# Decentralized Application for Depositing and Lending: Project Report

## 1. Introduction

**Purpose of the Project**

The goal of this project is to create a decentralized application (DApp) that allows users to perform secure transactions such as depositing, lending, and repayment of funds on the blockchain. This application enhances user transparency and security while leveraging blockchain technology to provide decentralized financial services.

**Importance of Blockchain Integration**

Blockchain offers distinct advantages in terms of security, transparency, and immutability. By using smart contracts, users can engage in peer-to-peer transactions without the need for intermediaries, ensuring trustless and secure interactions. Blockchain's decentralized nature allows us to create an efficient, secure, and automated system for financial operations.

**Advantages of Decentralized Lending over Traditional Lending**

Decentralized lending offers several key advantages over traditional lending systems. Firstly, it eliminates the need for intermediaries such as banks, resulting in lower transaction fees and faster processing times. Additionally, decentralized lending platforms are globally accessible, providing financial services to individuals in regions with limited banking infrastructure. The transparency of blockchain technology also ensures that all transactions are recorded on an immutable ledger, reducing the risk of fraud. Furthermore, decentralized platforms offer greater liquidity and flexibility, allowing users to leverage a wide range of crypto assets as collateral. Finally, decentralized lending platforms prioritize privacy, enabling users to borrow without undergoing stringent identity verification or credit checks.

## 2. Technology Stack

**Blockchain Platform**

The project is built on the Ethereum blockchain, a robust platform known for its ability to execute decentralized applications (DApps). Ethereum’s smart contract capabilities are ideal for building the deposit and lending functionality, offering transparency and security in financial transactions. Ethereum’s decentralized nature ensures that all transactions are recorded immutably, providing trustless operations without a central authority. While the project is built on Ethereum, it does not involve real cryptocurrency transactions. Instead, all data is created within the Solidity code, and the associated transaction fees are processed using SepoliaETH on the Sepolia test network, ensuring that no real-world cryptocurrency is exchanged during the development and testing phases.

**Development Tools**

For the development of this project, I used the following tools:

Frontend: HTML, JavaScript, and Bootstrap for designing user interfaces and handling user inputs.

Backend: Flask (Python) to handle communication between the frontend and the Ethereum blockchain, facilitating transactions.

Smart Contracts: Solidity was used to write smart contracts that govern the deposit, lending, and repayment functionalities. Solidity’s popularity and integration with Ethereum make it the ideal choice for this project.

Web3.js: To interact with the Ethereum blockchain from the frontend, Web3.js was utilized, facilitating communication between the browser and the blockchain.

Database: MariaDB was used to store user data, including a user table that records usernames and account creation timestamps. No sensitive information such as user passwords is stored within the database, ensuring a focus on minimal data storage for this project.

## 3. System Design and Architecture

The system is designed to provide decentralized financial services, specifically focusing on deposit, lending, and repayment functionalities. It leverages blockchain technology to create a trustless, transparent, and secure environment for users.

**Deposit Functionality:** The deposit function allows users to store a specified amount of digital assets into the system. The transaction is securely recorded on the Ethereum blockchain, ensuring that every deposit is verifiable and immutable. The deposited funds are tracked in the smart contract, and users can view their balance at any time. This feature ensures that users can safely hold their assets in a decentralized manner without relying on a central authority.

**Lending Functionality:** The lending feature allows users to borrow assets from the system’s available funds. Instead of using collateral, the system checks the total amount of assets available in the shared lending pool. Users are permitted to request a loan amount, and the smart contract ensures that the requested amount does not exceed the available assets in the pool. Once approved, the requested amount is transferred to the user, and the loan details are securely recorded on the blockchain for full transparency. This approach prevents users from borrowing more than what is available, thus maintaining the stability of the system’s overall funds.

**Repayment Functionality:** The repayment functionality, also payback function in code allows users to repay their loans, after which they regain full access to their deposit. The system calculates the total loan amount due, without applying interest, and updates the user’s balance accordingly. Both deposits and borrowings are managed through a shared lending pool, meaning neither lenders nor borrowers are aware of the specific allocation of their funds. The repayment process is automated via the smart contract, ensuring that the pool’s balance is updated accurately, while maintaining anonymity in how funds are distributed among users.

**User Data Management:** Although the system does not handle sensitive data like passwords, basic user data such as usernames and account creation timestamps are stored in the MariaDB database. This information is used to maintain a minimal level of user tracking for administrative purposes without compromising user privacy.

**Blockchain Integration:** All transactions—whether deposits, loans, or repayments—are processed on the Ethereum blockchain, providing transparency and immutability. While real cryptocurrency transactions are not performed, the system simulates these actions using test assets like SepoliaETH, ensuring that the system operates without involving real monetary risk during development and testing.

## 4. Implementation

**Front-End Implementation:**

The front-end of the decentralized application (DApp) is built using HTML, JavaScript, and Bootstrap, providing a clean and intuitive interface. Users interact with the DApp through multiple sections, including deposit and borrowing pages. These pages allow users to enter amounts for deposit or borrowing, and buttons execute the corresponding actions.

* The landing page welcomes users and asks for their name, which is stored and displayed on the dashboard.
* The dashboard shows total balance, available liquidity, and borrowing stats. A pie chart visualizes these stats for better user experience.
* The deposit page allows users to deposit money into the system, check their current balance, and pay back borrowed amounts.
* The borrowing page provides similar functionality but focuses on borrowing operations, allowing users to borrow, check borrowed value, and available borrowing limits.

**Back-End Implementation:**

The back-end is developed using Python's Flask framework, which handles the communication between the front-end and the smart contracts deployed on the Ethereum blockchain. Flask endpoints, such as /deposit, /borrow, and /balance, manage the user's requests and trigger appropriate blockchain interactions.

**Flask handles:**

Front-end rendering through templates, dynamically displaying user-specific information.

Managing user input for deposits, borrow amounts, and repayment actions.

Facilitating communication with the deployed smart contract, passing the user’s input to the contract functions and retrieving relevant information like balances and available liquidity.

**Smart Contract Implementation:**

The smart contract is written in Solidity and deployed on the Ethereum Sepolia test network. It manages the core functionality of the DApp, including deposits, borrowing, and repayments.

* Deposit Functionality: Users can deposit tokens into a common liquidity pool. The contract tracks each user’s deposit and updates their balance accordingly.
* Borrowing Functionality: The contract allows users to borrow from the liquidity pool, but it restricts borrowing to the available liquidity. This prevents users from borrowing more than the contract can provide.
* Repayment Functionality: Users repay the borrowed amount to regain full access to their deposit. The smart contract updates the user’s balance and total liquidity after repayment.
* Balance Tracking: The contract stores user-specific data, ensuring transparency of balances, borrowings, and available liquidity.

The Sepolia test network ensures that no real cryptocurrency is used. Instead, the contract interacts with SepoliaETH, a test currency, for all transactions and fee payments.

**Testing:** Thorough testing was performed to ensure the correctness of the DApp:

* Unit Testing: The smart contract functions were tested individually using Remix IDE. This ensured that the core functionalities, such as deposits, borrowing, and repayments, worked as expected.
* Integration Testing: The Flask back-end and Solidity contracts were tested together to ensure seamless interaction between the front-end inputs and blockchain operations. Transactions were tested to verify that user inputs (e.g., deposit amounts, borrow requests) were correctly processed by the contract and reflected in the front-end.
* User Testing: End-users were involved in testing the DApp. They provided feedback on usability and overall experience. Common user interactions, such as depositing, borrowing, and checking balances, were tested to verify functionality under real-use conditions.

## 5. Challenges and Solutions

**Gas Optimization:**

One significant challenge encountered during the development process was gas optimization for the smart contract transactions. Gas fees can become quite costly, especially when interacting with blockchain networks like Ethereum. To ensure that users experience minimal costs, efforts were made to optimize the smart contracts by reducing function complexity and minimizing storage operations.

However, during the testing phase, an issue was discovered when users attempted to repay more than their borrowed amount. In this case, MetaMask would issue a warning indicating that it could not calculate the required gas, resulting in a significantly higher gas fee being charged. This unexpected behavior was observed when the repayment exceeded the borrowed amount, leading to inefficiencies and excessive costs.

To address this, additional validation was added to the front-end and smart contracts to prevent users from entering repayment amounts that exceed their borrowed value. This ensures that the repayment logic is more streamlined and helps reduce gas consumption by limiting unnecessary calculations and storage changes. Despite these efforts, gas optimization remains a complex and ongoing challenge, especially when dealing with user errors or edge cases in decentralized applications.

**Session Management Issue:**

Due to time constraints during the development phase, session handling was not fully implemented in the front-end code. As a result, after completing actions on the borrow and deposit pages, the system redirects users back to the login page instead of the main dashboard. The lack of session persistence meant that users were unable to seamlessly navigate between pages within the application. This will be addressed in future updates by adding proper session handling to ensure a smoother user experience.

## 6. References

Due to the practical nature of this project, reference materials included industry websites such as Compound Finance and blockchain development tutorials for Solidity and Web3.js.

## 7. Conclusion and Future Work

**Project Summary:**

In conclusion, this decentralized application successfully allows users to deposit, lend, and repay funds on the Ethereum blockchain. The integration of smart contracts ensures that transactions are secure and trustless, while the frontend offers a user-friendly interface for interacting with the blockchain.

**Potential Improvements:**

Despite the project’s successful implementation of core functionalities such as deposit, borrowing, and repayment on the blockchain, several areas can be enhanced to improve both user experience and the overall performance of the decentralized application (DApp).

* Session Management: A key issue identified was the lack of session persistence in the front-end application. Currently, users are redirected to the login page after completing actions on the borrow and deposit pages, instead of seamlessly navigating to the main dashboard. Implementing session storage using browser cookies or local storage could improve the user experience by maintaining login states and allowing smoother transitions between pages. This would ensure that user data, such as account balances and borrowing details, are maintained between interactions.
* Error Handling for Overpayments: As highlighted during testing, overpayment of loans leads to gas fee calculation errors and excessively high fees. A more robust error handling mechanism can be implemented to prevent users from accidentally overpaying their loans. For instance, the front-end could validate input amounts, ensuring that repayment amounts do not exceed the borrowed sum. On the smart contract level, more safeguards can be included to reject such transactions outright, improving system reliability and saving users from unnecessarily high gas fees.
* Interest and Collateral System: While the current system does not include interest rates or collateral-based lending, these features could be valuable additions. Adding a dynamic interest calculation based on market conditions or collateral ratios could make the platform more competitive and offer users a broader range of lending and borrowing options. Additionally, incorporating a collateral system would enable users to secure larger loans, expanding the platform's use cases while ensuring risk mitigation for lenders.

**Future Enhancements:**

Potential enhancements include improving the user interface further to accommodate mobile users, as well as implementing additional security measures to safeguard user funds from any potential vulnerabilities.