FTGP-Group5 Project Report

Jinzhou Hu, Junyi Tang, Yuxiang Ge, Yangshu Wang

May 2024

1 Introduction

We created a short blog post that introduces and explains our application at https://medium.com/@geyuxiang2023/introducing-credit-score-lending-platform-43a1be380b29.

Our DApp introduces a new approach to finance by enabling users to lend cryptocurrencies, such as Ethereum, and provide opportunities for borrowing physical items—such as real estate and electronics—without actual physical transactions. Utilizing blockchain technology, the platform eliminates the need for traditional financial intermediaries like banks. This method reduces transaction costs and enhances the security and efficiency of financial exchanges by capitalizing on the inherent benefits of blockchain[5].

An innovative aspect of our platform is the credit scoring mechanism. By using the Etherscan API [9], we fetch and process user transaction history to calculate a credit score, which influences borrowing capacity. This credit score ranges from 300 to 850 points, determined by the total number of transactions and their average value. This ensures that lending decisions are informed and reflect the borrower's reliability.

This report is structured as follows: Section 2 discusses the background, use cases, competitors, and financial aspects of our application. Section 3 covers the design documentation, system architecture, and user interface. Implementation details, system functionality, data structures, and testing strategies are outlined in Section 4, which also includes a user manual. Section 5 evaluates the DApp, focusing on the credit scoring accuracy, potential enhancements, and limitations. The report concludes in Section 6.

2 Background

Decentralized Finance (DeFi) represents a revolutionary shift in the financial industry, aiming to eliminate the need for traditional intermediaries such as banks, exchanges, and brokerages. Instead, DeFi leverages blockchain technology to create a purely code-based financial system. This movement has experienced exponential growth, with the Total Value Locked (TVL) in DeFi platforms surging from $4k to over $175 billion from 2018 to 2021 alone according to DefiLlama [8].

If we take a closer look, the DeFi ecosystem is actually revolving around a set of protocols, or smart contracts. They are deployed and executed on public blockchains primarily Ethereum. Defi protocols not only equip users with the ability to conduct traditional financial activities like borrowing and lending, earning interest, and trading assets, but much faster and more convenient. Crypto assets, which are built upon blockchain, has continuously been gaining traction as the growth of Defi, and it is becoming a widely used financial instrument rather than a simple payment tool. The approval of spot Bitcoin Exchange-Traded Funds(ETFs) by both the Securities and Exchange Commissions (SECs) in the U.S. in January [11] and in Hong Kong a couple of weeks ago [7], combined with the continuous promotion on social media by Key Opinion Leaders (KOLs), entrepreneurs, and politicians, has increased mainstream adoption

and liquidity in the crypto assets market. These assets are now actively traded on exchanges and can also be used as collateral in financial transactions.

DeFi lending is where users borrow and lend cryptocurrencies without the need for traditional financial intermediaries. It is facilitated through smart contracts that specifying the terms of the loan, including interest rates and collateral requirements. There are currently two main types of Defi lending platforms: liquidity pool-based and peer-to-peer (P2P) based [6].

AAVE and Compound are the two largest liquidity pool-based platforms on the market [10]. Lenders can deposit assets they are willing to lend in exchange for interests which are collected into a liquidity pool, borrowers can then withdraw them by taking out a loan. P2P lending platforms, on the other hand, agreement on loans between two parties is backed by smart contracts. Lenders have the opportunity to set their own terms and interact directly with borrowers, making the lending experience more personalized. Therefore, P2P lending platforms often provides more appealing interest rates for both parties instead of the fixed rates on liquidity pool-based platforms.

Our DApp introduced a credit scoring mechanism that generates a credit score based on the evaluation through users' transaction history. This mechanism is important in Defi as it supplies lenders with a trustworthiness measuring instrument of borrowers. Traditional credit systems - like Ant Financial's Sesame Credit [4] which uses data such as transaction history, behavior patterns, and social relationships to evaluate creditworthiness - have paved a way for the scoring functionality, our DApp extended it and made it more transparent and more secure through the blockchain technology.

Our platform not only supports crypto lending but also includes a unique feature named 'Share Talk' allowing the lending and borrowing of physical items. Users can upload images and detailed descriptions of items they wish to lend or borrow and can communicate through an integrated chat system. This feature broadens the DeFi lending model beyond digital assets, promoting efficient utilization of physical assets.

Similar applications existed in the market include platforms like YouLend and Fat Llama [2, 1], which enable P2P lending of physical items but are centralized. Our product on the other hand, enhances security and transparency by leveraging blockchain technology, thereby mitigates the potential risks or unpleasantness associated with centralization. The last thing we are proud to share, is that our platform has the ability and potential to promote financial inclusion by extending financial services to regions where traditional banking infrastructure is scarce or unavailable. If we were further supported by experts and funded by investors, our DApp could make a substantial impact.

3 Design Documentation

3.1 Introduction about design choices

This DApp project serves as a lending platform where users can maximize the use of idle resources such as Ethereum and real estate. The architecture is designed to facilitate dual lending functionality.

Firstly, the crypto lending feature allows borrowers to request loans at given interest rates and market conditions, using higher amounts of ETH or stablecoins as collateral. This ensures stable transactions, reducing the risk associated with cryptocurrency volatility. Secondly, we created a platform called 'Share Talk' for the physical item lending market. Lenders can post requests with images and descriptions of items, essentially renting them out for a period. Additionally, we provide a chat feature, operating as a communication platform between lenders and borrowers. Furthermore, we developed a credit scoring feature using a script that calls the Etherscan API to fetch and process user history data. It calculates the number of past

transactions and their average value to determine credit scores, which are critical as they limit borrowing capacity.

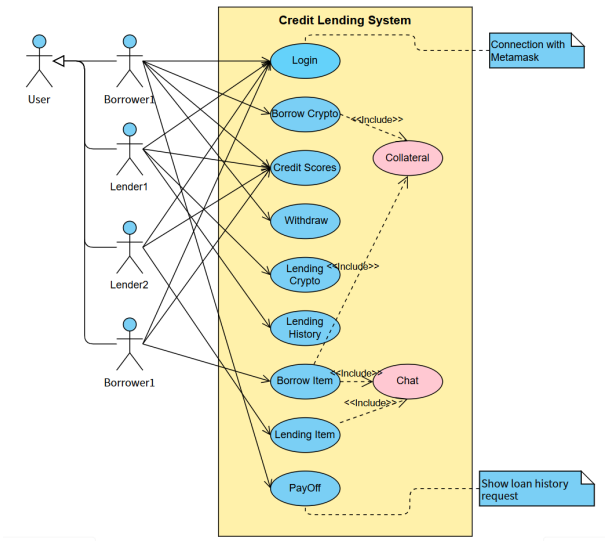

Figure 1: Use Case Diagram

3.2 Use Case Diagrams

The use case diagrams Figure 1 illustrate the main functionalities of our lending platform and the relationships between the participants. The platform primarily offers lending and borrowing functionalities, specifically for cryptocurrency and physical items, with integrated chat features for communication. The participants in the use case diagrams include two pairs of users (borrowers and lenders), who connect their Metamask accounts via the login feature.

Borrower1, with a given credit score, can borrow cryptocurrency by providing collateral, withdrawing funds, and eventually paying off the loan. Lender1 facilitates the lending of cryptocurrency and receives the repayment with interest when Borrower1 completes the payoff.

For physical items, Borrower2 can post descriptions and requirements for items they wish to borrow. Lender2, upon finding a suitable match, can initiate a chat conversation with Borrower2 to negotiate the terms of the loan.

3.3 Overall Process Introduction

The overall process, as shown in Figure 2, is divided into two main parts: the Crypto Lending Platform and Shared Talk. I will explain each part in detail.

3.3.1 Qualification Assessments

In both the Crypto Lending Platform and Shared Talk Platform, before sending a loan or borrowing request, the Borrower must ensure they have no outstanding balance on the platform. If they do, the request will be denied. Otherwise, they can proceed to the next step.

3.3.2 Credit Scores System

Users who want to borrow money log into Metamask and submit their wallet address to CreditScores.js. By using Etherscan's API, the code retrieves and processes the user's transaction

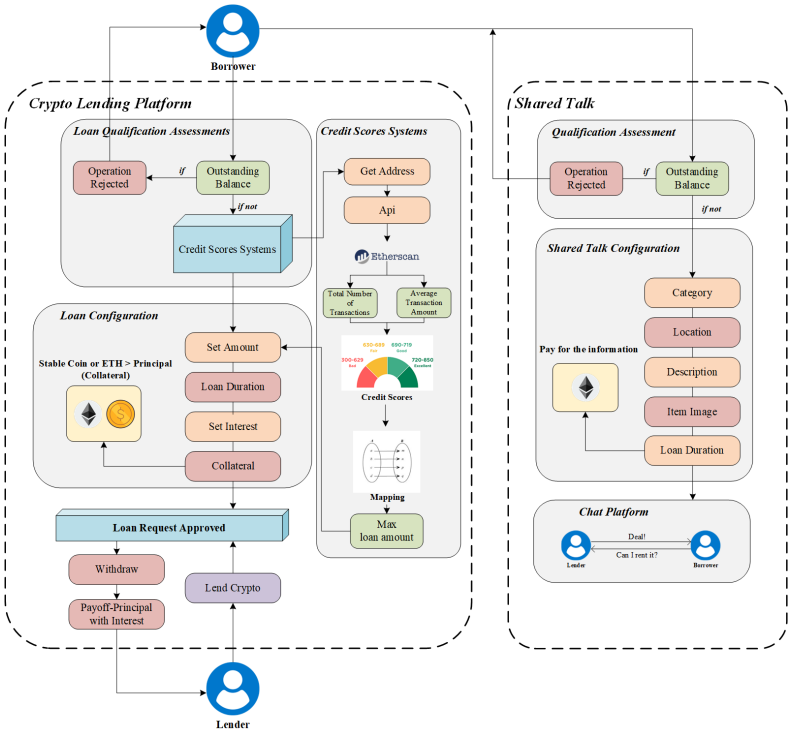

Figure 2: Flow Diagram

history to determine the total number of transactions and average transaction amount, converted into ETH. Based on these variables, CreditScores.js generates the user's credit score.

The credit score ranges from 300 to 850 points, varying with the user's transaction history and average transaction amount. The initial score is 300 points, increasing with The number of transactions and average transaction amount. The specific calculation formula is shown in Equation 1:

$$\begin{aligned} \text{credit score} = 300 &+ \min\left(\log_{10}(1 + \text{totalTransactions}) \times 100, 300\right) \\ &+ \min\left(\log_{10}(1 + \text{averageAmount}) \times 1000, 250\right) \end{aligned} \tag{1}$$

After calculating the score, it is mapped to determine the maximum loan amount. For scores below 580, the maximum loan is 0.5 ETH. For example, if the user's score is 600 points, the loan limit is approximately 0.72 ETH. The mapping between maximum loan amount and credit score is a piecewise function. Figure 3 shows the function graph of credit score, which increases with the number of transactions and the average transaction amount, and the increase gradually slows down until it reaches its maximum value. This credit score mechanism is also designed to simulate reality.

3.3.3 Configuration Stage

For the Crypto Lending Platform, Borrowers set the loan amount, duration, and interest based on their Credit Scores within the given limit. They then provide collateral. Most DeFi lending platforms require over-collateralization, meaning the value of the collateral must exceed the loan amount to mitigate cryptocurrency price volatility. For example, Aave has an LTV ratio of 80%, meaning you can borrow up to 80% of the value of the provided ETH. Due to our Credit Scores mechanism, the loan limit is predefined, so the collateral value must be at least equal to the loan value. This can be in ETH or USD-pegged stablecoins, but we use ETH for simulation.

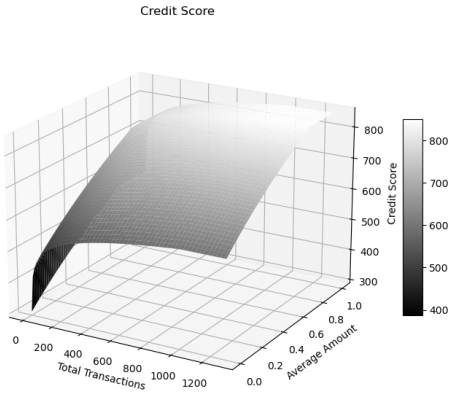

Figure 3: Credit Score Mechanism

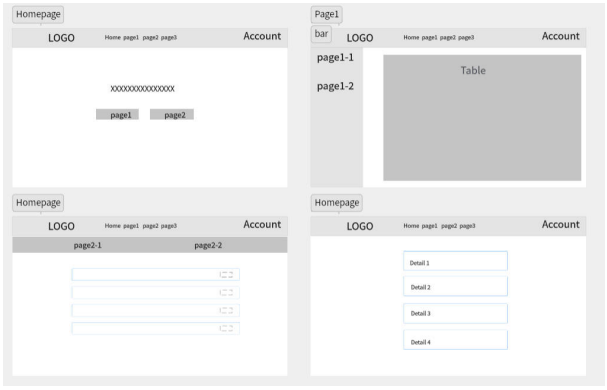

Figure 4: Wireframe Diagrams

For the Shared Talk Configuration, users select the item category (e.g., real estate, electronics, others), provide the address, description, desired rental period, and upload an image of the desired item. A nominal fee is charged for this service as a platform revenue model.

3.3.4 Request Approval

When borrowers want to take out loan, they can log in to the Borrow Crypto interface, then they need to set a total loan duration in months and an interest percentage. Borrowers also need to enter an ETH collateral that is not less than the loan amount. Then these requests will be approved.

In the Crypto Lending Platform, Lenders view the Borrower's loan request details, including Borrower's Address, Amount Needed, Loan Duration, Interest Percentage, Return Amount, Amount Raised, and Amount Remaining. Lenders can send the required amount to the specified address. Borrowers can then withdraw the funds. In the Payoff interface, Borrowers see the Lender's address, repayment amount, and time. Borrowers can repay the loan early, including the principal and interest, to the Lender. After that, ETH collateral is returned to borrower's wallet.

For Shared Talk, Lenders who have items that match the Borrower's request can initiate a chat to negotiate terms. Since physical items are involved, the platform provides a space for communication rather than transaction guarantees, facilitating information exchange and potential offline transactions.

3.4 Design of front-end

In the frontend development of this project, based on the functional points, as shown in Figure 4, there are four main pages, including a main page and three sub-pages. The following describes the design according to different modules.

Navbar: The navbar consists of three sections. The logo is displayed in the first section. The second section is for page switching, and on the far right is the logged-in user information.

Home Page: This page is divided into two parts. The first part introduces the core functionality of this Dapp to attract users. The second part contains a toggle button for switching between borrowing and repaying money.

Page 1: This page is used to display published loan request information and is divided into two sub-pages. The first sub-page is for Ethereum lending, including the basic published information. The second sub-page is for Shared Talk, which includes basic information and an entry point to the chat page.

Page 2: This page is used to publish lending information and is also divided into two sub-pages. The first sub-page is for publishing Ethereum lending, with input fields for basic information. The second sub-page is for publishing NFT lending, including basic information input fields.

Page 3: This page displays the borrowed information and includes a repayment button.

4 Project Execution

The architecture of our dApp uses the React framework for the front-end, Hardhat for back-end development, and is programmed in Solidity version 0.8.4. The operation of our system is depicted in Figure 8, with detailed data structures also provided. Our explanation is divided into two main sections: 'Crypto Lending Platform Execution' and 'Shared Talk.' Final user interface elements can be viewed in Figure 5, showcasing screenshots of the actual application.

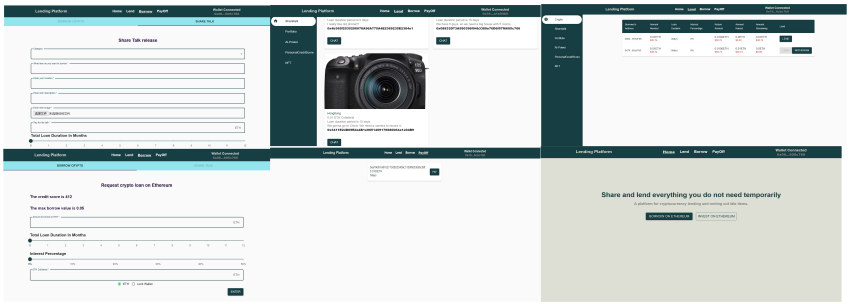

Figure 5: The Final UI Screenshots

4.1 Data Structure

The data flow diagram in Figure 6 illustrates the interactions between borrowers, lenders, and the smart contract within our DApp. It showcases the process of creating loans, providing collateral, lending funds, repaying loans, and withdrawing funds for both crypto and physical items.

4.2 Crypto Lending Platform Execution

Before a borrower can initiate a loan, they must undergo a qualification assessment. In the smart contract, we have implemented a function called `checkIfBorrowedBefore`. This

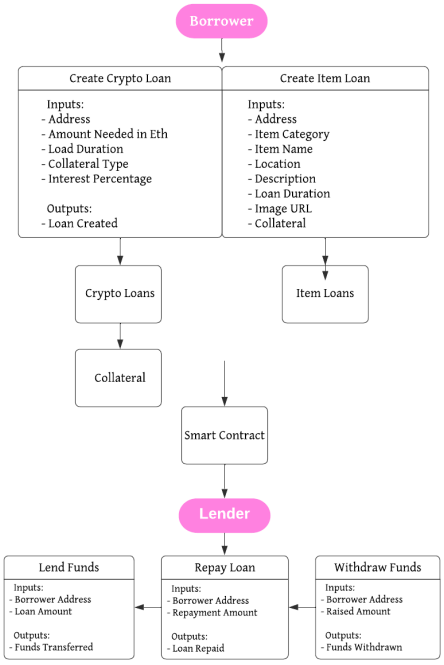

Figure 6: Data Flow Diagram

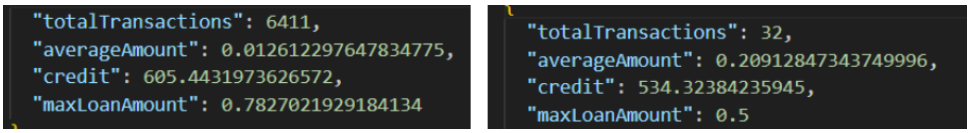

Figure 7: Calculate Credit Score Case

function checks the borrower's address against a list of all addresses with outstanding loans stored in the `borrowers` array. If the address is found, the function returns the message 'You have an outstanding loan'. If not, the borrower can proceed to send the loan request to the `createCryptoLoan` function.

The platform uses the Express framework and Etherscan API to process user data. It retrieves historical transaction data, calculates credit scores using a binary function based on transaction times and amounts, and determines the maximum loan amount through a segmented function. This data is stored, and the loan amount is displayed on the front-end.

Credit scores influence loan amounts. For example, scores of 605 and 534 allow for maximum loans of 0.78 ETH and 0.5 ETH, respectively, as shown in Figure 7. Borrowers must provide collateral, recorded in the `cryptoBorrower[collateral]` mapping.

Lenders can choose to fund loans, after which borrowers can withdraw funds using the `withdrawFunds` function. Loans can be repaid at any time during the term via the `cryptoRepay` function, which updates the loan status, handles fund transfers, and returns the collateral to the borrower. The lender then receives the principal and interest.

4.3 Shared Talk Execution

As shown in Figure 8, Shared Talk provides a platform for renting idle items with a simple contract design. First, a qualification assessment is conducted. If the borrower has outstanding loans, they cannot post rental items and will receive the message: "You have an outstanding loan!". If they pass, the borrower can upload an item using the Upload Image component on

the front end.

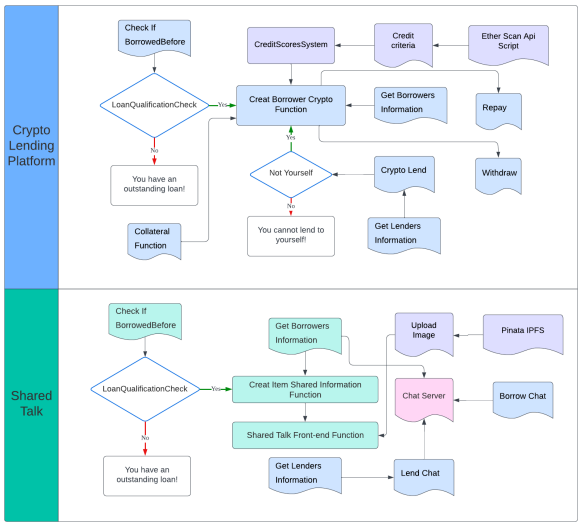

Figure 8: System Functions Diagram

Images are stored using IPFS for its decentralized and immutable storage benefits. We chose Pinata as our third-party IPFS storage provider since Infura does not support new users. We created a project on Pinata and imported the corresponding API URL into our project.

The stored image, along with other details such as description and location, is uploaded to the contract using the `GetBorrowersInformation` function. The front end retrieves this information and displays it on pages 1-2, as shown in Figure 4. When a lender finds a suitable item, they can use the chat button designed with React to enter our chat platform.

The chat functionality is implemented using React and Gun.js for decentralized and real-time messaging. Gun.js handles data storage and synchronization, while React manages state and UI rendering. Redux is used to obtain user addresses, ensuring each message correctly displays the sender's information. However, Gun.js is not supported by Vercel, which only supports static sites, so the front-end deployment on the web cannot fully utilize this feature. This part will be demonstrated offline during the presentation, with a full explanation of the principles.

4.4 Testing Strategy

Our team uses two main testing methods: unit testing and integration testing.

For unit testing, we use Jest and React Testing Library to test front-end components, including rendering, event handling, state management, and Redux integration. For example, we write tests to verify that a list component displays data based on Redux state and that user actions, such as button clicks, trigger the expected behavior.

For Solidity smart contracts, we first use Remix for quick validation of function behaviors with given inputs. We then perform comprehensive unit tests using Hardhat. In the `test` folder, we write JavaScript test scripts to automatically deploy contracts, execute functions, and verify state changes and event emissions. For example, we test whether a payment function correctly updates the contract's balance.

For integration testing, we create test files in the `test/` directory targeting specific components. One test case might simulate a user initiating a transaction from the front-end, with the back-end calling the smart contract's function. After execution, the results are returned to the back-end and displayed on the front-end. We verify each step to ensure correct data flow and seamless component integration.

4.5 User Manual

This manual guides end-users on how to effectively utilize our decentralized lending platform, which facilitates the use of idle resources such as Ethereum and real estate for lending purposes. The platform incorporates features for both cryptocurrency and physical item lending, along with a communication system for interaction between parties.

Platform Login

Open our DApp website. Click on ”Connect Wallet” at the top right of the homepage. Select MetaMask, authorize the connection to your wallet.

Cryptocurrency Lending

Navigate to the ”Borrow on Ethereum” section from the homepage. Click on the ”Borrow Crypto” button. Enter the desired loan amount, duration, and interest rate. The system automatically calculates the allowable loan amount and required collateral based on your credit score. Confirm that the collateral value is at least a specific percentage higher than the loan amount, Submit the loan application. Once your loan is fully funded by other users, funds can be withdrawn from your account from the ”Crypto” section under ”Lend”.

Physical Item Lending

In the ”Borrow” interface, select the ”Physical Item Lending” option. Fill in the category of the item (e.g., real estate, electronics), provide a detailed description, the desired rental period, and upload an image of the item. Pay for the message publication and submit the lending request. In the ”Shared Chat” section of the ”Lend” interface, lenders interested in your item can contact you via the platform’s chat feature to discuss and negotiate loan terms. Once terms are agreed upon, transactions can be completed offline as necessary.

Loan Repayment

Prior to the end of the loan term, log in and navigate to the ”Repay Loan” interface.Enter the amount you wish to repay, including both principal and interest. Upon completion of payment, the collateral is returned to your wallet, and the loan status is updated to ”Repaid.”

Note: Ensure that you fully understand all terms and conditions before engaging in any transactions or lending activities. For physical item lending, it is advisable to have a written contract to secure the transaction.

5 Critical Evaluation

Credit scoring accuracy hinges on robust data input and modeling[3]. Currently, our mechanism primarily uses transaction frequency and average amount, but this approach could be refined by incorporating specific demographic data, such as new user status or previous defaults. Furthermore, the existing model, represented in Figure 3, may not capture all nuances. Future models could leverage AI-powered deep learning with comprehensive datasets to enhance precision and fairness in credit scoring[12].

At present, although the lending platform will charge a certain amount of ETH collateral to borrowers, more effective punishment measures have not been established, such as reducing the borrower’s credit score and using special symbols to mark the borrower. In this way, other lenders can make certain references when lending to borrowers. The future lending platform model can introduce more tiered punishment measures for overdue repayments.

6 Conclusion

In this project, we developed a DApp leveraging blockchain technology to create a platform for lending and borrowing both cryptocurrency and physical assets. Our platform maximizes the use of idle resources, including Ethereum and physical items like real estate and electronics.

Smart contracts reduce transaction costs, enhance security, and provide financial services to regions lacking traditional infrastructure, promoting financial inclusion and the sharing economy.

Our dApp introduces two innovations. Firstly, we build a script to access historical transaction data via the EtherScan API for calculating credit scores. These scores are then used to set borrowing limits. Secondly, we have established a ”Shared Talk” platform, which facilitates the rental of physical items. This platform allows users to share images and descriptions of items and engage in conversations to facilitate exchanges.

One limitation of our dApp is the simplicity of the credit scoring model. In the future, this model could be improved by incorporating additional user information, such as past due records, to enhance its accuracy. Our penalties for overdue or defaulted loans are also insufficient. Future versions could introduce comprehensive penalty measures and legally binding contracts for physical transactions.

Overall, our decentralized lending platform showcases blockchain technology's potential in finance. With further development, it can provide safe, efficient, and inclusive financial services to a broader range of users.

References

[1] Fat llama. `https://fatllama.com/`. Accessed: 19 May 2024.

[2] Youlend. `https://youlend.com/`. Accessed: 19 May 2024.

[3] Hussein A Abdou and John Pointon. Credit scoring, statistical techniques and evaluation criteria: a review of the literature. *Intelligent systems in accounting, finance and management*, 18(2-3):59–88, 2011.

[4] Ant Financial. Ant financial unveils china's first credit-scoring system using online data. 2015. Retrieved 20 May 2024.

[5] Imran Bashir. *Mastering Blockchain*. Packt, fourth edition, 2023.

[6] Jonathan Chiu, Emre Ozdenoren, Kathy Zhichao Yuan, and Shengxing Zhang. On the fragility of defi lending. *SSRN*, November 30 2022. Available at SSRN: https://ssrn.com/abstract=4328481.

[7] CryptoSlam. Hong kong to debut bitcoin and ethereum etfs on april 30, April 2024. Accessed: 2024-05-20.

[8] DeFiLlama. Defillama. `https://defillama.com/`. Accessed: 2024-05-19.

[9] Etherscan. Etherscan documentation. `https://docs.etherscan.io/`. Accessed: 2024-05-19.

[10] Emilio Frangella and Lasse Herskind. Aave v3 technical paper. 2022.

[11] Gary Gensler. Statement on the approval of spot bitcoin exchange-traded products, January 2024. Accessed: 2024-05-20.

[12] Björn Rafn Gunnarsson, Seppe Vanden Broucke, Bart Baesens, María Óskarsdóttir, and Wilfried Lemahieu. Deep learning for credit scoring: Do or don't? *European Journal of Operational Research*, 295(1):292–305, 2021.

7 Appendix

Final distribution of equity shares:

Equity share: Jingzhou Hu (1), Junyi Tang (1), Yuxiang Ge (1), Yangshu Wang (1)

Our GitHub project link is: https://github.com/HjzQAQ/FTGP23Group505/tree/main.

(The link must be clickable directly without copying because LaTeX does not support underscores, so the underscores in the above link are omitted.)

7.1 Sprint Report 1

FTGP Group [5]: Sprint 1 Report (22nd March)

Team Members:

- Jingzhou Hu
- Junyi Tang
- Yuxiang Ge
- Yangshu Wang

Product Vision:

Our DApp aims to leverage the transparency and security of blockchain technology to develop an innovative decentralized application that ensures user data safety and transaction transparency while providing a user-friendly, efficient online platform.

We have considered various product ideas including Token Exchange, Clinical Practice Based, Musical/Artist Copyright related, Fitness DApp, and even considered Dating DApp… Yet, in the end only two main applications stand out and were considered both innovative and viable: one is a second-hand trading market similar to eBay and Xianyu, allowing users to list items for sale on the blockchain and conduct transactions using cryptocurrency (most likely Ether) due to our backend development with Solidity. The second is a ticketing platform akin to Ticketmaster or Viagogo, focusing on offering a reliable, decentralized environment for ticket transactions, which includes direct sales of primary tickets and the resale of secondary tickets. Both ideas aim to explore how blockchain technology can create real-world value while providing a secure, transparent, and decentralized trading platform for users.

However, given that our initial session was a preliminary brainstorming meeting, and our commitment is towards developing an optimal Dapp, we have not settled on a final theme yet. The ultimate decision will be shaped by our collective individual brainstorming in the forthcoming weeks. We decided on assigning each of us a role of coming up with at least three different types of both innovative and viable DApp ideas associated with at least three different user stories each, and we aim to achieve this goal approximately two weeks from now, around April 5th, when we plan to converge our ideas and finalize the theme that will guide our development efforts.

Sprint 1 Planning (Sprint duration 22nd March – 19th April):

This should summarize your sprint planning meeting. The meeting should be done at the beginning of each sprint. You must specify your sprint vision and select which items from the product backlog you plan on completing during the next sprint (sprint backlog). Additionally, you must select the product owner and the scrum master.

- Product Owner: Jingzhou Hu, responsible for defining the product direction, managing the product backlog, and ensuring that the team understands and executes the tasks.
- Scrum Master: Jingzhou Hu, responsible for ensuring the team follows agile and Scrum practices, assisting team members in removing impediments, and facilitating communication and collaboration within the team.

Sprint Vision:

As mentioned in the Product Vision, our plan for the next two weeks is to come up with different ideas on the theme of our DApp and gather to discuss and settle on the final one. As our next sprint is after the Easter Break on Apr 19, we aim to finish brainstorming on Apr 5, and commence the development of our MVB until Apr 19. It would be ideal that we can deliver the MVB by Apr 19. This is just an initial plan which we can adjust according to our real progress.

Sprint Backlog:

- Phase 1 (Mar 18 – Apr 5): Brainstorming
  - Each team member should generate at least three innovative and viable DApp ideas, along with at least three user stories for each idea. We should then individually conduct market research on existing DApps in similar domains to ensure the uniqueness of our ideas. We should also investigate the technical feasibility, potential user base, and market demand for each proposed idea. This step is crucial for us to have a diverse range of feasible while innovative options to choose from.
  - Schedule collaborative meetings (probably around Apr 5) to discuss and evaluate the generated ideas. These sessions will be instrumental in narrowing down the choices to the most promising ones. We should engage in activities or structured discussions to reach a consensus on the final theme. This might involve voting, weighing pros and cons, or other decision-making strategies.
- Phase 2 (Apr 5 – Apr 19): Initial Development
  - Once the theme is decided, we should begin outlining the core features and functionalities of the chosen DApp idea, aligning with the selected user stories. In the meantime, we should explore existing literature to uncover any similar DApp implementations, understanding their architecture, features, and limitations, and gain insights to inform our own design decisions.
  - Next, we should prepare the development environment, tools, and access rights necessary for starting the MVB development.
  - We should be developing and aim to deliver the MVB ideally by Apr 19.

Anything else you would like to share:

Given that we are yet to finalize the theme of our DApp, our comprehensive Product Backlog will be developed in the subsequent meeting. This future session will be crucial as we'll outline the specific tasks, features, and objectives that the team will focus on in the upcoming Sprint. This approach ensures that our Product Backlog is directly aligned with our finalized project theme and provides a clear, actionable roadmap for our development efforts moving forward.

7.2 Sprint Report 2

FTGP Group [Group Number]: Sprint 2 Report (19th April)

Sprint 1 Review (Sprint duration 22nd March – 19th April):

This should summarize your sprint review meeting. The meeting should be done at the end of each sprint. You need to identify which tasks from your sprint backlog were completed, which were altered, and which were not completed. Please also use this process to reflect and improve your next sprints.

Completed work, i.e. which tasks were complete and by who:

During Sprint 1, our team divided the work into two main phases. We completed brainstorming on April 5 and then proceeded with the development of our MVB until April 19.

- During the brainstorming session, we considered various sectors within the financial domain that involve transaction-based DApps, such as financing, consumer spending, lending, and gambling. Each team member generated at least three innovative and viable DApp ideas, along with at least three user stories for each idea. We ultimately decided to develop a P2P lending platform. As a distinctive feature, we introduced a credit scoring mechanism as our innovation. This mechanism scores users based on the number of historical transaction events, transaction values, and payment delinquencies, which then adaptively adjusts their borrowing limits. This decision was collaboratively finalized by all team members.

- For the development of our MVB, we implemented the basic front-end structure using React and CSS, a task accomplished by Yuxiang Ge. We also implemented simple transaction functionality, allowing different accounts to conduct mortgage operations under varying interest rates, which was integrated with the corresponding front-end components. This part was jointly implemented by Junyi Tang and Jingzhou Hu. Additionally, the establishment of the credit scoring criteria was carried out by Yangshu Wang.

Changes, i.e. tasks that have changed/not completed and why:

We changed our project during Sprint 1 to develop a lending platform based on a credit scoring system. Here's the detailed process and reasons for this change:

During Sprint 1, we initially considered two project ideas: a blockchain marketplace similar to eBay, enabling cryptocurrency transactions, and a decentralized ticketing platform like Ticketmaster for secure ticket sales and resales. Our brainstorming session revealed that both the dating app and the eBay-like platform would require databases with a significant amount of non-textual information, primarily images. Storing such image data directly on the blockchain would be inefficient and could compromise decentralized privacy protections due to the blockchain's limitations on handling large data volumes.

Additionally, our team members expressed a strong interest in developing a DApp within the financial sector. Out of twelve potential options, we decided to pursue a lending platform based on a credit scoring system. This credit scoring system is intended to be our main innovation. We plan to leverage the Etherscan API to access a user's recent transaction history, including details like transaction amounts and timestamps. Using our proprietary scoring system, we will assign a base credit score to users, which will be adjusted downwards in cases of defaults, such as delayed repayments. This adjustment will directly impact the users' borrowing limits and could affect their collateral. This pivot in our project focus underscores our adaptation to the practical challenges and technological constraints encountered during the initial development phase.

Agreed weekly "Equity share", i.e. how this sprint's work was split:

- Equity share: Jingzhou Hu (1), Junyi Tang (1), Yuxiang Ge (1), Yangshu Wang (1)

As mentioned in Sprint 1 Review, each member of our team had clear responsibilities. Jingzhou and Junyi were in charge of developing the smart contracts and corresponding front-end integration. Yuxiang was responsible for the front-end development, and Yangsu was tasked with establishing the standards for the credit mechanism.

Sprint 2 Planning (Sprint duration 22nd – 26th April)

This should summarize your sprint planning meeting. The meeting should be done at the beginning of each sprint. You must specify your sprint vision and select which items from the product backlog you plan on completing during the next sprint (sprint backlog). Additionally, you must select the product owner and the scrum master.

- Product Owner: Jingzhou Hu, responsible for defining the product direction, managing the product backlog, and ensuring that the team understands and executes the tasks.

- Scrum Master: Jingzhou Hu, responsible for ensuring the team follows agile and Scrum practices, assisting team members in removing impediments, and facilitating communication and collaboration within the team.

Sprint Vision:

During Sprint 1, we have completed the development of the lending function and the borrowing function, including the collateral and the corresponding front-end components, and have conducted tests. This part was completed before April 19. Moving forward, we will focus on developing the credit scoring system, which will be divided into three phases: feasibility validation, development, and testing. These phases are scheduled to be completed before April 20, April 22, and April 24, respectively. This is just an initial plan, which we can adjust according to our real progress.

Sprint Backlog:

- Phase 1 (Apr 18 – Apr 20): Feasibility validation
  - After our preliminary research, we've determined that it's not feasible to directly access transaction records from MetaMask because it does not offer an API for this functionality. Instead, we propose using the Etherscan API to retrieve recent transaction details of an account, such as transaction times and values, and then return this data to the blockchain. We need to write a simple smart contract to verify the feasibility of this function. Junyi will primarily handle this part. On April 18th, we will meet offline to further discuss and refine the specific plans for the credit system.
- Phase 2 (Apr 20 – Apr 22): Design & Development
  - After completing our verification, we have outlined two preliminary design options. The first option involves using manually set criteria to weight transaction records, such as awarding points for large transactions and frequent transactions with high ratings. These criteria are provisional and may be refined later. The second option, which is still under discussion, involves integrating AI to provide intelligent scoring. This would entail building our own dataset to train a deep learning model, then deploying the model via Chainlink to perform API calls and push the data results to the blockchain. The development of this dataset and further specifics will be discussed among our four team members.
  - We will proceed with the development based on Option 1, focusing on building the framework and making improvements while ensuring completeness. Specifically, Junyi and Jingzhou will handle the development of this part of the contract. The front-end page development will be conducted by Yuxiang.
- Phase 3 (Apr 22 – Apr 24): Testing
  - On April 24, we will conduct offline discussions and platform testing. By the end of this phase, our platform will have implemented several key features:
  - The ability to offer loans with varying interest rates and durations.
  - The functionality to handle collateral.
  - The implementation of an intelligent credit scoring system.

We have sketched out the mechanism of our lending platform, where users are categorized as Borrowers and Lenders, each handling Loan Review and Loan Settings, respectively. A key innovation in our platform is the implementation of Credit Scores. We are considering two options: Option 1 involves manually setting criteria for scoring, and Option 2 involves building our own dataset to train a model, which would be encapsulated and integrated using Chainlink

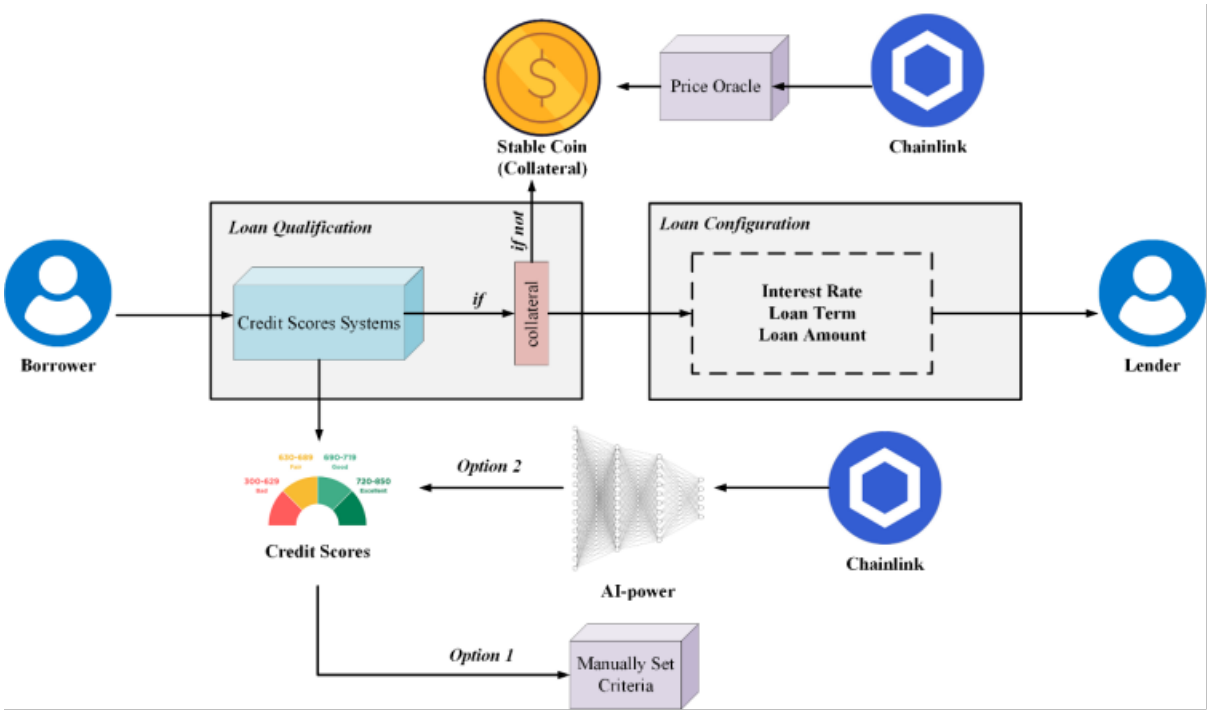

Figure 9: Sprint 2 Diagram

to import API results. Additionally, we have opted to use a stable currency (pegged to the US dollar) as collateral.

The above is a brief demonstration and description of our platform's functionalities. We would greatly appreciate any feedback.

7.3 Sprint Report 3

FTGP Group 5: Sprint 3 Report (26th April)

Sprint 2 Review (Sprint duration 22nd – 26th April):

This should summarize your sprint review meeting. The meeting should be done at the end of each sprint. You need to identify which tasks from your sprint backlog were completed, which were altered, and which were not completed. Please also use this process to reflect and improve your next sprints.

Completed work, i.e. which tasks were complete and by who:

During Sprint 2, we mainly completed the following work:

- Front-end development, developed a specific executable interface, and added a new page related to credit score in the front-end, optimize the payoff function of the front-end, which was mainly carried out by Yuxiang Ge.
- Verify the feasibility of obtaining users' historical transaction records through Etherscan website, and carry out data cleaning of historical transaction records, test the interface for front-end borrow, which is carried out by Yangshu Wang.
- According to the historical transaction record of the cleaned user, the credit score corresponding to the user is given through certain logic and specific algorithm, test the functionality of the front end collateral, which is carried out by Junyi Tang.
- Deliver the credit score to the smart contract, set different loan amount ceilings for credit scores of different scores, and remind when the upper limit is exceeded, test the front-end lending interface, which is carried out by Jingzhou Hu.

The above also involves the transfer of data between the front-end interaction and the back-end, which is done by everyone.

Changes, i.e. tasks that have changed/not completed and why:

The function of front-end interaction still needs to be improved. At present, the display of credit score in front-end is still controversial, and the controversial point is whether to display the credit score of each user and the update logic of credit score. Instruction to send wallet addresses from the front-end to the back-end are not implemented for the time being.

Agreed weekly "Equity share", i.e. how this sprint's work was split:

- Equity share: Jingzhou Hu (1), Junyi Tang (1), Yuxiang Ge (1), Yangshu Wang (1)

As mentioned before, there is a clear division of labor for each group. Yuxiang Ge was responsible for the development and update of the front-end interface; Jingzhou Hu, Junyi Tang, and Yangshu Wang focused on the development of the back end of the credit scoring mechanism and related front-end interactive testing and data transmission.

Sprint 3 Planning (Sprint duration 29th April – 3rd May)

This should summarize your sprint planning meeting. The meeting should be done at the beginning of each sprint. You must specify your sprint vision and select which items from the product backlog you plan on completing during the next sprint (sprint backlog). Additionally, you must select the product owner and the scrum master.

- Product Owner: Jingzhou Hu, responsible for defining the product direction, managing the product backlog, and ensuring that the team understands and executes the tasks.
- Scrum Master: Jingzhou Hu, responsible for ensuring the team follows agile and Scrum practices, assisting team members in removing impediments, and facilitating communication and collaboration within the team.

Sprint Vision:

During Sprint 2, we have completed the feasibility verification of obtaining user historical transaction data from the Etherscan website, implemented a basic credit scoring mechanism based on historical transaction data, and set a maximum loan amount for users based on credit scores. And the backend testing of these contents has been completed, while the frontend is also continuing to improve functionality and optimize interaction. Next, the focus will be on adding appropriate interface interactions regarding the credit scoring mechanism in the front-end and transmitting address instructions from the front-end to the back-end, and running and testing the entire project after completing the above functions.

Sprint Backlog:

- Phase 1 (April 29th to May 1st): Coordination testing between front-end and back-end
  - After previous development, we have preliminarily completed the backend work of the credit scoring mechanism. Next, we need to implement and test the transmission of instructions between the front end and back-end, including sending wallet addresses from the front-end to the back-end, and transmitting credit scores and loan amount limits from the back-end to the front-end. And it is necessary to add prompts on the upper limit of the loan amount on the front-end interface. Prior to this, we had already developed the most basic logic of the credit mechanism. On April 29th, we will hold an offline meeting to further discuss the rationality of the established credit mechanism and its interaction with the front-end.
- Phase 2 (May 1st to May 3rd): Operation and testing of the lending platform

– We have added a credit scoring mechanism to the lending platform, which includes functions such as borrowing and repayment, collateral, credit scoring, etc. At this stage, further testing of these functions will be conducted, with a focus on testing the stability and reliability of the lending platform.

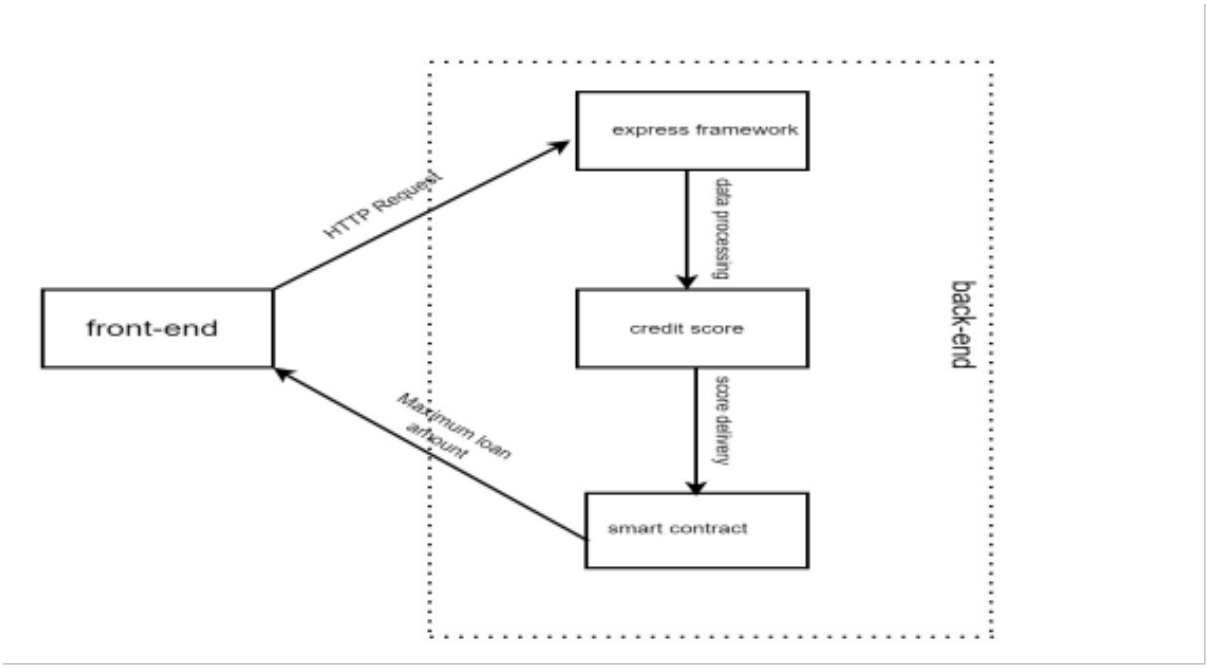

Figure 10: Sprint 3 Diagram

As shown in the figure, this is our idea of adding a credit scoring mechanism to the lending platform. When a user wants to borrow money, an HTTP request will be sent from the front-end to the back-end, which includes the user's wallet address. After processing by the back-end, the credit score of the user will be generated, and the information of the maximum loan amount will be returned to the front-end. If the user's loan does not exceed the maximum loan amount, the next step can be taken, and if the user's credit score is insufficient, the loan will fail. Hope to get your feedback and suggestions!

7.4 Sprint Report 4

FTGP Group 5: Sprint 4 Report (3rd May)
Sprint 3 Review (Sprint duration 29th April – 3rd May):
Completed work, i.e. which tasks were complete and by who:
During Sprint 4, our main work was to perfect and upgrade the project in development. The main tasks are as follows:

- Beautify the front-end pages, including more detailed adjustments to layout and colors. Test the completed functions, mainly to make page transitions more logical and add some necessary prompt windows. And completed the development of new features. This part was done by Yuxiang Ge.
- Continue to perfect the credit mechanism by linking the historical data obtained with the credit mechanism algorithm, and encapsulate necessary components, APIs into the project files. And cooperated with the backend development teammates to complete the development of new functions. This part was completed by Yangshu Wang.

- Complete the development of the credit mechanism algorithm by looking up relevant literature and referencing similar software. Additionally, cooperate with the front-end development teammates to display the credit score and ensure smooth transmission. And participated in the backend development of the new NFT lending feature. This part was completed by Junyi Tang.

- Completed the backend development tasks for image transmission and storage. And completed the backend development tasks for NFT lending, and completed the development of new information release functions. Additionally, deployed the completed function code to the blockchain and completed deployment testing. This part was completed by Jingzhou Hu.

Changes, i.e. tasks that have changed/not completed and why:

Building on the completion of last week's tasks, we added NFT lending and information release functions to improve the system's usability and robustness. However, in the development of the information release function module, the information could not be transmitted properly and needs further improvement.

Agreed weekly "Equity share", i.e. how this sprint's work was split:

- Equity share: Jingzhou Hu (1), Junyi Tang (1), Yuxiang Ge (1), Yangshu Wang (1)

The overall development tasks are the same as before. Additionally, due to the addition of new features, Jingzhou's backend tasks have increased. The other three teammates will also assist Jingzhou in completing the backend development and actively participate in tasks such as documentation writing.

Sprint 4 Planning (Sprint duration 22nd – 26th April)

- Product Owner: Jingzhou Hu, responsible for defining the product direction, managing the product backlog, and ensuring that the team understands and executes the tasks.

- Scrum Master: Jingzhou Hu, responsible for ensuring the team follows agile and Scrum practices, assisting team members in removing impediments, and facilitating communication and collaboration within the team.

Sprint Vision:

During Sprint 3, we completed the development of the credit mechanism lending function, including the formulation of the credit mechanism algorithm and the addition of frontend components. Next, we need to focus on completing the frontend and backend development of the newly added NFT collateral lending function and the information release function, as well as the deployment of both frontend and backend. This part of the tasks is expected to be completed by May 3rd. We will also adjust the tasks according to the actual progress.

Sprint Backlog:

- Phase 1 (Apr. 29 – May 1): "NFT lending" function development

  - During the team meeting, the design and conceptualization of the new features were completed. Yuxiang is responsible for adding the "Borrow NFT" page, where users can enter details about the NFT and the loan amount, post the loan, and pass the information to the NFT module under the "Lending" page, displaying the details and adding an entry point for chat functionality. Junyi and Jingzhou will complete the corresponding backend module development. Yangshu is in charge of the preparations before deployment.

- Phase 2 (May 1 – May 2): "Chat" function development

- Yuxiang is primarily responsible for developing the chat page and the real-time display of chat messages. Jingzhou and Junyi will complete the corresponding backend development. Yangshu will handle the integration of the frontend and backend, among other tasks.

- Phase 3 (May 3): Functionality Testing and Frontend/Backend Deployment
  - Frontend Deployment
  - Backend Deployment
  - Test the specific functionality of the final project.

7.5 Code

Solidity code

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.4;
// Import related libraries
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/Pausable.sol";
import "hardhat/console.sol";
contract LoanLending is Pausable, ReentrancyGuard, AccessControl, Ownable {
uint256 id = 0;
enum LoanState { CREATED, FUNDED, TAKEN, REPAID, FAILED }
enum CollateralState { NOT_PAID, PAID, WALLET_LOCKED, GUARANTOR, WAITING }
enum TypeOfSecurity { COLLATERAL, WALLETLOCK, GUARANTOR }
LoanState loanStateChoice;
CollateralState collateralStateChoice;
TypeOfSecurity typeofSecurityChoice;
bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
bytes32 public constant LENDER_ROLE = keccak256("LENDER_ROLE");
struct Borrower {
address borrower;
address lender;
uint256 amtNeededETH;
uint256 amtRemainingETH;
uint256 amtRaised;
uint256 collateralDeposits;
uint256 loanDuration; // Assuming this is in days for clarity
uint256 interestPercentage;
uint256 returnAmount;
uint256 id;
string itemCategory;
string itemName;
string location;
string description;
string imgURI;
LoanState loanState;
CollateralState collateralState;
}
```

```
    mapping(address => Borrower) public borrowers;
    address[] public borrowerAddresses;
    event CollateralPaid(address indexed sender, uint256 collateralAmount, uint256 timestamp);
    event LoanCreated(address indexed borrower, uint256 amount, uint256 duration, uint256
timestamp);
    event Lend(address indexed borrower, uint256 amount);
    event Repay(address indexed borrower, uint256 amount);
    event LoanRepaid(address indexed borrower, uint256 amount, uint256 timestamp);
    event LoanDefaulted(address indexed borrower, uint256 timestamp);
    event LoanExtended(address indexed borrower, uint256 newDuration);
    constructor() {
    _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _setupRole(ADMIN_ROLE, msg.sender);
    _setupRole(LENDER_ROLE, msg.sender);
    }
    modifier onlyAdmin() {
    require(hasRole(ADMIN_ROLE, msg.sender), "Caller is not an admin");
    _;
    }
    modifier onlyLender() {
    require(hasRole(LENDER_ROLE, msg.sender), "Caller is not a lender");
    _;
    }
    function createLoan(
    address _address,
    uint256 _amtNeededInETH,
    uint256 _loanDuration,
    string memory _collateralType,
    uint256 _interestPercentage,
    string memory _category,
    string memory _item,
    string memory _location,
    string memory _description,
    string memory _imageURI,
    uint256 _collateral
    ) public payable {
    require(checkIfBorrowedBefore(_address), 'You have an outstanding loan');
    require(msg.value >= _collateral, 'The amount of collateral is not enough');
    require(keccak256(abi.encodePacked(_category)) == keccak256(abi.encodePacked('Mortgage')),
"Mortgage category is allowed");
    TypeOfSecurity securityType;
    uint256 interest = (_interestPercentage / 100) * _amtNeededInETH * (_loanDuration / 30);
// Assuming _loanDuration is in days
    uint256 returnAmount = interest + _amtNeededInETH;
    if (keccak256(abi.encodePacked(_collateralType)) == keccak256(abi.encodePacked('LOCK
WALLET'))) {
    securityType = TypeOfSecurity.WALLETLOCK;
    } else if (keccak256(abi.encodePacked(_collateralType)) == keccak256(abi.encodePacked('ETH')))
{
    securityType = TypeOfSecurity.COLLATERAL;
    require(msg.value >= _amtNeededInETH / 2, 'The amount of collateral is not enough');
```

```
    emit CollateralPaid(msg.sender, msg.value, block.timestamp);
    collateralStateChoice = CollateralState.PAID;
    } else {
    securityType = TypeOfSecurity.GUARANTOR;
    }
    handleLoanCreation(
    _address,
    _amtNeededInETH,
    _loanDuration,
    _interestPercentage,
    returnAmount,
    _collateral,
    _category,
    _item,
    _location,
    _description,
    _imageURI
    );
    }
    function checkIfBorrowedBefore(address _address) public view returns (bool) {
    for (uint256 currentAddress = 0; currentAddress < borrowerAddresses.length; currentAd-
dress++) {
    if (borrowerAddresses[currentAddress] == _address) {
    return false;
    }
    }
    return true;
    }
    function cryptoBorrowers(address _address) public view returns (uint256, uint256, address)
{
    _address = msg.sender;
    require(!checkIfBorrowedBefore(msg.sender), 'You do not have an outstanding loan');
    Borrower storage borrower = borrowers[_address];
    return (borrower.loanDuration, borrower.returnAmount, borrower.lender);
    }
    function getId() public returns (uint) {
    id++;
    return id;
    }
    function getBorrowers() view public returns (address[] memory) {
    return borrowerAddresses;
    }
    function getBorrower(address _address) public view returns (uint256, uint256, uint256, uint256,
uint256, uint256, uint256) {
    Borrower storage borrower = borrowers[_address];
    return (borrower.amtNeededETH, borrower.loanDuration, borrower.interestPercentage, bor-
rower.returnAmount, borrower.amtRaised, borrower.amtRemainingETH, borrower.id);
    }
    function withdrawFunds(address payable _borrower) public payable nonReentrant {
    Borrower storage borrower = borrowers[_borrower];
    require(msg.sender == borrower.borrower, 'You have no rights to withdraw');
```

```
    require(borrower.collateralDeposits > 0, 'You have to pay collateral');
    _borrower.transfer(borrower.amtRaised);
    borrower.amtRaised = 0;
    }
    function cryptoLend(address payable _borrower, address _lender) external payable onlyLender {
    Borrower storage borrower = borrowers[_borrower];
    require(_lender != borrower.borrower, 'You cannot lend to yourself');
    if (msg.value > borrower.amtRemainingETH) {
    revert('Exceed the loan limit');
    }
    borrower.amtRaised += msg.value;
    emit Lend(_borrower, msg.value);
    if (borrower.amtRaised == borrower.amtNeededETH) {
    borrower.loanState = LoanState.FUNDED;
    }
    borrower.lender = msg.sender;
    borrower.amtRemainingETH = borrower.amtNeededETH - borrower.amtRaised;
    }
    function fetchMortgageBorrowers(address _address) public view returns (address, string memory, string memory, string memory, string memory, uint256, uint256) {
    Borrower storage borrower = borrowers[_address];
    return (borrower.borrower, borrower.itemName, borrower.location, borrower.description, borrower.imgURI, borrower.collateralDeposits, borrower.loanDuration);
    }
    function repayLoan(address payable _borrower) public payable nonReentrant {
    Borrower storage borrower = borrowers[_borrower];
    require(msg.value >= borrower.returnAmount, 'Insufficient repayment amount');
    (bool success, ) = borrower.lender.call{value: msg.value}("");
    require(success, "Failed to send Ether");
    borrower.loanState = LoanState.REPAID;
    emit Repay(_borrower, msg.value);
    _borrower.transfer(borrower.collateralDeposits);
    for (uint256 i = 0; i < borrowerAddresses.length; i++) {
    if (borrowerAddresses[i] == _borrower) {
    borrowerAddresses[i] = borrowerAddresses[borrowerAddresses.length - 1];
    borrowerAddresses.pop();
    break;
    }
    }
    emit LoanRepaid(_borrower, msg.value, block.timestamp);
    }
    function checkOverdue(address _borrower) internal view returns (bool) {
    Borrower storage borrower = borrowers[_borrower];
    if (borrower.loanDuration == 0) {
    return false;
    }
    uint256 dueDate = borrower.loanDuration * 1 days;
    return block.timestamp > dueDate;
    }
    function confiscateCollateral(address _borrower) public onlyAdmin {
```

```
    require(checkOverdue(_borrower), ”Loan is not overdue”);
    Borrower storage borrower = borrowers[_borrower];
    address lender = borrower.lender;
    payable(lender).transfer(borrower.collateralDeposits);
    borrower.collateralDeposits = 0;
    borrower.loanState = LoanState.FAILED;
    for (uint256 index = 0; index < borrowerAddresses.length; index++) {
    if (borrowerAddresses[index] == _borrower) {
    borrowerAddresses[index] = borrowerAddresses[borrowerAddresses.length - 1];
    borrowerAddresses.pop();
    break;
    }
    }
    emit LoanDefaulted(_borrower, block.timestamp);
    }
    function handleDefault(address _borrower) public onlyAdmin {
    confiscateCollateral(_borrower);
    }
    function requestExtension(address _borrower) public onlyAdmin {
    Borrower storage borrower = borrowers[_borrower];
    require(borrower.loanState == LoanState.FUNDED, ”Loan must be funded to request ex-
tension”);
    borrower.loanDuration += 30;
    borrower.returnAmount += 0.2 ether;
    payable(borrower.lender).transfer(0.2 ether);
    emit LoanExtended(_borrower, borrower.loanDuration);
    }
    function handleLoanCreation(
    address _borrower,
    uint256 _amtNeededInETH,
    uint256 _loanDuration,
    uint256 _interestPercentage,
    uint256 _returnAmount,
    uint256 _collateral,
    string memory _category,
    string memory _item,
    string memory _location,
    string memory _description,
    string memory _imageURI
    ) internal {
    Borrower storage borrower = borrowers[_borrower];
    borrower.borrower = _borrower;
    borrower.amtNeededETH = _amtNeededInETH;
    borrower.loanDuration = _loanDuration;
    borrower.interestPercentage = _interestPercentage;
    borrower.returnAmount = _returnAmount;
    borrower.amtRaised = 0;
    borrower.collateralDeposits = _collateral;
    borrower.amtRemainingETH = _amtNeededInETH;
    borrower.id = getId();
    borrower.itemCategory = _category;
```

```
borrower.itemName = _item;
borrower.location = _location;
borrower.description = _description;
borrower.imgURI = _imageURI;
borrower.loanState = LoanState.CREATED;
borrowerAddresses.push(_borrower);
emit LoanCreated(_borrower, _amtNeededInETH, _loanDuration, block.timestamp);
}
} Front Code
borrowpage
import React, { useState } from 'react';
import Navbar from '../components/Navbar';
import BorrowForm from '../components/BorrowForm';
import BorrowItem from '../components/BorrowItem';
import { Box, Paper, Tab, Tabs } from '@mui/material';
import { TabContext, TabPanel } from '@mui/lab';
const BorrowPage = () => {
const [tabIndex, setTabIndex] = useState('1');
const handleTabChange = (event, newValue) => {
setTabIndex(newValue);
};
const tabPanelStyles = {
backgroundColor: '#F5F5F5',
padding: '16px'
};
return (
<>
<Navbar />
<TabContext value={tabIndex}>
<Paper elevation={3} sx={ width: '100%', mt: 2 }>
<Tabs
value={tabIndex}
onChange={handleTabChange}
indicatorColor="primary"
textColor="primary"
variant="fullWidth"
>
<Tab label="Borrow Crypto" value="1" />
<Tab label="Borrow Item" value="2" />
</Tabs>
</Paper>
<TabPanel value="1" sx={tabPanelStyles}>
<BorrowForm />
</TabPanel>
<TabPanel value="2" sx={tabPanelStyles}>
<BorrowItem />
</TabPanel>
</TabContext>
</>
);
};
```

```
export default BorrowPage;
lendingpage
import { Box, Button, Hidden, List, ListItem, ListItemText } from '@mui/material';
import React, { useEffect } from 'react';
import LendSections from '../components/LendSections';
import Navbar from '../components/Navbar';
import PaidIcon from '@mui/icons-material/Paid';
import HomeIcon from '@mui/icons-material/Home';
// import CarRentalIcon from '@mui/icons-material/CarRental';
// import CableIcon from '@mui/icons-material/Cable';
// import YardIcon from '@mui/icons-material/Yard';
// import BedroomParentIcon from '@mui/icons-material/BedroomParent';
// import CreditCardIcon from '@mui/icons-material/CreditCard';
import { Outlet } from 'react-router-dom';
import { useNavigate } from 'react-router-dom';
import KeyboardArrowDownIcon from '@mui/icons-material/KeyboardArrowDown';
import { Link } from 'react-router-dom';
const Lendingpage = () => {
const [category, setCategory] = React.useState('Crypto');
const [isActive, setIsActive] = React.useState(false);
const onClick = () => {
setIsActive(!isActive);
};
const navigate = useNavigate();
useEffect(() => {
let isCancelled = false; // Changed the initial value to false
if (!isCancelled) {
navigate('/lend/crypto');
}
return () => {
isCancelled = true; // Set to true on cleanup
};
}, [navigate]);
return (
<div style={ height: '100%' }>
<Navbar />
<Hidden lgUp>
<Button onClick={onClick} sx={ border: '1px solid black', margin: 2 }>
{category}
<KeyboardArrowDownIcon />
</Button>
<List sx={ display: `$isActive ? 'inline' : 'none'}` }>
<ListItem button color="#dddfd4" component={Link} to="/lend/crypto">
<ListItemText
onClick={event => {
setCategory(event.target.innerText);
setIsActive(false);
}
>
Crypto
</ListItemText>
```

```
</ListItem>
<ListItem button color=”#dddfd4” component={Link} to=”/lend/mortgage”>
<ListItemText
onClick={event => {
setCategory(event.target.innerText);
setIsActive(false);
}
>
Mortgage
</ListItemText>
</ListItem>
</List>
</Hidden>
<div style={ display: ’flex’ }>
<Hidden lgDown>
<section
style={
display: ’flex’,
flexDirection: ’column’,
alignItems: ’center’,
justifyContent: ’center’,
backgroundColor: ’#173e43’,
height: ’100vh’
}
>
<Box sx={ justifyContent: ’space-between’, height: ’100%’, marginTop: ’auto’ }>
<LendSections title=”crypto” selected Icon={<PaidIcon />} />
<LendSections title=”mortgage” Icon={<HomeIcon />} />
</Box>
</section>
</Hidden>
<Outlet style={ margin: 0 } />
</div>
</div>
);
};
export default Lendingpage;
loginpage:
import React from ’react’;
import { Button, Container } from ’@mui/material’;
import { useDispatch } from ’react-redux’;
import { getAddress } from ’../features/ConnectWalletSlice’;
const LoginPage = () => {
const dispatch = useDispatch();
const handleWalletConnect = async () => {
if (typeof window.ethereum !== ’undefined’) {
dispatch(getAddress());
} else {
alert(’Install MetaMask, please’);
}
};
```

```
return (
<Container
maxWidth="sm"
style={
display: 'flex',
flexDirection: 'column',
alignItems: 'center',
justifyContent: 'center',
height: '100vh',
backgroundColor: '#dddfd4'
}
>
<Button variant="contained" color="primary" onClick={handleWalletConnect}></Button>
</Container>
);
};
export default LoginPage;
payoffpage
import React, { useEffect, useState } from 'react';
import { useSelector } from 'react-redux';
import { ethers } from 'ethers';
import { Box, Typography, Container } from '@mui/material';
import Navbar from '../components/Navbar';
import Lender from '../components/Lender';
import { LOANLENDING_CONTRACT_ADDRESS, abi } from '../constants';
const PayOffPage = () => {
// Get user address from Redux state
const userAddress = useSelector(state => state.connectWallet.address);
// Initialize Ethereum provider and contract
const provider = new ethers.providers.Web3Provider(window.ethereum);
const contract = new ethers.Contract(LOANLENDING_CONTRACT_ADDRESS, abi, provider.getSigner(us
// Define state variables
const [loanData, setLoanData] = useState(null);
const [errorMsg, setErrorMsg] = useState('');
// Fetch loan information from the contract
useEffect(() => {
const fetchLoanData = async () => {
try {
const borrowerData = await contract.cryptoBorrowers(userAddress);
const formattedData = {
duration: parseFloat(borrowerData[0]) / 10,
amount: Number(ethers.utils.formatEther(borrowerData[1])).toFixed(4),
lender: borrowerData[2]
};
setLoanData(formattedData);
} catch (error) {
setErrorMsg(error.message);
console.error('Error fetching borrower data:', error);
}
};
fetchLoanData();
```

```
    }, [userAddress, contract]);
    return (
    <div>
    <Navbar />
    <Container sx={ display: 'flex', flexDirection: 'column', alignItems: 'center', mt: 4 }>
    {loanData ? (
    <Lender loanAmount={loanData.amount} lender={loanData.lender} loanDuration={loanData.duration}
/>
    ) : (
    <Box sx={ display: 'flex', alignItems: 'center', justifyContent: 'center', height: '100vh' }>
    <Box sx={ p: 3, bgcolor: 'black', borderRadius: 2 }>
    <Typography variant="h2" color="white" gutterBottom>
    Error!!
    </Typography>
    <Typography variant="h6" color="gray">
    {errorMsg}
    </Typography>
    </Box>
    </Box>
    )}
    </Container>
    </div>
    );
    };
    export default PayOffPage;
    borrowform
    import React, { useState } from 'react';
    import { useSelector } from 'react-redux';
    import { ethers } from 'ethers';
    import { Container, Box, Typography, TextField, Button, Slider, FormControl, RadioGroup,
FormControlLabel, Radio, InputAdornment, CircularProgress } from '@mui/material';
    import { LOANLENDING_CONTRACT_ADDRESS, abi } from '../constants';
    import { maxValue } from '../features/MaxBorrow';
    const BorrowForm = () => {
    // State variables
    const [loanAmount, setLoanAmount] = useState('');
    const [loanDuration, setLoanDuration] = useState(0);
    const [interestRate, setInterestRate] = useState(0);
    const [collateralAmount, setCollateralAmount] = useState('');
    const [currency, setCurrency] = useState('ETH');
    const [loading, setLoading] = useState(false);
    const [errorMessage, setErrorMessage] = useState('');
    // Get the user's address from Redux state
    const userAddress = useSelector(state => state.connectWallet.address);
    // Ethereum provider and contract instance
    const provider = new ethers.providers.Web3Provider(window.ethereum);
    const contract = new ethers.Contract(LOANLENDING_CONTRACT_ADDRESS, abi, provider.getSigner(us
    // Handle loan creation
    const handleCreateLoan = async (event) => {
    event.preventDefault();
    setLoading(true);
```

```
try {
const parsedLoanAmount = ethers.utils.parseUnits(loanAmount.toString(), 'ether');
const transaction = await contract.createCryptoLoan(
userAddress,
parsedLoanAmount,
loanDuration * 10,
currency,
interestRate * 1000,
{
from: userAddress,
value: ethers.utils.parseEther(collateralAmount)
}
);
await transaction.wait();
resetForm();
} catch (error) {
setErrorMessage(error.message);
console.error('Error creating loan:', error);
}
setLoading(false);
};
// Reset form fields
const resetForm = () => {
setLoanAmount('');
setLoanDuration(0);
setInterestRate(0);
setCollateralAmount('');
setCurrency('ETH');
};
return (
<Container maxWidth="md" sx={{ mt: 5 }}>
<Typography variant="h3" align="center" gutterBottom>
Request Crypto Loan on Ethereum
</Typography>
{errorMessage & (
<Typography variant="h6" color="error" align="center" gutterBottom>
{errorMessage}
</Typography>
)}
<form onSubmit={handleCreateLoan}>
<Box mb={3}>
<TextField
fullWidth
required
type="number"
label="Amount to Borrow (USD)"
value={loanAmount}
onChange={(e) => setLoanAmount(e.target.value)}
InputProps={
endAdornment: <InputAdornment position="end">USD</InputAdornment>,
inputProps: { min: 0, max: maxValue, step: '0.01' }
```

```
}
/>
</Box>
<Box mb={3}>
<Typography variant=”h6”>Loan Duration (Months)</Typography>
<Slider
value={loanDuration}
onChange={(e, newValue) => setLoanDuration(newValue)}
step={1}
marks
min={0}
max={12}
valueLabelDisplay=”auto”
/>
</Box>
<Box mb={3}>
<Typography variant=”h6”>Interest Rate (%)</Typography>
<Slider
value={interestRate}
onChange={(e, newValue) => setInterestRate(newValue)}
step={0.1}
marks
min={0}
max={50}
valueLabelDisplay=”auto”
/>
</Box>
<Box mb={3}>
{currency === ’ETH’ & (
<TextField
fullWidth
required
type=”number”
label=”ETH Collateral”
value={collateralAmount}
onChange={(e) => setCollateralAmount(e.target.value)}
InputProps={
endAdornment: <InputAdornment position=”end”>ETH</InputAdornment>,
inputProps: { min: 0.0, max: maxValue, step: ’0.01’ }
}
/>
)}
</Box>
<Box mb={3} display=”flex” justifyContent=”center”>
<FormControl component=”fieldset”>
<RadioGroup
row
value={currency}
onChange={(e) => setCurrency(e.target.value)}
>
<FormControlLabel value=”ETH” control={<Radio />} label=”ETH” />
```

```
<FormControlLabel value=”Lock Wallet” control={<Radio />} label=”Lock Wallet” />
</RadioGroup>
</FormControl>
</Box>
<Box display=”flex” justifyContent=”center” mb={3}>
<Button variant=”contained” color=”primary” type=”submit” disabled={loading}>
{loading ? <CircularProgress size={24} /> : ’Submit’}
</Button>
</Box>
</form>
{loading & (
<Box display=”flex” justifyContent=”center” mt={3}>
<CircularProgress />
</Box>
)}
</Container>
);
};
export default BorrowForm;
lender
import { Box, Button, Paper, Typography } from ’@mui/material’;
import React from ’react’;
import { useSelector } from ’react-redux’;
import { ethers } from ’ethers’;
import { LOANLENDING_CONTRACT_ADDRESS, abi } from ’../constants’;
const Lender = ({ loanDuration, loanAmount, lender }) => {
const address = useSelector(state => state.connectWallet.address);
const provider = new ethers.providers.Web3Provider(window.ethereum);
const contract = new ethers.Contract(LOANLENDING_CONTRACT_ADDRESS, abi, provider.getSigner(ad
const pay = async () => {
try {
const tx = await contract.cryptoRepay(address, {
value: ethers.utils.parseUnits(loanAmount.toString(), ’ether’)
});
tx.wait();
} catch (err) {
console.log(err);
}
};
return (
<Paper
sx={
margin: ’10px’,
marginLeft: ’auto’,
marginRight: ’auto’,
//backgroundColor: ’green’,
//width: ’100%’,
display: ’flex’,
alignItems: ’center’,
justifyContent: ’center’
}
```

```
>
<Box
mr={2}
p={2}
sx={
display: 'flex',
flexDirection: 'column',
alignItem: 'center',
//width: '100%',
//backgroundColor: 'blue',
justifyContent: 'space-between'
}
>
<Typography gutterBottom>{lender}</Typography>
<Typography color="primary" gutterBottom>
{loanAmount}ETH
</Typography>
<Typography color="primary" gutterBottom>
{loanDuration * 30}days
</Typography>
</Box>
<Button variant="contained" sx={ marginRight: '15px' } onClick={pay}>
Pay
</Button>
</Paper>
);
};
export default Lender;
lendertable
import React, { useState, useEffect } from 'react';
import { useSelector } from 'react-redux';
import { ethers } from 'ethers';
import { Container, Paper, Table, TableBody, TableCell, TableContainer, TableHead, TableRow,
Box, TextField, Button, Typography, InputAdornment } from '@mui/material';
import CloseIcon from '@mui/icons-material/Close';
import { LOANLENDING_CONTRACT_ADDRESS, abi } from '../constants';
import { getETHPrice } from '../utils/getEthPrice';
import makeStyles from '@mui/styles/makeStyles';
const useStyles = makeStyles(theme => ({
modal: {
width: '90vw',
[theme.breakpoints.down('xs')]: {
width: '100vw'
},
[theme.breakpoints.up('md')]: {
width: '72vw'
},
display: 'flex',
justifyContent: 'center',
alignItems: 'center',
height: '50vh',
```

```
zIndex: 1000,
boxShadow: '10px 2px 20px 1px'
},
tableContainer: {
maxWidth: '800px',
[theme.breakpoints.down('sm')]: {
maxWidth: '500px'
}
}
}));
const LendingTable = () => {
const classes = useStyles();
const [borrowers, setBorrowers] = useState([]);
const [showModal, setShowModal] = useState(false);
const [lendAmount, setLendAmount] = useState('');
const [selectedBorrower, setSelectedBorrower] = useState('');
const [withdrawEnabled, setWithdrawEnabled] = useState(false);
const address = useSelector(state => state.connectWallet.address);
const provider = new ethers.providers.Web3Provider(window.ethereum);
const contract = new ethers.Contract(LOANLENDING_CONTRACT_ADDRESS, abi, provider.getSigner(ac
const fetchBorrowers = async () => {
try {
const borrowerAddresses = await contract.getBorrowers();
const ethPrice = await getETHPrice();
const borrowerData = await Promise.all(borrowerAddresses.map(async addr => {
const data = await contract.getBorrower(addr);
return {
id: parseInt(data[6]),
address: addr,
amountNeeded: ethers.utils.formatEther(data[0]),
loanDuration: parseFloat(data[1]) / 10,
interestPercentage: parseFloat(data[2]) / 1000,
returnAmount: Number(ethers.utils.formatEther(data[3])).toFixed(4),
amountRaised: ethers.utils.formatEther(data[4]),
amountRemaining: ethers.utils.formatEther(data[5]),
amountNeededUSD: (ethers.utils.formatEther(data[0]) * ethPrice).toFixed(2),
amountRaisedUSD: (ethers.utils.formatEther(data[4]) * ethPrice).toFixed(2),
amountRemainingUSD: (ethers.utils.formatEther(data[5]) * ethPrice).toFixed(2),
returnAmountUSD: (ethers.utils.formatEther(data[3]) * ethPrice).toFixed(2)
};
}));
setBorrowers(borrowerData);
} catch (error) {
console.error('Error fetching borrowers:', error);
}
};
useEffect(() => {
fetchBorrowers();
}, []);
const handleLend = async e => {
e.preventDefault();
```

```
try {
await contract.cryptoLend(selectedBorrower, address, {
from: address,
value: ethers.utils.parseUnits(lendAmount.toString(), 'ether')
});
setShowModal(false);
fetchBorrowers();
} catch (error) {
console.error('Error lending ETH:', error);
}
};
const handleWithdraw = async () => {
try {
await contract.withdrawFunds(address);
setWithdrawEnabled(true);
} catch (error) {
console.error('Error withdrawing funds:', error);
}
};
return (
<Container maxWidth="lg" className={classes.tableContainer} sx={ mt: 4, p: 2 }>
<TableContainer component={Paper}>
<Table>
<TableHead sx={ backgroundColor: '#173e43' }>
<TableRow>
{['Borrower Address', 'Amount Needed', 'Loan Duration', 'Interest Percentage', 'Return Amount',
'Amount Raised', 'Amount Remaining', 'Actions'].map((header, idx) => (
<TableCell key={idx} sx={ color: 'white' }>{header}</TableCell>
))}
</TableRow>
</TableHead>
<TableBody>
{borrowers.length > 0 ? borrowers.map(borrower => (
<TableRow key={borrower.id}>
<TableCell>{borrower.address.slice(0, 4)}...{borrower.address.slice(-4)}</TableCell>
<TableCell>{borrower.amountNeeded} ETH<br /><Typography variant="body2" color="red">$borrower.am
<TableCell>{borrower.loanDuration} days</TableCell>
<TableCell>{borrower.interestPercentage}
<TableCell>{borrower.returnAmount} ETH<br /><Typography variant="body2" color="red">$borrower.return
<TableCell>{borrower.amountRaised} ETH<br /><Typography variant="body2" color="red">$borrower.amo
<TableCell>{borrower.amountRemaining} ETH<br /><Typography variant="body2" color="red">$borrower.a
<TableCell>
{borrower.amountRemaining > 0 ? (
<Button variant="contained" onClick={() => { setShowModal(true); setSelectedBorrower(borrower.address)
}>Lend</Button>
) : (
<Button variant="contained" disabled>Lend</Button>
)}
{borrower.amountRaised >= borrower.amountNeeded & (
<Button variant="contained" onClick={handleWithdraw}>Withdraw</Button>
)}
```

```
</TableCell>
</TableRow>
)) : (
<TableRow>
<TableCell colSpan={8} align="center">No borrowers yet</TableCell>
</TableRow>
)}
</TableBody>
</Table>
</TableContainer>
{showModal & (
<Box className={classes.modal}>
<Paper>
<Box p={5}>
<CloseIcon sx={ float: 'right', cursor: 'pointer' } onClick={() => setShowModal(false)} />
<form onSubmit={handleLend}>
<TextField
fullWidth
required
label="Borrower's Address"
value={selectedBorrower}
InputProps={ readOnly: true }
sx={ mb: 3 }
/>
<TextField
fullWidth
required
type="number"
label="Amount to Lend (ETH)"
value={lendAmount}
onChange={(e) => setLendAmount(e.target.value)}
InputProps={
endAdornment: <InputAdornment position="end">ETH</InputAdornment>,
inputProps: { min: 0.0, step: '0.01' }
}
sx={ mb: 3 }
/>
<Button variant="contained" type="submit">Send</Button>
</form>
</Box>
</Paper>
</Box>
)}
</Container>
);
};
export default LendingTable;
```

Function image□

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
def calculateCreditScore(totalTransactions, averageAmount):
score = 300
score += np.minimum(np.log10(1 + totalTransactions) * 100, 300)
score += np.minimum(np.log10(1 + averageAmount) * 1000, 250)
return score
transaction_threshold = 10**3 - 1
amount_threshold = 10**(0.25) - 1
totalTransactions = np.linspace(0, 1.3 * transaction_threshold, 100)
averageAmount = np.linspace(0, 1.3 * amount_threshold, 100)
X, Y = np.meshgrid(totalTransactions, averageAmount)
Z = calculateCreditScore(X, Y)
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap='gray')
ax.set_xlabel('Total Transactions')
ax.set_ylabel('Average Amount')
ax.set_zlabel('Credit Score')
plt.title('Credit Score')
colorbar = plt.colorbar(surf, shrink=0.5, aspect=10)
ax.view_init(elev=20, azim=-60)
plt.show()
creditscore
const express = require('express');
const app = express();
const axios = require('axios');
const fs = require('fs');
app.use(express.json());
const ETHERSCAN_API_KEY = 'JIZZMGHYYD46MICJIFYVZNXRA5UKBMHDP5';
const ETHERSCAN_API_URL = 'https://api-sepolia.etherscan.io/api';
function convertWeiToEther(weiValue) {
return parseFloat(weiValue) / 1e18;
}
function calculateCreditScore(totalTransactions, averageAmount) {
const score = 300
+ Math.min(Math.log10(1 + totalTransactions) * 100, 300)
+ Math.min(Math.log10(1 + averageAmount) * 1000, 250);
return score;
}
function calculateMaxLoanAmount(score) {
if (score < 580) {
return 0.5;
} else if (score < 670) {
return 0.5 + (score - 580) * (1.0 / 90);
} else if (score < 740) {
return 1.5 + (score - 670) * (1.0 / 70);
} else if (score < 800) {
return 2.5 + (score - 740) * (1.0 / 60);
} else if (score <= 850) {
return 3.5 + (score - 800) * (1.5 / 50);
}
return 5.0;
```

```
}
async function getTransactionHistory(address) {
try {
const params = new URLSearchParams({
module: 'account',
action: 'txlist',
address: address,
startblock: 0,
endblock: 99999999,
sort: 'asc',
apikey: ETHERSCAN_API_KEY
});
const response = await axios.get(`$ETHERSCAN_API_URL}?$params.toString()}`);
if (!Array.isArray(response.data.result)) {
throw new Error('The response from the Etherscan API is not an array');
}
const transactions = response.data.result.map(tx => ({
valueInEther: convertWeiToEther(tx.value)
}));
const totalTransactions = transactions.length;
const averageTransactionValue = transactions.reduce((acc, tx) => acc + tx.valueInEther, 0)
/ totalTransactions;
const creditScore = calculateCreditScore(totalTransactions, averageTransactionValue);
const maxLoanAmount = calculateMaxLoanAmount(creditScore);
const resultData = {
totalTransactions: totalTransactions,
averageAmount: averageTransactionValue,
credit: creditScore,
maxLoanAmount: maxLoanAmount
};
fs.writeFileSync('transaction_summary.json', JSON.stringify(resultData, null, 2), 'utf-8');
console.log('Transaction summary with credit score and max loan amount has been written
to transaction_summary.json');
return resultData;
} catch (error) {
console.error('Error:', error.message);
return null;
}
}
app.post('/get-transaction-history', async (req, res) => {
const address = req.body.address;
try {
const summary = await getTransactionHistory(address);
res.json(summary);
} catch (error) {
res.status(500).send(error.message);
}
});
const PORT = 3000;
app.listen(PORT, () => {
console.log(`Server running on http://localhost:$PORT}`);});
```