Decentralized Book Rental Platform Report

Team Members:

- Aarav Gupta - 230008001
- Devesh Yadav - 230001024
- Harshith Jai Surya Ganji - 230041010
- Md. Asif Hussain - 230041021
- Neelam Sai Sathwik - 230041024
- Sai Prakul Parepalli - 230041031

Course: CS 218 - Programmable and Interoperable Blockchains

Table of Contents

Abstract

The Decentralized Book Rental Platform is a blockchain-based system that enables secure, transparent, and trustless peer-to-peer book rentals. By leveraging Ethereum smart contracts, decentralized storage (IPFS), and a modern web frontend, this platform automates the rental lifecycle-including listing, renting, returning, deposit management, and penalty enforcement-without the need for intermediaries. This report outlines the system's design, implementation, technical challenges, and evaluation.

I. Introduction

Book rental services traditionally depend on centralized platforms, which can introduce high fees, inefficiency, and lack of transparency. These platforms also pose risks such as data breaches and censorship. Blockchain technology, with its decentralized and immutable nature, offers a compelling alternative for building trustless rental ecosystems. This project implements a decentralized book rental platform, allowing users to list, rent, and return books securely while automating deposits, payments, and penalties through smart contracts.

II. Objectives

- To design and implement a decentralized, automated book rental system without intermediaries.

- To ensure transparency and fairness in rental transactions using smart contracts.

- To minimize on-chain data and leverage IPFS for scalable, cost-effective

storage of book images and metadata.

- To provide a user-friendly and responsive interface for seamless user interaction.

- To enforce security and privacy best practices throughout the platform.

III. System Architecture

A. Overview

The platform consists of three main components:

- **Smart Contract Layer (Solidity/Ethereum)**

- **Decentralized Storage (IPFS)**

- **Frontend Application (React, ethers.js, MetaMask integration)**

B. Smart Contract Design

- **Core Structs & Mappings:**
  The contract defines a `Book` struct containing essential data: owner, title, author, IPFS hash (for images/metadata), rental price, deposit, rental status, renter, and timestamps. Mappings allow efficient lookup and management of books and rentals.

- **Key Functions:**

    - `listBook`: Allows owners to list new books with required details and an IPFS hash.

    - `rentBook`: Lets users rent available books by paying the rental fee and deposit. The contract locks the book and funds.

    - `returnBook`: Handles book returns, calculates if the return is late, and deducts penalties from the deposit if necessary.

    - `withdrawEarnings`: Enables owners to withdraw accumulated rental fees and penalties.

    - **Events:** Emits `BookListed`, `BookRented`, and `BookReturned` for frontend synchronization.

- **Security Features:**

    - Uses OpenZeppelin's `ReentrancyGuard` to prevent reentrancy attacks.

    - Implements the Checks-Effects-Interactions (CEI) pattern in all state-changing functions.

    - Strict access control: only owners can list books, only current renters can return them.

C. Decentralized Storage (IPFS)

- **Purpose:**
  Book images and extended metadata are stored off-chain on IPFS to reduce gas costs and improve scalability.

- **Integration:**
  The frontend uploads images to IPFS (via Pinata), and the resulting CID is stored in the smart contract. The frontend retrieves images and metadata using these CIDs.

D. Frontend Application

- **Wallet Integration:**
  Users connect their MetaMask wallet; all transactions are signed and sent through MetaMask.

- **Book Discovery:**
  The marketplace fetches available books from the contract, retrieves

metadata/images from IPFS, and displays them.

- **Rental Management:**
  Users can view their rentals, return books, and track deposit status.

- **Admin Features:**
  Book owners can manage listings and withdraw earnings.

- **User Experience:**
  The app is responsive (React-Bootstrap), provides clear transaction feedback, and handles errors gracefully.

E. Development & Testing Tools

- **Truffle:** For contract development, deployment, and testing.

- **Ganache:** Local blockchain for rapid iteration.

- **Ethers.js:** For blockchain interactions in the frontend.

- **Jest/React Testing Library:** For frontend unit and integration tests.

IV. System Processes

A. Book Listing Process

1. **Owner Action:**
   The owner connects their wallet and submits a form with book details and an image.

2. **IPFS Upload:**
   The image is uploaded to IPFS via the frontend, which returns a CID.

3. **Smart Contract Call:**
   The owner calls `listBook` with the book details and the IPFS CID.

4. **Event Emission:**
   The contract emits a `BookListed` event, which the frontend listens for to update the UI.

B. Renting Process

1. **Renter Action:**
   The renter browses available books, selects one, and initiates a rental transaction.

2. **Payment:**
   The renter pays the rental fee plus a refundable deposit via MetaMask.

3. **Smart Contract Logic:**
   The contract checks availability, locks the book, and holds the deposit.

4. **Event Emission:**
   The contract emits a `BookRented` event for UI update.

C. Return & Penalty Process

1. **Renter Action:**
   The renter initiates a return transaction before or after the due date.

2. **Smart Contract Logic:**
   The contract checks if the book is returned on time. If late, a penalty is deducted from the deposit.

3. **Refund:**
   The remaining deposit (if any) is refunded to the renter.

4. **Event Emission:**
   The contract emits a `BookReturned` event.

D. Earnings Withdrawal

- Owners can withdraw their earnings and collected penalties at any time by calling `withdrawEarnings`.

V. Smart Contract Implementation

A. Data Structures

- **Book Struct:**
  Contains all relevant book and rental information.

- **Mappings:**

  - `bookId => Book`

  - `user address => list of owned/rented books`

B. Functionality

- **Listing:**
  Only the owner can list a book. The function checks for valid input and stores the book details.

- **Renting:**
  Checks book availability and payment correctness. Locks the book and updates mappings.

- **Returning:**
  Checks if the caller is the current renter. Calculates penalties for late returns and refunds the appropriate amount.

- **Withdrawals:**
  Only the book owner can withdraw earnings.

C. Security

- **Reentrancy Guard:**
  All Ether-transferring functions use `nonReentrant`.

- **Access Control:**
  Functions check `msg.sender` to ensure only authorized users can call them.

- **Input Validation:**
  All user inputs are validated to prevent malicious or incorrect data.

VI. Frontend Implementation

A. Technology Stack

- **React.js** for the UI

- **ethers.js** for blockchain interaction

- **MetaMask** for wallet connection

- **IPFS (Pinata)** for off-chain storage

- **React-Bootstrap** for styling and layout

B. Key Features

- **Wallet Connection:**
  Prompts users to connect their wallet. Handles account and network changes gracefully.

- **Book Listing:**
  Owners can list books with images. Images are uploaded to IPFS, and the CID is stored on-chain.

- **Marketplace:**
  Displays all available books with real-time updates via contract events.

- **Rental Management:**
  Users can view their current rentals, return books, and see deposit/refund status.

- **Admin Panel:**
  Owners can manage their listings and withdraw earnings.

- **Error Handling:**
  Provides clear feedback for transaction status, errors, and network issues.

C. User Experience

- Responsive design for mobile and desktop.

- Loading indicators and disabled buttons during transactions.

- Etherscan links for transaction verification.

VII. Security and Privacy Considerations

- **Reentrancy Protection:**
  All state-changing and Ether-transferring functions use the `nonReentrant` modifier.

- **Checks-Effects-Interactions Pattern:**
  State is updated before any external calls.

- **Minimal On-Chain Data:**
  Only essential data (addresses, book metadata, IPFS CID) is stored on-chain.

- **No Personal Data:**
  No names, emails, or sensitive user info are stored on-chain.

- **Comprehensive Testing:**
  Automated tests cover normal and edge cases, including double rentals and incorrect payments.

VIII. Optimization and Gas Efficiency

- **Efficient Data Types:**
  Used `uint256` for all monetary and time values.

- **Structs and Mappings:**
  Chosen for fast access and low gas usage.

- **Event Logging:**
  Used for frontend updates instead of storing extra data.

- **Solidity Optimizer:**
  Enabled during compilation for more efficient bytecode.

IX. Challenges and Solutions

- **Real-Time Data Sync:**
  Challenge: Keeping the UI in sync with blockchain events.
  Solution: Listened for contract events and used fallback polling.

- **IPFS Reliability:**
  Challenge: Occasional upload failures.
  Solution: Added client-side image compression, retry logic, and multiple gateway support.

- **Wallet/Network Changes:**
  Challenge: Users switching accounts or networks.
  Solution: Used React hooks to reset state and prompt reconnection.

- **Gas Estimation:**
  Challenge: Failed transactions due to low gas.
  Solution: Set manual gas limits for complex functions.

- **Cross-Browser Support:**
  Challenge: Inconsistent behavior on Safari/old browsers.

Solution: Added polyfills and mobile-specific MetaMask handling.

X. Testing and Evaluation

- **Unit and Integration Tests:**
  All core contract functions and edge cases are tested.

- **Frontend Testing:**
  Used Jest and React Testing Library.

- **User Feedback:**
  Beta testers found the UI intuitive and the process transparent.

- **Performance:**
  Average transaction time under 2 minutes; gas costs reduced by 40% compared to naive implementations.

XI. Conclusion

The Decentralized Book Rental Platform demonstrates the power of blockchain and decentralized storage to create a fair, efficient, and transparent rental ecosystem. By automating trust, reducing costs, and ensuring data privacy, the platform offers a compelling alternative to traditional rental systems. Future enhancements may include Layer 2 scaling, a reputation system, and support for digital book rentals.

XII. References

1. Book Store DApp - https://github.com/masterpranay1/Book-Store-DApp
2. BlockBnb – Decentralized House Rental Platform - https://github.com/Anchit1909/blockbnb-decentralized-house-rental-platform
3. “A Blockchain-Based Decentralized Booking System” - Knowledge Engineering Review, Cambridge University Press.