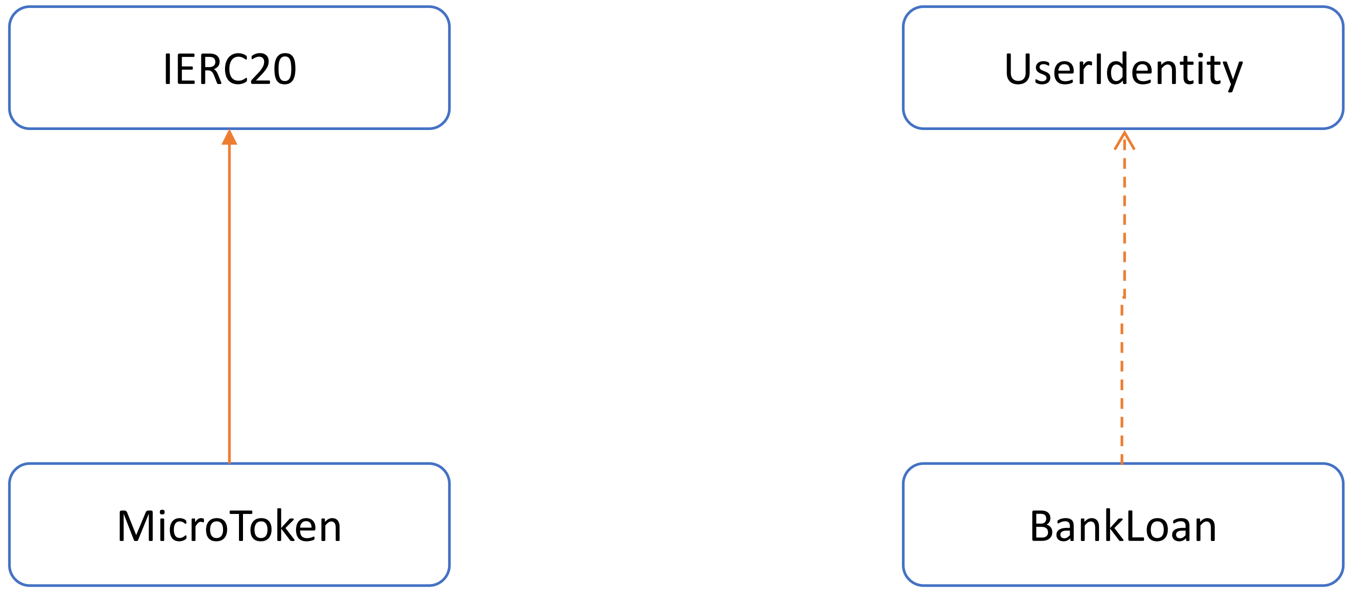
Bank Server Architecture Diagram


API Layer
PlanRouter
GET /loan-plans
GET /loan-plans/{planId}
POST /loan-plan
PATCH /loan-plan/{planId}
DELETE /loan-plan/{planId}
Payment Transactions
GET /loan-payments
GET / loan-payments /{paymentId}
POST / loan-payments
PATCH / loan-payments /{paymentId}
DELETE / loan-payments /{paymentId}
Service Layer
PlanService
getPlans() : Plan[]
getPlanById(httpRequest) : Plan
createPlan(httpRequest) : void
updatePlan(httpRequest) : void
deletePlan(httpRequest) : void
PaymentService
getPayments() : Payment[]
getPaymentById(httpRequest) : Payment
createPayment(httpRequest) : void
updatePayment(httpRequest) : void
deletePayment(httpRequest) : void
DB Connection
<mongoose>
PlanSchema
minMonths – Number
maxMonths – Number
minAmount – Number
maxMonths – Number
Interest – Number
PaymentSchema
loanId – Number
amount – Number
txnHash – String
Mongo DB

Deployment Diagram



Smart Contract Dependency Diagram


IERC20
UserIdentity
MicroToken
BankLoan