CSE 542 Introduction to Blockchain

Ahmedabad University

Lending and Borrowing Protocol

Group 17

Nish Parikh	AU2140039
Kathan Dave	AU2140113
Vatsal Kayastha	AU2140142
Vanaja Agrawal	AU2140213

Problem Statement

- **Objective :** Develop a lending and borrowing protocol on the blockchain using Solidity, inspired by Aave. Users can lend cryptocurrency to earn interest and borrow against deposited collateral.

- **Key Features:** Users can lend cryptocurrency to a liquidity pool, earning interest on deposits, and borrow cryptocurrency by providing collateral, paying interest on borrows.

- **Challenges:** Managing interest rates effectively and creating a new currency for deposits and loans within a single blockchain environment pose significant hurdles.

Stakeholders

- **Lenders:** These are users who deposit digital assets into the lending pool with the intention of earning interest on the amount they lend. They contribute liquidity to the pool, facilitating borrowing activities.
- **Borrowers:** Users who require digital assets can borrow from the lending pool by providing collateral as security. They access liquidity provided by lenders, allowing them to fulfill their borrowing needs.

Smart contracts

1. ERC20.sol : Manages AToken generation and burning based on user deposits and withdrawals from the lending pool.

2. MyToken.sol : Generates our own currency which has symbol IBT which we are minting to users accounts and is used for transactions as a second cryptocurrency(testing).

3. LendingPool.sol : Provides core functionalities for the lending and borrowing protocol, including deposit, withdrawal, borrowing, and repayment, while managing collateralization, interest calculation, and interactions with ERC20.sol for AToken management.

List of Databases and technology used

- **Databases:** Data is retrieved directly from the smart contracts deployed on the blockchain.
- **Technology Stack:**
  - **Solidity:** Used to write the smart contracts that define the logic and functionalities of the lending and borrowing protocol.
  - **Hardhat:** Utilized for project configuration, testing, and deployment of the smart contracts.
  - **Polygon Amoy Testnet:** Chosen as the test network for deploying and testing the smart contracts, providing a scalable and low-cost environment for development and experimentation.

Structure of blockchain

- **PoS Consensus:** Polygon utilizes a Proof of Stake (PoS) consensus mechanism where validators stake tokens as collateral to validate transactions and secure the network.
- **Validators and Staking:** Validators propose and validate blocks, while users can stake tokens as delegators to support validators and earn rewards.
- **Finality:** PoS provides faster and more predictable finality for transactions compared to Ethereum's PoW, enhancing transaction efficiency.
- **Scalability and Efficiency:** PoS contributes to Polygon's scalability and efficiency by reducing energy consumption and enabling faster block confirmation times.

System Architecture

UML Sequencing diagram

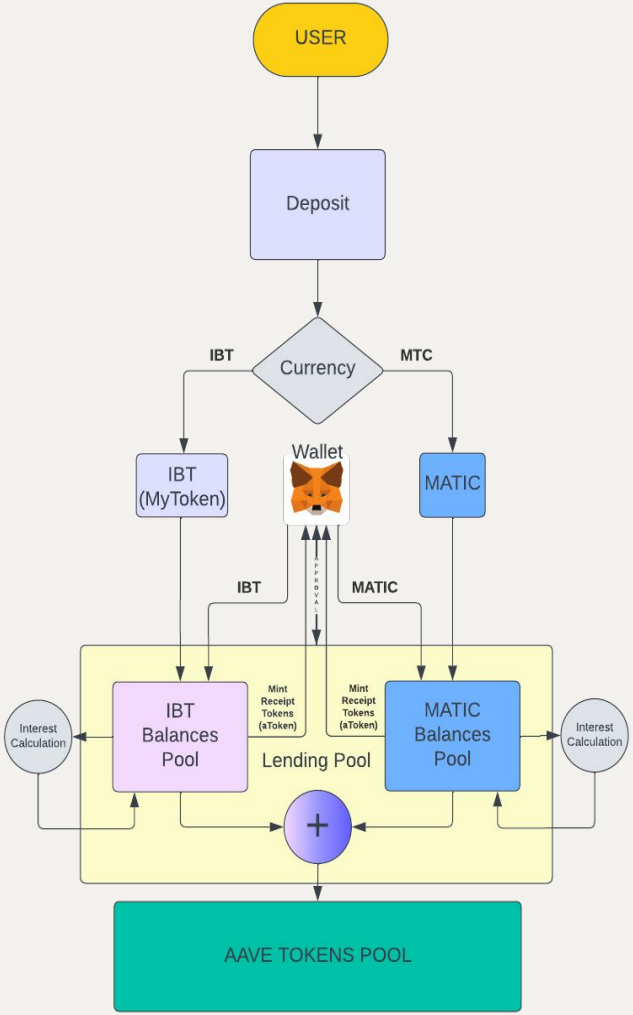
Logical Designs & flowchart

Functionalities

- Deposit
- Withdraw
- Borrow
- Replay

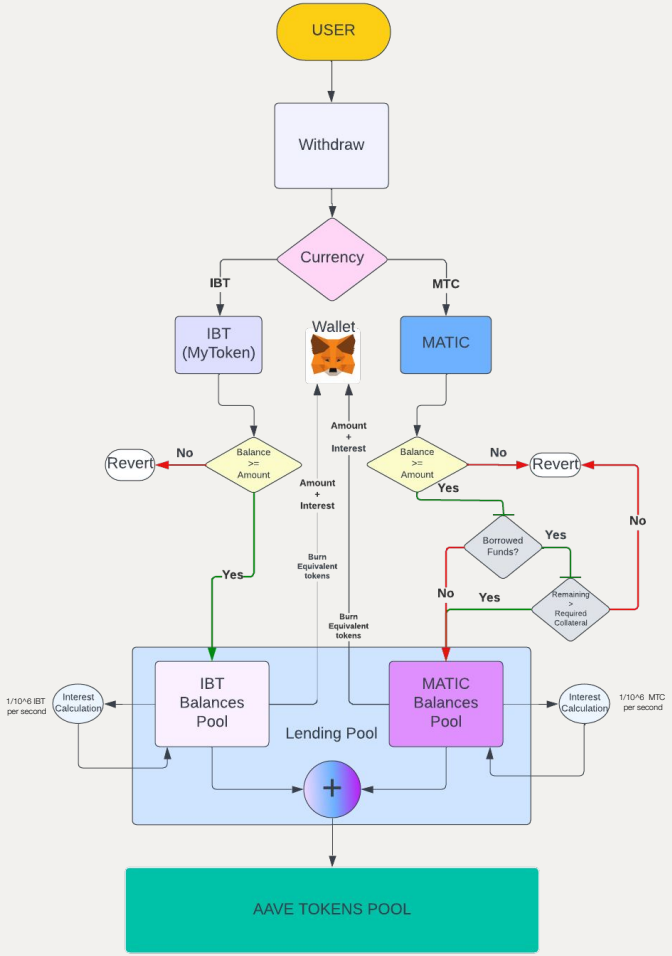
Deposit/Lend

[Link to chart]



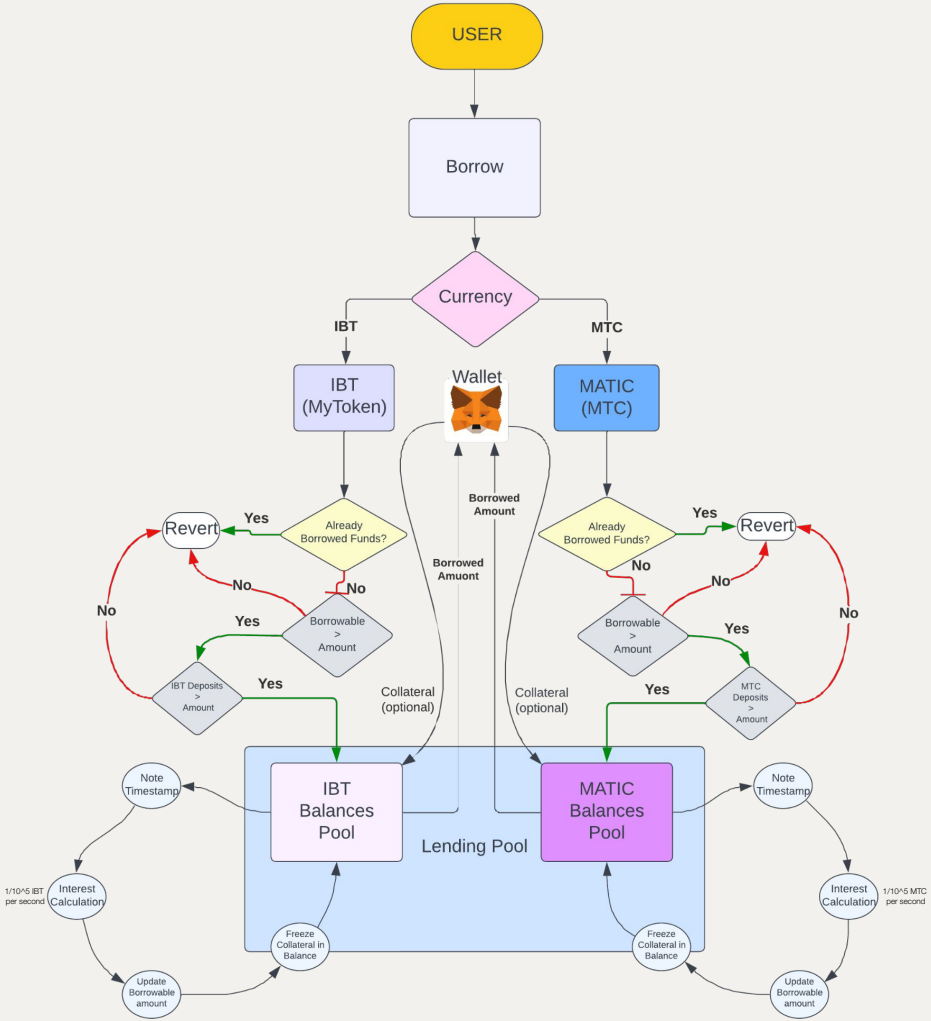
Withdraw

[Link to chart]



Borrow

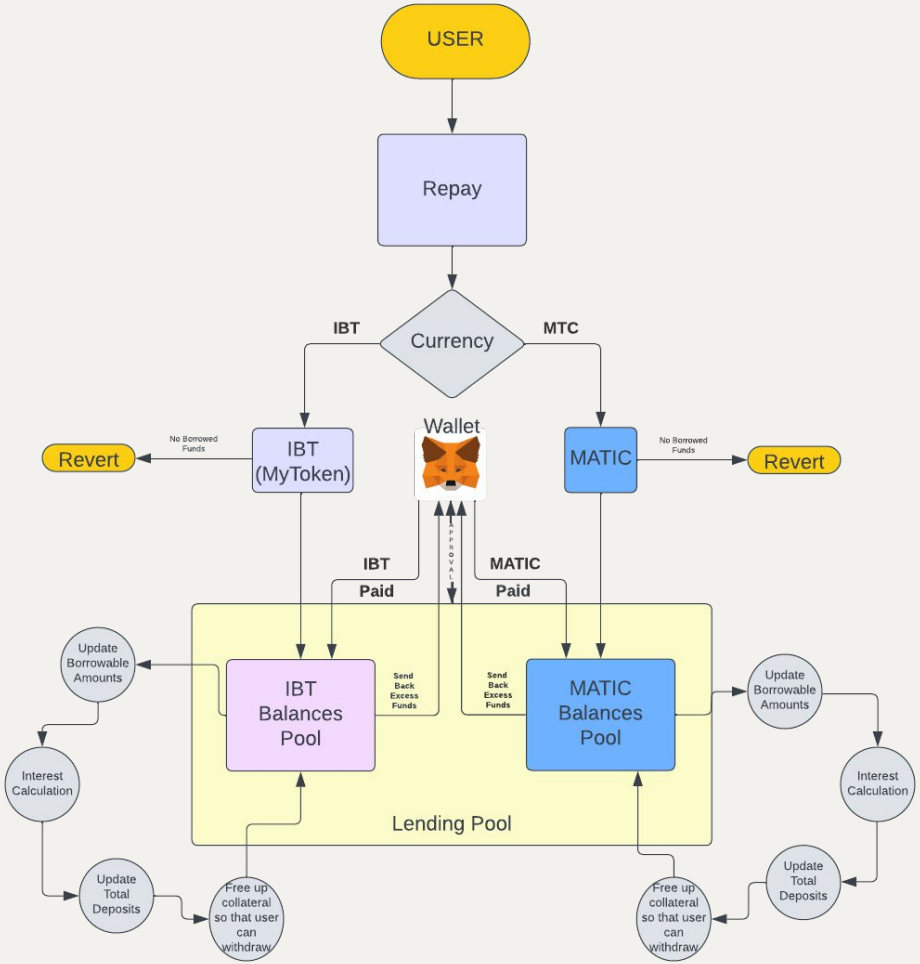
[Link to chart]



Ahmedabad University

Repay

[Link to chart](#)



Ahmedabad University

Programs/code developed and deployed on the Polygon Amoy Testnet

- Live Demo

MyToken.sol - https://amoy.polygonscan.com/address/0xB2def282Dd54101639D1e940eBd47D2F87dc8830

LendingPool.sol - https://amoy.polygonscan.com/address/0xf2389E52327AdD85650C3A8DD1739822690BfC80

Github: https://github.com/Vatsalkayastha/Aave-Blockchain

Results

- Link

Programs/code developed and deployed with test results

```
// Deposit funds into the lending pool and mint tokens
function depositMATIC() external payable {
    uint256 _maticAmount = msg.value;
    MTC_price = 1.72 * (10 ** 18);
    require(_maticAmount > 0, "Amount must be greater than 0");

    token.mintTokenswithMTC(msg.sender, _maticAmount);
    matic_balances[msg.sender] += _maticAmount;


    matic_totalDeposits += _maticAmount;
    if (matic_balances[msg.sender] == _maticAmount) {
        matic_isFirstWithdraw = true;
        borrowable_amount[msg.sender] = 0;
        matic_deposit_timestamp[msg.sender] = block.timestamp;
        matic_accruedInterest[msg.sender] = 0;
        matic_interest_balances[msg.sender] = _maticAmount;
    } else {

        matic_accruedInterest[msg.sender] +=
        ((block.timestamp - matic_deposit_timestamp[msg.sender]) *
        matic_balances[msg.sender]) / 1000000;
        matic_deposit_timestamp[msg.sender] = block.timestamp;
        matic_interest_balances[msg.sender] += _maticAmount;
    }
    borrowable_amount[msg.sender] += (_maticAmount * (7) * (MTC_price)) /10**19;
    console.log(matic_balances[msg.sender]);
    console.log(matic_deposit_timestamp[msg.sender]);
    console.log(matic_accruedInterest[msg.sender]);

    emit Deposit(msg.sender, _maticAmount, _maticAmount);
    emit BalanceAfterDeposit(msg.sender, matic_balances[msg.sender]);
}
```

Ahmedabad University

```
function depositIBT(uint256 _tokenAmount) external {    infinite gas
    uint256 _ibtAmount = _tokenAmount;
    require(_ibtAmount > 0, "Amount must be greater than 0");
    require(
        mytoken_address.transferFrom(msg.sender, address(this), _ibtAmount),
        "Transfer failed"
    );

    token.mintTokensWithUSD(msg.sender, _ibtAmount);
    ibt_balances[msg.sender] += _ibtAmount;
    ibt_totalDeposits = mytoken_address.balanceOf(address(this));
    if (ibt_balances[msg.sender] == _ibtAmount) {
        ibt_isFirstWithdraw = true;
        ibt_deposit_timestamp[msg.sender] = block.timestamp;
        ibt_accruedInterest[msg.sender] = 0;
        ibt_interest_balances[msg.sender] = _ibtAmount;
    } else {
        ibt_accruedInterest[msg.sender] +=
            ((block.timestamp - ibt_deposit_timestamp[msg.sender]) *
                ibt_balances[msg.sender]) /
            1000000;
        ibt_deposit_timestamp[msg.sender] = block.timestamp;
        ibt_interest_balances[msg.sender] += _ibtAmount;
    }
    console.log(ibt_balances[msg.sender]);
    console.log(ibt_deposit_timestamp[msg.sender]);
    console.log(ibt_accruedInterest[msg.sender]);

    emit Deposit(msg.sender, _ibtAmount, _ibtAmount);
    emit BalanceAfterDeposit(msg.sender, ibt_balances[msg.sender]);
}
```

```
function withdrawIBT(uint256 _amount) external payable{    infinite gas
    uint256 time_now;
    time_now = block.timestamp;
    console.log(ibt_balances[msg.sender]);
    require(ibt_balances[msg.sender] >= _amount, "Insufficient balance");

    if (ibt_isFirstWithdraw) {
        ibt_withdrawInterest[msg.sender] = ((
            ibt_accruedInterest[msg.sender]
        ) +
            ((time_now - ibt_deposit_timestamp[msg.sender]) *
                ibt_balances[msg.sender]) /
            1000000);
        console.log(ibt_withdrawInterest[msg.sender]);
        mytoken_address.transfer(msg.sender,
            _amount + ibt_withdrawInterest[msg.sender]
        );
        ibt_withdrawInterest[msg.sender] = 0;

        ibt_interest_balances[msg.sender] -= _amount;
        ibt_isFirstWithdraw = false;
    } else if (!ibt_isFirstWithdraw) {
        ibt_withdrawInterest[msg.sender] = (((time_now -
            ibt_deposit_timestamp[msg.sender]) * ibt_balances[msg.sender]) /
            1000000);
        console.log(ibt_withdrawInterest[msg.sender]);
        mytoken_address.transfer(msg.sender,
            _amount + ibt_withdrawInterest[msg.sender]
        );
        ibt_interest_balances[msg.sender] -= _amount;
        ibt_withdrawInterest[msg.sender] = 0;
    }

    ibt_deposit_timestamp[msg.sender] = time_now;
    token.burnTokensWithUSD(msg.sender, _amount);
    ibt_balances[msg.sender] -= _amount;
    ibt_totalDeposits = mytoken_address.balanceOf(address(this));
    console.log("Balances Left :");
    console.log(ibt_balances[msg.sender]);

    emit Withdraw(msg.sender, _amount);
}
```

Ahmedabad University

```
function withdrawMATIC(uint256 _amount) public {    infinite gas
    uint256 time_now;
    time_now = block.timestamp;
    console.log(matic_balances[msg.sender]);
    require(matic_balances[msg.sender] >= _amount, "Insufficient balance");
    require((matic_balances[msg.sender]-_amount)*(7)*(MTC_price)/(10**19))>=
    total_borrowed[msg.sender], "Sorry! Collateral depreciated");
    if (matic_isFirstWithdraw) {
        matic_withdrawInterest[msg.sender] = ((
            matic_accruedInterest[msg.sender]
        ) +
            ((time_now - matic_deposit_timestamp[msg.sender]) *
                matic_balances[msg.sender]) /
            1000000);
        console.log(matic_withdrawInterest[msg.sender]);
        payable(msg.sender).transfer(
            _amount + matic_withdrawInterest[msg.sender]
        );
        matic_withdrawInterest[msg.sender] = 0;

        matic_interest_balances[msg.sender] -= _amount;
        matic_isFirstWithdraw = false;

    } else if (!matic_isFirstWithdraw) {
        matic_withdrawInterest[msg.sender] = (((time_now -
            matic_deposit_timestamp[msg.sender]) *
            matic_balances[msg.sender]) / 1000000);
        console.log(matic_withdrawInterest[msg.sender]);
        payable(msg.sender).transfer(
            _amount + matic_withdrawInterest[msg.sender]
        );
        matic_interest_balances[msg.sender] -= _amount;

    }

    matic_deposit_timestamp[msg.sender] = time_now;
    token.burnTokenswithMTC(msg.sender, _amount);
    matic_balances[msg.sender] -= _amount;
    borrowable_amount[msg.sender] =
        (matic_balances[msg.sender] * (MTC_price) * 7) /
        10**19;
    matic_totalDeposits -= _amount+ matic_withdrawInterest[msg.sender];
    console.log("Balances Left :");
    console.log(matic_balances[msg.sender]);
    matic_withdrawInterest[msg.sender] = 0;
    emit Withdraw(msg.sender, _amount);
}
```

```
function borrow_matic(uint256 _amount) external payable {    infinite gas
    require(!matic_isBorrower[msg.sender],
    "You have already borrowed funds! Clear Debt To borrow again!");
    require(matic_totalDeposits>_amount,"Not Enough Funds!");
    require(msg.value>_amount || (borrowable_amount[msg.sender]*(10**18)/MTC_price) >
    _amount,"Provide Collateral To continue the transaction");


    matic_timestamp_borrow[msg.sender] = block.timestamp;
    matic_borrowedAmounts[msg.sender] += _amount;
    total_borrowed[msg.sender] += _amount*(MTC_price)/10**18;
    borrowable_amount[msg.sender] -= _amount*(MTC_price)/10**18;
    matic_isBorrower[msg.sender] = true;
    matic_isFirstRepay[msg.sender] = true;
    matic_totalDeposits -= _amount;
    payable(msg.sender).transfer(_amount);
    emit Borrow(msg.sender, _amount);
}
```

```
function borrow_ibt(uint256 _amount) external payable {    infinite gas
    ibt_totalDeposits = mytoken_address.balanceOf(address(this));
    require(!ibt_isBorrower[msg.sender],
    "You have already borrowed funds! Clear Debt To borrow again!");
    require(ibt_totalDeposits>_amount,"Not Enough Funds");
    require(msg.value>_amount || borrowable_amount[msg.sender]>
    _amount,"Provide Collateral To continue the transaction");


    ibt_timestamp_borrow[msg.sender] = block.timestamp;
    ibt_borrowedAmounts[msg.sender] += _amount;
    total_borrowed[msg.sender] += _amount;
    borrowable_amount[msg.sender] -= _amount;
    ibt_isBorrower[msg.sender] = true;
    ibt_isFirstRepay[msg.sender] = true;

    mytoken_address.transfer(msg.sender,_amount);
    ibt_totalDeposits = mytoken_address.balanceOf(address(this));
    emit Borrow(msg.sender, _amount);

}
```

```
function matic_repay() external payable{    infinite gas
    require(matic_isBorrower[msg.sender],"You have not borrowed any funds!");
    uint256 time_now;
    time_now = block.timestamp;
    console.log("Borrowed at timestamp : ");
    console.log(matic_timestamp_borrow[msg.sender]);

    if(matic_isFirstRepay[msg.sender]){
        matic_repayable_interest[msg.sender] = 0;
        matic_isFirstRepay[msg.sender] = false;
    }

    matic_repayable_interest[msg.sender] +=
    (time_now - matic_timestamp_borrow[msg.sender])*
    matic_borrowedAmounts[msg.sender]/100000;
    console.log("Interest : ");
    console.log(matic_repayable_interest[msg.sender]);

    uint256 total_repayable;
    total_repayable = matic_repayable_interest[msg.sender] +
    matic_borrowedAmounts[msg.sender];

    matic_timestamp_borrow[msg.sender] = time_now;
```

```
    if (msg.value >= total_repayable) {
        matic_isBorrower[msg.sender] = false;
        if (msg.value > total_repayable) {
            // Return Extra funds and collateral
            payable(msg.sender).transfer(msg.value - total_repayable);
            console.log("Sent Back excess Funds and Collateral!");
            console.log(msg.value - total_repayable);
        }
        matic_totalDeposits += total_repayable;
        borrowable_amount[msg.sender] +=
         matic_borrowedAmounts[msg.sender]*(MTC_price)/10**18;
        total_borrowed[msg.sender] -=
        matic_borrowedAmounts[msg.sender]*(MTC_price)/10**18;
        matic_borrowedAmounts[msg.sender] = 0;
        matic_repayable_interest[msg.sender] = 0;
    }
    else{
        matic_isBorrower[msg.sender] = true;
        if(matic_repayable_interest[msg.sender]<= msg.value){
            borrowable_amount[msg.sender] +=
            (msg.value-matic_repayable_interest[msg.sender])*(MTC_price)/10**18;
            total_borrowed[msg.sender] -=
            (msg.value-matic_repayable_interest[msg.sender])*(MTC_price)/10**18;
            matic_borrowedAmounts[msg.sender] -= msg.value-matic_repayable_interest[msg.sender];
            matic_repayable_interest[msg.sender] = 0;
        }
        else{
            matic_repayable_interest[msg.sender] -= msg.value;
        }
        matic_totalDeposits += msg.value;
    }
emit Repay(msg.sender, msg.value);
}
```

Ahmedabad University

```
function ibt_repay(uint256 _amount) external{    infinite gas
    uint256 repay_amnt = _amount;

    require(ibt_isBorrower[msg.sender],"You have not borrowed any funds!");
    require(mytoken_address.transferFrom(msg.sender, address(this), repay_amnt));
    uint256 time_now;
    time_now = block.timestamp;
    console.log("Borrowed at timestamp : ");
    console.log(ibt_timestamp_borrow[msg.sender]);

    if(ibt_isFirstRepay[msg.sender]){
        ibt_repayable_interest[msg.sender] = 0;
        ibt_isFirstRepay[msg.sender] = false;
    }

    ibt_repayable_interest[msg.sender] +=
    (time_now - ibt_timestamp_borrow[msg.sender])
    *ibt_borrowedAmounts[msg.sender]/100000;
    console.log("Interest : ");
    console.log(ibt_repayable_interest[msg.sender]);

    uint256 total_repayable;
    total_repayable = ibt_repayable_interest[msg.sender] + ibt_borrowedAmounts[msg.sender];

    ibt_timestamp_borrow[msg.sender] = time_now;
```

```
    if (repay_amnt >= total_repayable) {
        ibt_isBorrower[msg.sender] = false;
        if (repay_amnt > total_repayable) {
            // Return Extra funds and collateral
            mytoken_address.transfer(msg.sender, repay_amnt - total_repayable);
            console.log("Sent Back excess Funds and Collateral!");
            console.log(repay_amnt - total_repayable);

        }
        ibt_totalDeposits = mytoken_address.balanceOf(address(this));
        borrowable_amount[msg.sender] += ibt_borrowedAmounts[msg.sender];
        total_borrowed[msg.sender] -= ibt_borrowedAmounts[msg.sender];
        ibt_borrowedAmounts[msg.sender] = 0;
        ibt_repayable_interest[msg.sender] = 0;
    }
    else{
        ibt_isBorrower[msg.sender] = true;
        if(ibt_repayable_interest[msg.sender]<= repay_amnt){
            borrowable_amount[msg.sender] +=
            repay_amnt-ibt_repayable_interest[msg.sender];
            total_borrowed[msg.sender] -=
            repay_amnt-ibt_repayable_interest[msg.sender];
            ibt_borrowedAmounts[msg.sender] -=
            repay_amnt-ibt_repayable_interest[msg.sender];
            ibt_repayable_interest[msg.sender] = 0;
        }
        else{
            ibt_repayable_interest[msg.sender] -= repay_amnt;
        }
        ibt_totalDeposits = mytoken_address.balanceOf(address(this));
    }
    emit Repay(msg.sender, repay_amnt);
    }
}
```

Conclusion

- Our project has successfully developed a lending and borrowing protocol on the blockchain, like Aave's functionalities.
- Using Solidity and Hardhat, we've built a robust platform for financial activities, deployed on the scalable and interoperable Polygon blockchain.
- Smart contracts ensure transparent and automated processes, including collateralization and interest rate management.
- Despite challenges, such as interest rate management and currency creation, our project has effectively addressed them.
- In conclusion, our protocol offers a decentralized, efficient solution for users, fostering growth in the DeFi ecosystem.

References

1. Aave. (n.d.). GitHub - aave/aave-protocol: Aave Protocol Version 1.0 - Decentralized Lending Pools. GitHub. https://github.com/aave/aave-protocol

2. Aave/aave-protocol. (n.d.). GitHub. https://github.com/aave/aave-protocol/blob/master/docs/Aave_Protocol_Whitepaper_v1_0.pdf

3. Aave. (n.d.). GitHub - aave/aave-protocol: Aave Protocol Version 1.0 - Decentralized Lending Pools. GitHub. https://github.com/aave/aave-protocol

4. *Getting started with Hardhat: Ethereum Development Environment for professionals by Nomic Foundation*. Hardhat. (n.d.). https://hardhat.org/hardhat-runner/docs/getting-started#quick-start
5. *Introducing the amoy testnet for Polygon pos*. The Value Layer of the Internet. (n.d.). https://polygon.technology/blog/introducing-the-amoy-testnet-for-polygon-pos
6. TESTNET Polygon Pos Chain Amoy (Matic) Blockchain Explorer. (n.d.). https://amoy.polygonscan.com/
7. YouTube. (2024, January 25). *AAVE tutorial (how to Lend & Borrow Crypto on AAVE)*. YouTube. https://www.youtube.com/watch?v=2DfZXOijqcw

THANK YOU!