LIBRARY MANAGEMENT

1. INTRODUCTION

1.1Project Overview

The Blockchain-Powered Library Management System project aims to enhance library operations by leveraging blockchain technology to improve efficiency, security, and transparency. It offers features such as blockchain-based user registration and authentication, catalog management with an immutable ledger, streamlined borrowing and returning through smart contracts, transparent transaction records, secure interlibrary loans, automated fine management, reservation systems, user ratings and reviews, secure document delivery, robust data privacy, data-driven analytics, and decentralized governance. The system benefits from enhanced security, transparency, and a streamlined user experience, leading to operational efficiency, reliable record-keeping, and data-driven decision-making, ultimately revolutionizing library services for both administrators and users.

1.2 Purpose

The purpose of a blockchain-powered library management system (LMS) is to make libraries more secure, transparent, and efficient, while also reducing costs.Blockchain is a distributed ledger technology that allows for secure, transparent, and tamper-proof data storage. This makes it ideal for use in a variety of applications, including library management.A blockchain-powered LMS could be used to store and manage all aspects of library data, including book records, user records, and circulation records. This data would be stored on the blockchain in a secure and tamper-proof manner.Users could access library data through a web or mobile application. The application would authenticate users using blockchain-based authentication.All data transactions on a blockchain are recorded and publicly visible. This would ensure that all library activity is transparent and auditable.Blockchain could help to streamline library operations by automating tasks such as book checkouts and returns. This would free up library staff to focus on other tasks, such as providing assistance to users.Blockchain could also help libraries to save money on operational costs.

2. EXISTING PROBLEM

2.1 Existing Problem

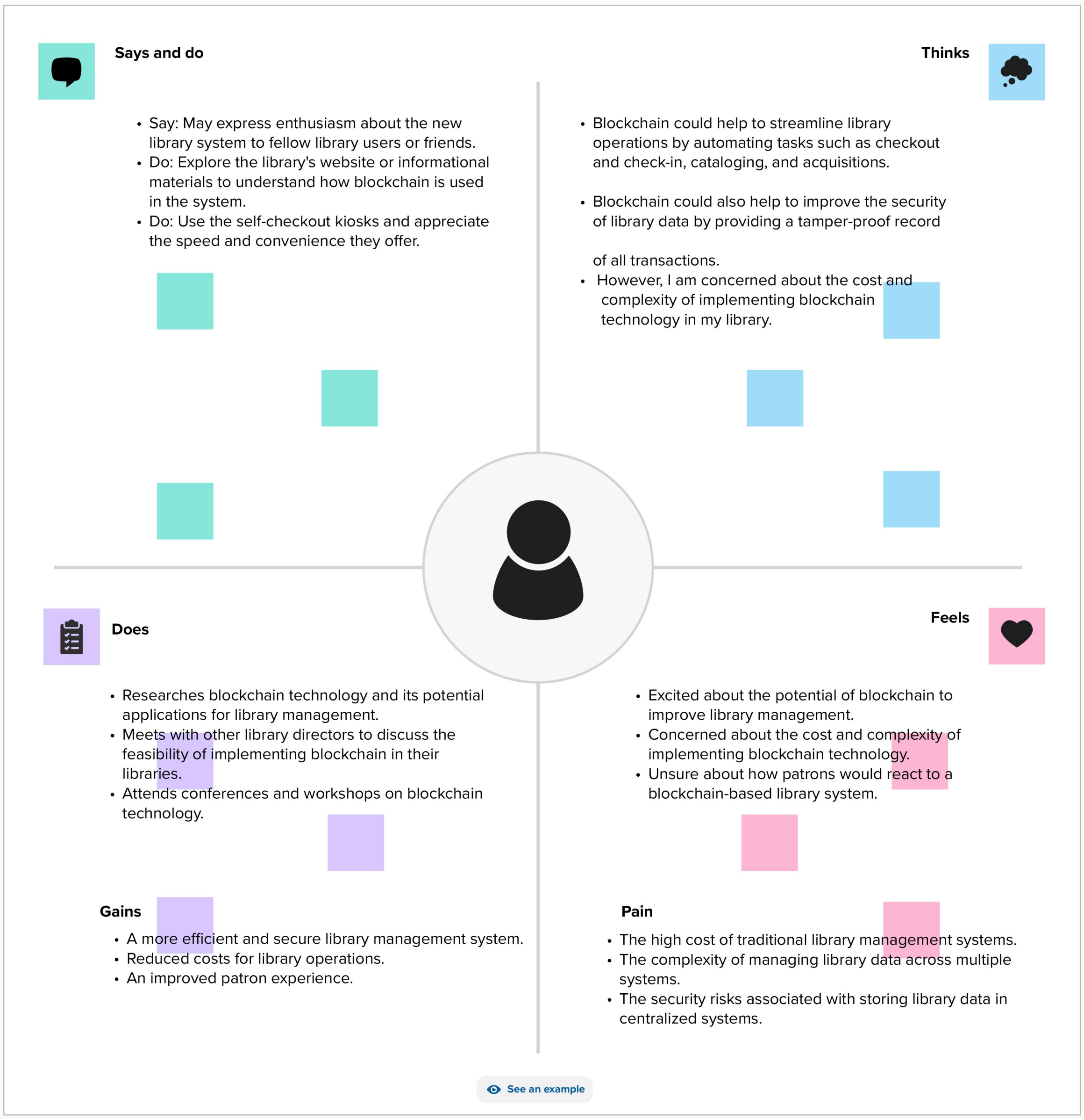
Library Management Systems face numerous challenges, including limited accessibility and the reliance on manual cataloging and updates, resulting in data inaccuracies and inefficiencies. Data security concerns arise from centralized data storage, inefficient resource tracking, and inadequate user engagement features. Manual check-out and return processes lead to delays and errors, while interlibrary connectivity issues hinder resource sharing. Operational complexities and inconsistent management of overdues and fines further compound the problems.Additionally, the lack of robust analytics and reporting capabilities limits data-driven decision-making. These issues underscore the need for more advanced Library Management Systems that can enhance accessibility, security, efficiency, and user satisfaction.

2.2 Problem Statement Definition

Libraries today face critical challenges in terms of efficiency, security, and transparency in their operations. Traditional Library Management Systems often struggle to provide real-time data updates, protect user data from security breaches, and ensure transparent and tamper-proof transaction records. These limitations hinder the overall library experience and fail to meet the expectations of users in a digital age.The problem at hand involves the inefficiencies and security vulnerabilities inherent in conventional Library Management Systems. These systems often rely on manual processes for cataloging and updating data, resulting in inaccuracies and operational inefficiencies. Centralized data storage raises concerns about data security and privacy, leaving user information vulnerable to breaches. Furthermore, transaction records may lack transparency and immutability, impacting the integrity of library operations. As a result, there is a pressing need for a Blockchain-Powered Library Management System that can address these issues by leveraging blockchain technology to enhance efficiency, security, and transparency in library operations, ultimately delivering a modern and user-centric library experience.

3. IDEATION AND PROPOSED SOLUTION

3.1 Empathy Map Canvas



Says and do

- Say: May express enthusiasm about the new library system to fellow library users or friends.
- Do: Explore the library's website or informational materials to understand how blockchain is used in the system.
- Do: Use the self-checkout kiosks and appreciate the speed and convenience they offer.

Thinks

- Blockchain could help to streamline library operations by automating tasks such as checkout and check-in, cataloging, and acquisitions.
- Blockchain could also help to improve the security of library data by providing a tamper-proof record

  of all transactions.
- However, I am concerned about the cost and complexity of implementing blockchain technology in my library.

Does

- Researches blockchain technology and its potential applications for library management.
- Meets with other library directors to discuss the feasibility of implementing blockchain in their libraries.
- Attends conferences and workshops on blockchain technology.

Feels

- Excited about the potential of blockchain to improve library management.
- Concerned about the cost and complexity of implementing blockchain technology.
- Unsure about how patrons would react to a blockchain-based library system.

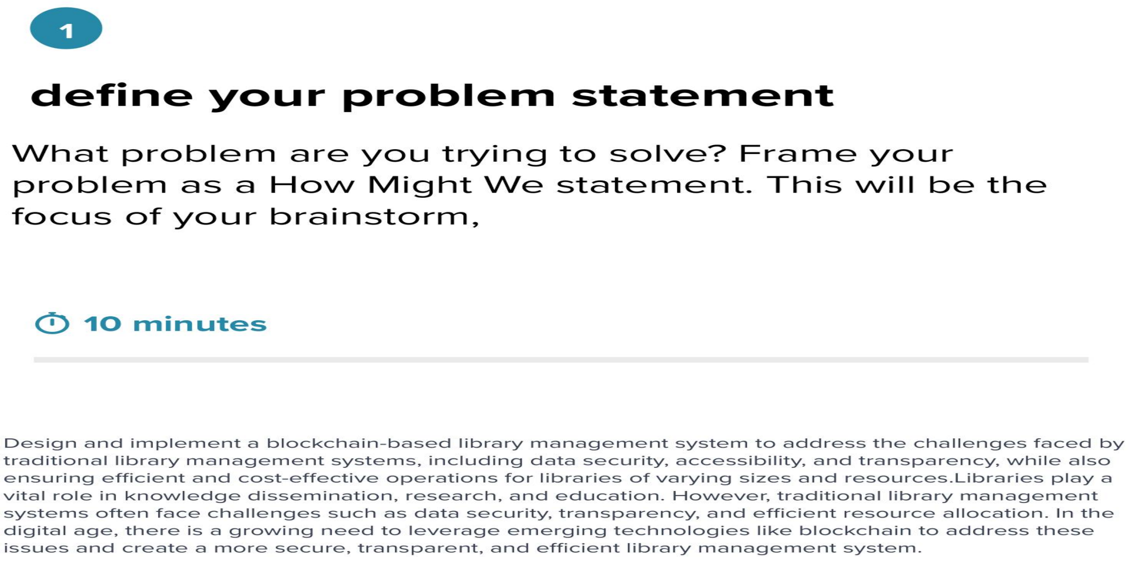
Gains

- A more efficient and secure library management system.
- Reduced costs for library operations.
- An improved patron experience.

Pain

- The high cost of traditional library management systems.
- The complexity of managing library data across multiple systems.
- The security risks associated with storing library data in centralized systems.

3.2 Ideation and Brainstorming



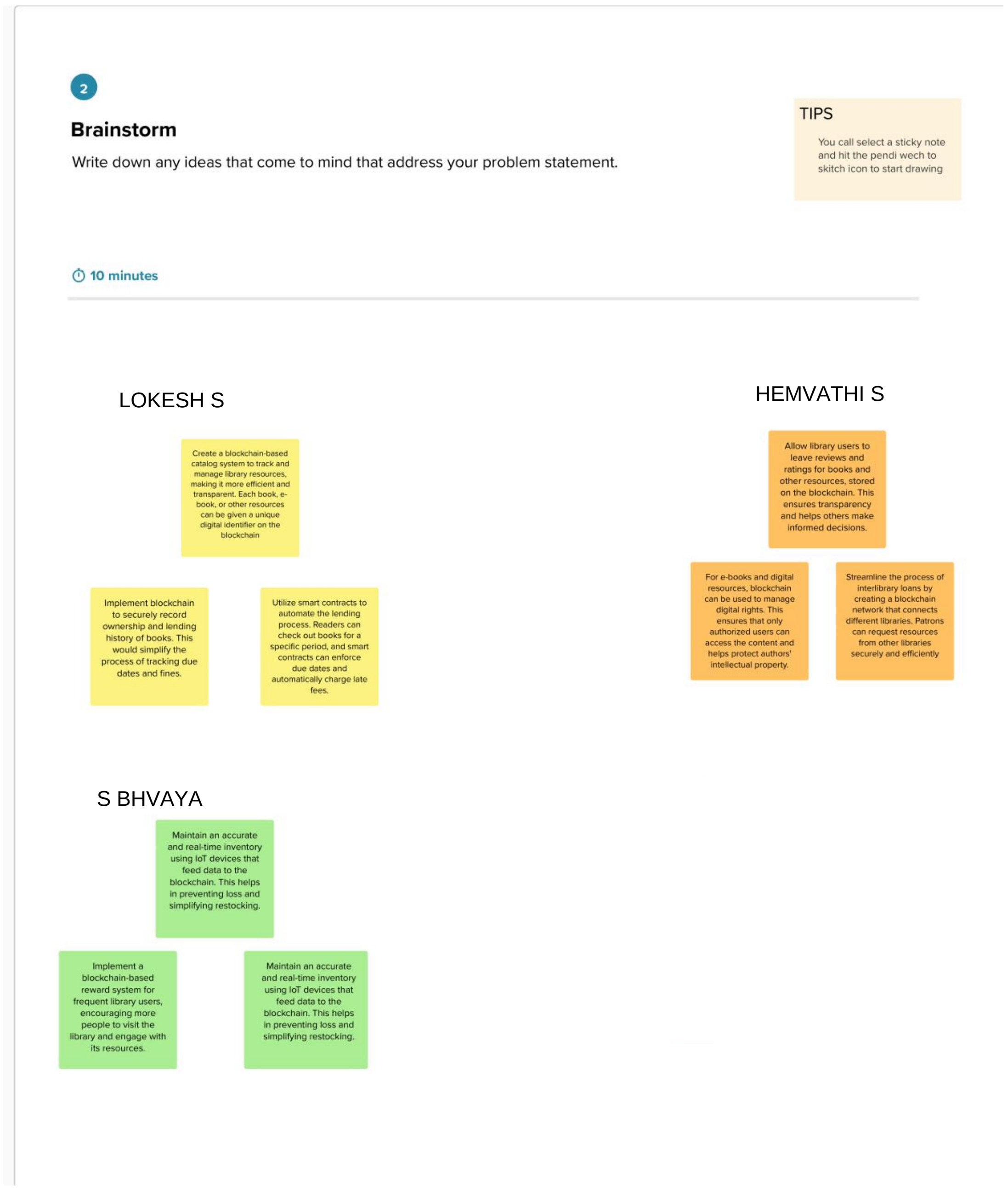
1

define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm,

10 minutes

Design and implement a blockchain-based library management system to address the challenges faced by traditional library management systems, including data security, accessibility, and transparency, while also ensuring efficient and cost-effective operations for libraries of varying sizes and resources.Libraries play a vital role in knowledge dissemination, research, and education. However, traditional library management systems often face challenges such as data security, transparency, and efficient resource allocation. In the digital age, there is a growing need to leverage emerging technologies like blockchain to address these issues and create a more secure, transparent, and efficient library management system.



2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

TIPS

You call select a sticky note and hit the pendi wech to skitch icon to start drawing

10 minutes

LOKESH S

Create a blockchain-based catalog system to track and manage library resources, making it more efficient and transparent. Each book, e-book, or other resources can be given a unique digital identifier on the blockchain

Implement blockchain to securely record ownership and lending history of books. This would simplify the process of tracking due dates and fines.

Utilize smart contracts to automate the lending process. Readers can check out books for a specific period, and smart contracts can enforce due dates and automatically charge late fees.

HEMVATHI S

Allow library users to leave reviews and ratings for books and other resources, stored on the blockchain. This ensures transparency and helps others make informed decisions.

For e-books and digital resources, blockchain can be used to manage digital rights. This ensures that only authorized users can access the content and helps protect authors' intellectual property.

Streamline the process of interlibrary loans by creating a blockchain network that connects different libraries. Patrons can request resources from other libraries securely and efficiently

S BHVAYA

Maintain an accurate and real-time inventory using IoT devices that feed data to the blockchain. This helps in preventing loss and simplifying restocking.

Implement a blockchain-based reward system for frequent library users, encouraging more people to visit the library and engage with its resources.

Maintain an accurate and real-time inventory using IoT devices that feed data to the blockchain. This helps in preventing loss and simplifying restocking.

3

GROUP IDEAS

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

15 minutes

SECURITY

ensures that resources are easily traceable, reducing the chances of loss and enabling quick inventory updates.

Blockchain can be configured to maintain the privacy of library patrons. Personal data can be anonymized or encrypted, ensuring that only authorized personnel have access to specific information.

Blockchain allows for fine-grained access control. Libraries can manage who can view, modify, or add data to the blockchain, enhancing security and privacy.

FEATURES

Enhanced library efficiency and transparency.

Improved patron experience and resource access.

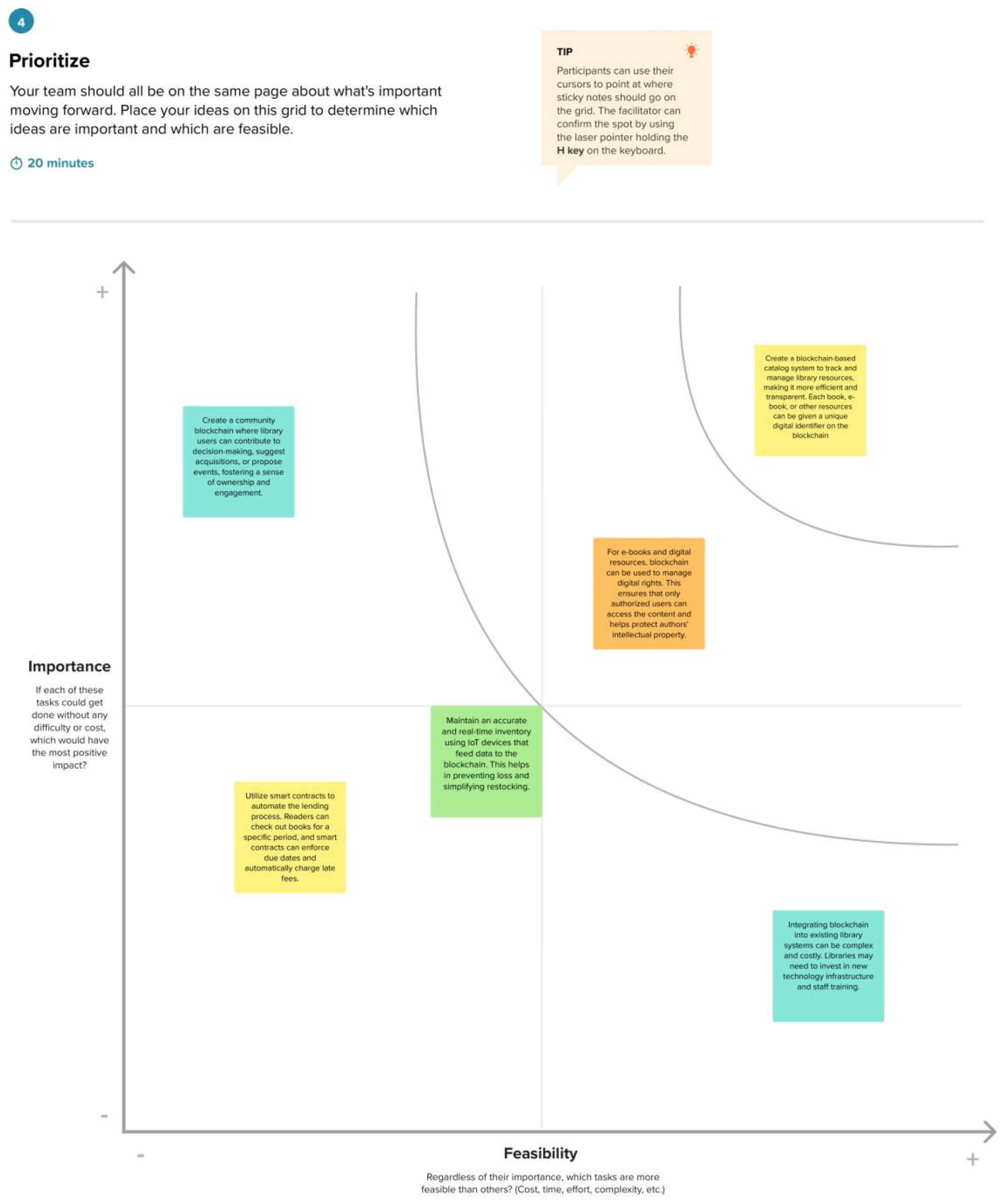
Minimized resource loss and overdue fines

CHALLENGES

Integrating blockchain into existing library systems can be complex and costly. Libraries may need to invest in new technology infrastructure and staff training.

Developing and maintaining a blockchain-based system can be expensive. This includes the cost of blockchain technology, smart contract development, and ongoing maintenance.

Developing a user-friendly interface for the platform


4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.

Create a community blockchain where library users can contribute to decision-making, suggest acquisitions, or propose events, fostering a sense of ownership and engagement.

Create a blockchain-based catalog system to track and manage library resources, making it more efficient and transparent. Each book, e-book, or other resources can be given a unique digital identifier on the blockchain

For e-books and digital resources, blockchain can be used to manage digital rights. This ensures that only authorized users can access the content and helps protect authors' intellectual property.

Importance

If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?

Maintain an accurate and real-time inventory using IoT devices that feed data to the blockchain. This helps in preventing loss and simplifying restocking.

Utilize smart contracts to automate the lending process. Readers can check out books for a specific period, and smart contracts can enforce due dates and automatically charge late fees.

Integrating blockchain into existing library systems can be complex and costly. Libraries may need to invest in new technology infrastructure and staff training.

Feasibility

Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

4 .REQUIREMENT ANALYSIS

4.1 Functional Requirements

User Management:

i. User Registration: Users should be able to create blockchain-based accounts with unique IDs and passwords.
ii. User Authentication: Implement secure authentication mechanisms using blockchain keys.
iii. User Roles: Define different user roles, such as librarians, administrators, and patrons, with varying permissions.

Catalog Management:

i. Resource Upload: Allow library staff to upload, update, and manage the catalog of books, e-books, journals, and other resources.

ii. Immutability: Ensure that catalog data is stored on the blockchain, making it tamper-proof and auditable.
iii. Search and Discovery: Enable users to search for resources, view detailed information, and check availability.

Borrowing and Returning:

i. Check-Out and Return: Enable users to check out and return library items through the system.
ii. Smart Contracts: Implement smart contracts to enforce borrowing rules, due dates, and automated renewals

Transparent Transaction Records:

i. Record Keeping: Record all library transactions on the blockchain, including borrowing, returning, and reservations.
ii. Real-time Updates: Ensure that transaction records are updated in real-time for users to view.

Interlibrary Loans:

i. Interlibrary Loan Requests: Facilitate secure interlibrary loans and resource sharing between different libraries.
ii. Resource Tracking: Allow libraries to track the location and status of resources on loan.

Fines and Overdues:

i. Automatic Fine Calculation: Automatically calculate fines for overdue items using smart contracts.
ii. Notification System: Notify users about fines and due dates through automated alerts.

Reservation System:

i. Resource Reservations: Allow users to reserve books and resources, with priority based on a transparent reservation queue.
ii. Reservation Management: Enable library staff to manage and allocate reserved resources.

User Ratings and Reviews:

i. User Feedback: Enable users to rate and review library resources, contributing to a user-driven rating system.
ii. Review Moderation: Implement a moderation system for user-generated content.

Data Privacy and Access Control:

i. Access Control: Implement role-based access control to safeguard user data and comply with privacy regulations.
ii. User Data Protection: Encrypt and protect user data stored on the blockchain.

Analytics and Reporting:

i. Data Analytics: Generate reports and insights from blockchain data to inform decision-making, resource acquisition, and library management.
ii. Custom Reports: Allow library staff to create custom reports based on specific criteria.

.

These functional requirements are essential for the successful development and implementation of a Blockchain-Powered Library Management System, ensuring that it delivers improved efficiency, security, and transparency in library operations.

4.2 Non-Functional Requirements

Security:

i. Data Security: Ensure the highest level of data security by implementing robust encryption and access control mechanisms to protect user data and transaction records stored on the blockchain.
ii. Privacy Compliance: Adhere to data privacy regulations (e.g., GDPR) to safeguard user privacy and ensure compliance with relevant legal requirements.
iii. Blockchain Security: Choose a secure blockchain platform and implement best practices to protect against hacking, fraud, and unauthorized access.

Performance:

i. Responsiveness: Ensure that the system is responsive and capable of handling concurrent user requests without significant delays or downtimes.
ii. Scalability: Design the system to handle a growing user base and an expanding library catalog efficiently by scaling resources as needed.
iii. Transaction Speed: Optimize blockchain transaction processing for quick borrowing, returning, and reservations.

Reliability:

i. System Uptime: Maintain a high level of system availability to minimize disruptions to library services.
ii. Data Integrity: Guarantee the integrity of blockchain-stored data to prevent data corruption or loss.
iii. Fault Tolerance: Implement mechanisms to recover from system failures or data discrepancies without affecting user experience.

Usability:

i. User-Friendly Interface: Design an intuitive and user-friendly interface that caters to both experienced and novice users, ensuring easy navigation and resource discovery.
ii. Accessibility: Ensure that the system is accessible to individuals with disabilities in compliance with accessibility standards (e.g., WCAG).

Interoperability:

i. Integration: Support integration with external systems, databases, and library networks to enable seamless resource sharing and interlibrary loans.
ii. Standard Compliance: Adhere to industry standards (e.g., MARC, Z39.50) for resource description and data exchange.

Compliance:

i. Regulatory Compliance: Ensure compliance with all relevant library and data protection regulations and standards.
ii. Blockchain Governance: Implement mechanisms for adhering to blockchain governance protocols and community guidelines.

Performance Testing:

i. Load Testing: Perform load testing to ensure the system can handle peak usage without degradation in performance.
ii. Stress Testing: Assess the system's behavior under extreme conditions to identify potential weaknesses.

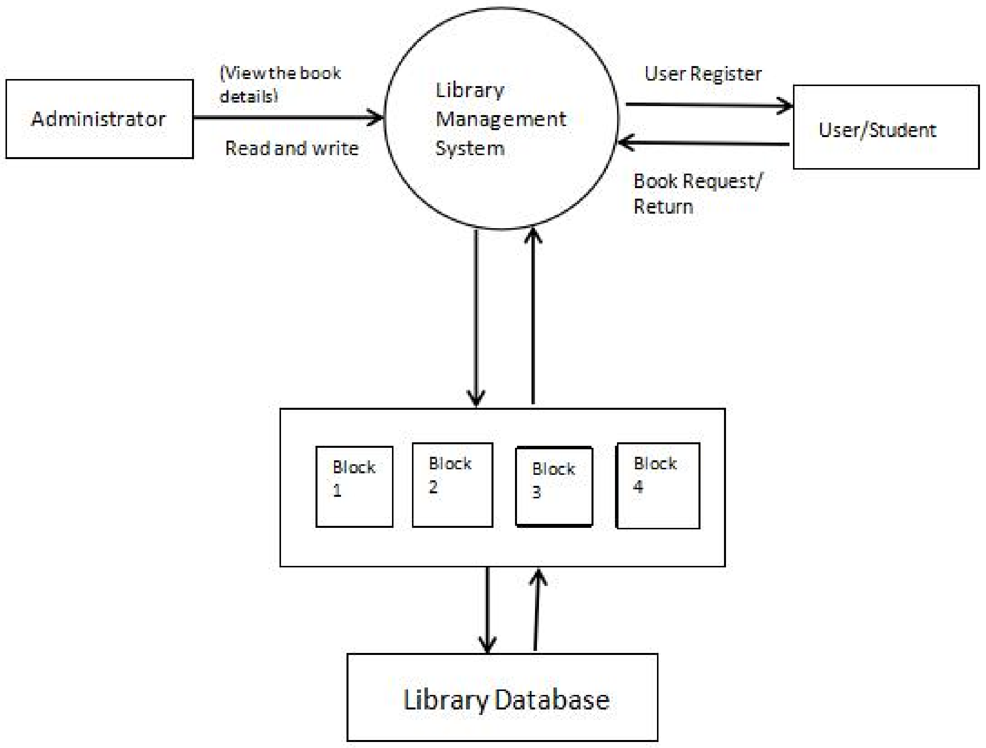
Data Backup and Recovery:

i. Data Backup: Implement regular, automated data backups to protect against data loss.
ii. Disaster Recovery: Establish a disaster recovery plan to quickly restore the system in case of unexpected events.

These non-functional requirements are crucial for the successful deployment and operation of a Blockchain-Powered Library Management System, guaranteeing a secure, reliable, and user-friendly experience while adhering to legal and regulatory standards.

5.PROJECT DESIGN

5.1 Data Flow Diagram & User Stories



LIBRARY MANAGEMENT IN BLOCKCHAIN

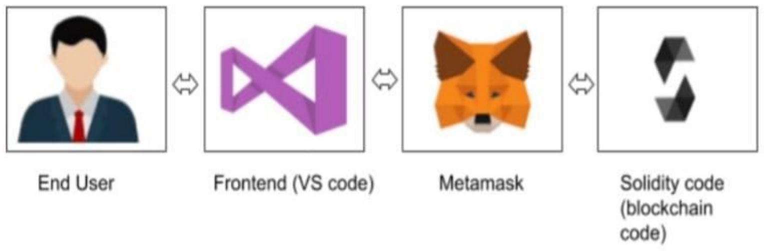
Story 1

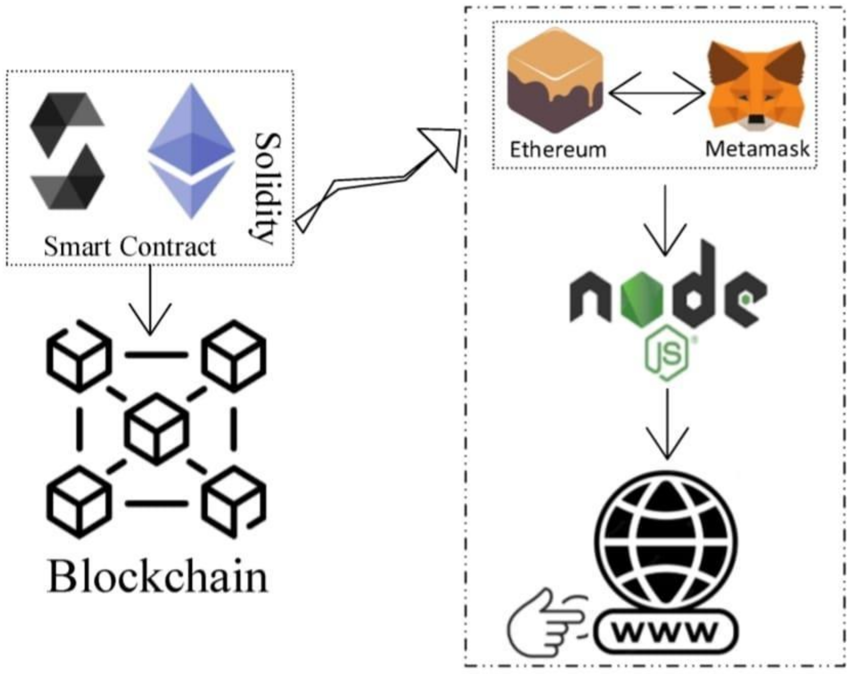
John, a university student, logs in using secure blockchain credentials and searches for a book. He initiates a borrowing request, and a smart contract on the blockchain confirms it. The system updates the catalog and notifies John of the due date. He retrieves the book from the library using location information, and upon return, another smart contract verifies it. Transaction data is recorded for analytics, allowing library staff to monitor usage. John's experience showcases the Blockchain-Powered Library Management System's efficiency and security, benefiting both users and administrators.

Story 2

Sarah, a graduate student, logs into the library system with her secure blockchain credentials. She reserves a high-demand book, and the system records her reservation on the blockchain. The transparent reservation queue allows her to monitor her position. Sarah also provides feedback on her library experiences, securely stored on the blockchain. Notifications keep her updated on her reservation's status. When the book becomes available, she retrieves it. Library staff can analyze user feedback to improve services and resources, enhancing the overall library experience.

5.2 Solution Architecture

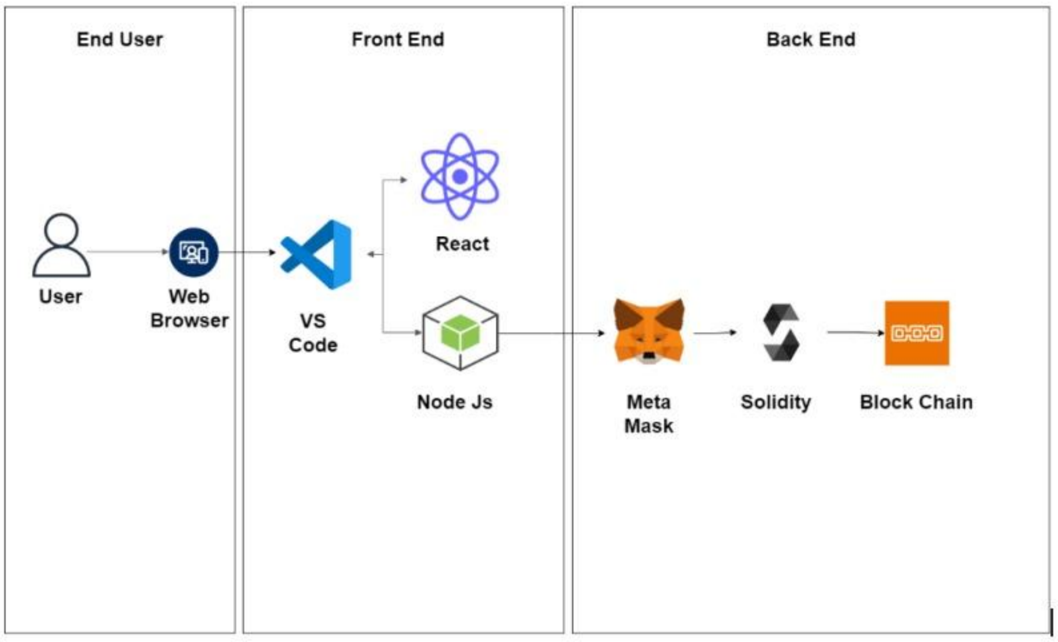




Interaction between web and the Contract

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



GENERAL ARCHITECTURE

6.2 Sprint Planning and Estimation

Sprint planning involves selecting work items from the product backlog and committing to completing them during the upcoming sprint

Reviewing Product Backlog: Our project team, consisting of the product owner, scrum master, and development team, regularly reviews the items in the product backlog.We evaluate user stories and technical tasks, taking into account the project's evolving needs and priorities.

Setting Sprint Goals: Based on the product backlog, our team establishes clear sprint goals. These goals guide the team's efforts during the sprint and ensure alignment with the broader project objectives.

Breaking Down User Stories: User stories and tasks are further decomposed into smaller, actionable sub-tasks. This detailed breakdown helps create a comprehensive plan for the sprint.

Estimating Work: Our development team employs agile estimation techniques, such as story points and t-shirt sizes, to estimate the effort required for each task. These estimates guide the team in understanding the scope and complexity of work for the sprint.

Sprint Backlog: The selected user stories and tasks, along with their estimates, constitute the sprint backlog. This forms the basis for what the team will work on during the sprint.

Estimation Techniques

Story Points: Story points serve as a relative measure of the complexity and effort needed to complete a task. Tasks are assigned story point values based on their complexity compared to reference tasks.

T-Shirt Sizes: To provide a quick and high-level estimate of effort, tasks are categorized into t-shirt sizes, such as small, medium, and large. This approach simplifies the estimation process for less complex tasks.

6.1 Sprint Delivering Schedule

Week 1: Establish the Core

• Set up the basic blockchain infrastructure.

• Implement a minimal user registration and authentication system.

• Develop a rudimentary transaction recording feature.

• Focus on fundamental security measures.

Week 2: Expand and Enhance

• Extend transaction recording to support more transaction types.

• Add basic real-time updates for transaction status.

• Enhance user authentication with multi-factor authentication.

• Begin developing a user dashboard.

Week 3: Finalize and Prepare

• Complete the user dashboard with additional features.

• Perform minimal compliance checks.

• Conduct basic testing and issue resolution.

• Create essential user support resources.

7. CODING AND SOLUTIONING

7.1 Feature 1

Smart contract (Solidity)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract BookRegistry {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can perform this action");
        _;
    }

    struct Book {
        string title;
        string author;
        address currentOwner;
    }

    mapping(uint256 => Book) public books;
    uint256 public bookCount;

    event BookAdded(uint256 indexed bookId, string title, string author, address
indexed owner);
    event OwnershipTransferred(uint256 indexed bookId, address indexed
previousOwner, address indexed newOwner);

    function addBook(uint256 registration, string memory _title, string memory
_author) external onlyOwner {
        books[registration] = Book(_title, _author, owner);
        bookCount++;
        emit BookAdded(registration, _title, _author, owner);
    }

    function transferOwnership(uint256 registrationId, address _newOwner) external {
        require(_newOwner != address(0), "Invalid address");
        require(_newOwner != books[registrationId].currentOwner, "The new owner is
the same as the current owner");
        require(msg.sender == books[registrationId].currentOwner, "Only the current
owner can transfer ownership");

        address previousOwner = books[registrationId].currentOwner;
        books[registrationId].currentOwner = _newOwner;

        emit OwnershipTransferred(registrationId, previousOwner, _newOwner);
    }

    function getBookDetails(uint256 registrationId) external view returns (string
memory, string memory, address) {

        Book memory book = books[registrationId];
        return (book.title, book.author, book.currentOwner);
    }
}
```

The provided Solidity code defines a BookRegistry smart contract for managing a decentralized book registry. It allows the owner to add books, transfer ownership, and retrieve book details. Books are represented by a Book struct, and events are emitted for book additions and ownership transfers. The code ensures data immutability and transparency on the blockchain.

7.2 Feature 2

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import { contract } from "./connector";

function Home() {
  const [Id, setId] = useState("");
  const [Title, setTitle] = useState("");
  const [Author, setAuthor] = useState("");
  const [TranId, setTranId] = useState("");
  const [Owner, setOwner] = useState("");
  const [BookId, setBookId] = useState("");
  const [BookDet, setBookDet] = useState("");
  const [Wallet, setWallet] = useState("");




  const handleId = (e) => {
    setId(e.target.value)
  }

  const handleTitle = (e) => {
    setTitle(e.target.value)
  }

  const handleAuthor = (e) => {
    setAuthor(e.target.value)
  }

  const handleAddBook = async () => {
    try {
      let tx = await contract.addBook(Id.toString(), Title, Author)
      let wait = await tx.wait()
      alert(wait.transactionHash)
    } catch (error) {
      alert(error)
    }
  }

  const handleTid = (e) => {
    setTranId(e.target.value)
  }

  const handleNewOwner = (e) => {
    setOwner(e.target.value)
  }
  const handleTransfer = async () => {
    try {
      let tx = await contract.transferOwnership(TranId.toString(), Owner)
      let wait = await tx.wait()
      console.log(wait);
      alert(wait.transactionHash)
    } catch (error) {
      alert(error)
    }
  }

  const handleBookId = (e) => {
    setBookId(e.target.value)
  }

  const handleBookDetails = async () => {
    try {
      let tx = await contract.getBookDetails(BookId.toString())
      let arr = []
      tx.map(e => arr.push(e))

      setBookDet(arr)
    } catch (error) {
      alert(error)
      console.log(error);
    }
  }

  const handleWallet = async () => {
    if (!window.ethereum) {
      return alert('please install metamask');
    }

    const addr = await window.ethereum.request({
      method: 'eth_requestAccounts',
    });

    setWallet(addr[0])

  }

 return (
  <div>
   <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Book Details On
Blockchain</h1>
     {!Wallet ?

       <Button onClick={handleWallet} style={{ marginTop: "30px", marginBottom:
"50px" }}>Connect Wallet </Button>
       :
        <p style={{ width: "250px", height: "50px", margin: "auto", marginBottom:
"50px", border: '2px solid #2096f3' }}>{Wallet.slice(0, 6)}....{Wallet.slice(-6)}</p>
      }
   <Container>
    <Row>
     <Col style={{marginRight:"100px"}}>
      <div>


       <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleId} type="number" placeholder="Enter Registration number"
value={Id} /> <br />

      <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleTitle} type="string" placeholder="Enter Book Title" value={Title}
/> <br />

                <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleAuthor} type="string" placeholder="Enter Book Author"
value={Author} /><br />

       <Button onClick={handleAddBook} style={{ marginTop: "10px" }}
variant="primary">Add book</Button>
      </div>
     </Col>
      <Col>
        <div>
                <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleTid} type="number" placeholder="Enter Registration Id"
value={TranId} /> <br />

                <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleNewOwner} type="string" placeholder="Enter New owner
metamask address" value={Owner} /><br />

           <Button onClick={handleTransfer} style={{ marginTop: "10px" }}
variant="primary">Transfer book ownership</Button>

        </div>
     </Col>
   </Row>
   <Row style={{marginTop:"100px"}}>
     <Col >
      <div style={{ margin:"auto" }}>
        <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleBookId} type="number" placeholder="Enter book Id"
value={BookId} /><br />

       <Button onClick={handleBookDetails} style={{ marginTop: "10px" }}
variant="primary">Get Identity Details</Button>
```

```
      {BookDet ? BookDet?.map(e => {
         return <p>{e.toString()}</p>
       }) : <p></p>}
      </div>
     </Col>
   </Row>


   </Container>

  </div>
 )
}
```

export default Home;

ContractABI (Application Binary Interface):

The abi variable holds the ABI of an Ethereum smart contract.ABIs are essential for encoding and decoding function calls and data when interacting with the Ethereum blockchain.

MetaMask Check:

The code first checks whether the MetaMask wallet extension is installed in the user's browser. If MetaMask is not detected, it displays an alert notifying the user that MetaMask is not found and provides a link to download it.

Ethers.js Configuration:

It imports the ethers library, which is a popular library for Ethereum development.It creates a provider using Web3Provider, which connects to the user's MetaMask wallet and provides access to Ethereum. It creates a signer to interact with The Ethereum blockchain onbehalfoftheuser.ItdefinesanEthereumcontractaddress and sets up the contract object using ethers.Contract, allowing the JavaScript code to interact with the contract's functions.In summary, this code is used for interacting with an Ethereum smart contract through MetaMask and ethers.js. It configures the necessary Ethereum provider and signer for communication with the blockchain and sets up a contract object for executing functions and fetching data from the specified contract address using the providedABI.

8. PERFORMANCE TESTING

8.1 Performance Metrics

Smart Contract Libraries: Evaluate the performance of smart contract libraries available for your blockchain platform. Consider factors like ease of use, security, and efficiency when creating and deploying smart contracts.

Blockchain Protocol: Assess the performance of the underlying blockchain protocol, whether it's Ethereum, Hyperledger, or another platform. Consider factors like transaction speed, scalability, and consensus mechanism.

Security: Measure the security of the blockchain platform, including its resistance to attacks and vulnerabilities. Assess the availability of security libraries and practices for developing secure smart contracts.

Interoperability: Evaluate how well the blockchain platform integrates with other systems and networks. A strong library management system should support interoperability with various APIs and tools.

Documentation and Resources: Consider the availability and quality of documentation and educational resources for developers. Comprehensive documentation and tutorials can enhance performance.

Community and Support: A thriving community and support system are crucial for addressing issues and optimizing performance. Assess the availability of forums, developer communities, and support channels.

Tooling: Evaluate the performance of development tools and frameworks for blockchain. These tools can significantly impact the speed and efficiency of development.

Testing and Simulation: Implement testing and simulation libraries to measure and improve the performance of smart contracts and the blockchain network.

Scalability Solutions: Assess the availability of libraries and solutions for scaling the blockchain network, as scalability is a critical performance factor.

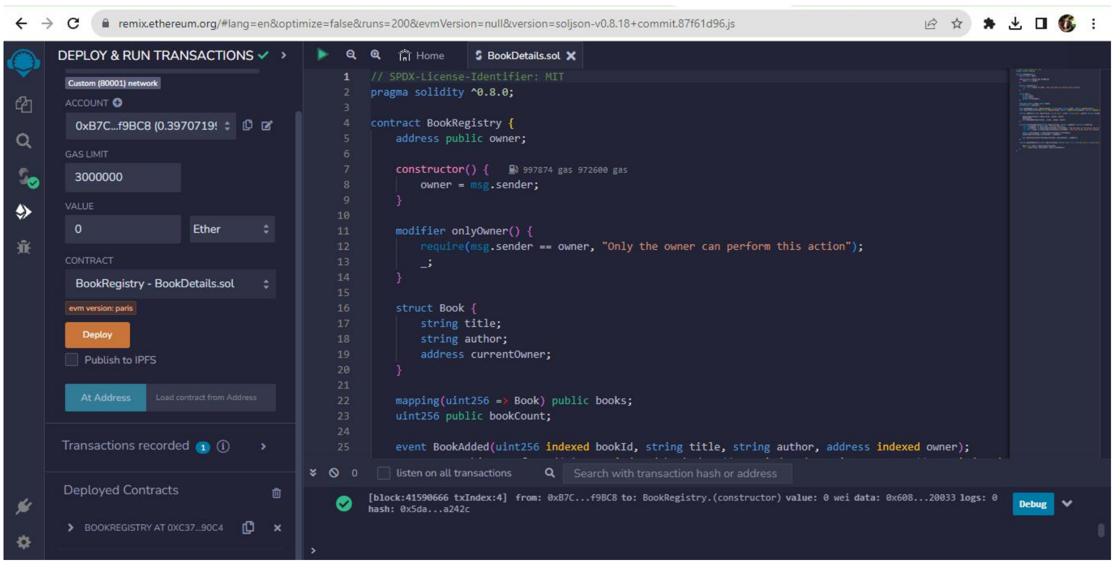
Monitoring and Analytics: Implement libraries for real-time monitoring and performance analytics to identify and resolve issues promptly.

Regulatory Compliance: Consider libraries or tools that help ensure compliance with legal and regulatory requirements in your jurisdiction.

Integration with Databases: Evaluate libraries for integrating blockchain with traditional databases to handle data efficiently

9. RESULTS

9.1 Output Screenshots



```
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
```

DEPLOY & RUN TRANSACTIONS

Custom (80001) network

ACCOUNT: 0xB7C...f9BC8 (0.3970719!

GAS LIMIT: 3000000

VALUE: 0 Ether

CONTRACT: BookRegistry - BookDetails.sol

evm version: paris

Transactions recorded 1

Deployed Contracts

BOOKREGISTRY AT 0XC37...90C4

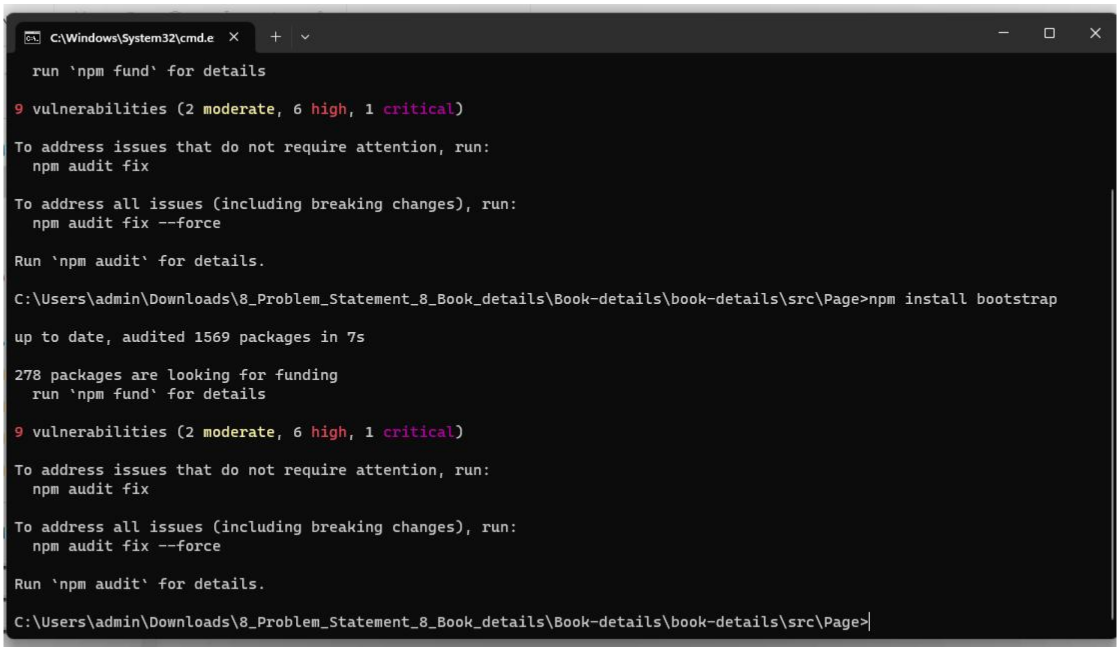
BookDetails.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract BookRegistry {
    address public owner;

    constructor() {   997874 gas 972600 gas
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can perform this action");
        _;
    }

    struct Book {
        string title;
        string author;
        address currentOwner;
    }

    mapping(uint256 => Book) public books;
    uint256 public bookCount;

    event BookAdded(uint256 indexed bookId, string title, string author, address indexed owner);
```

```
[block:41590666 txIndex:4] from: 0xB7C...f9BC8 to: BookRegistry.(constructor) value: 0 wei data: 0x608...20033 logs: 0
hash: 0x5da...a242c
```

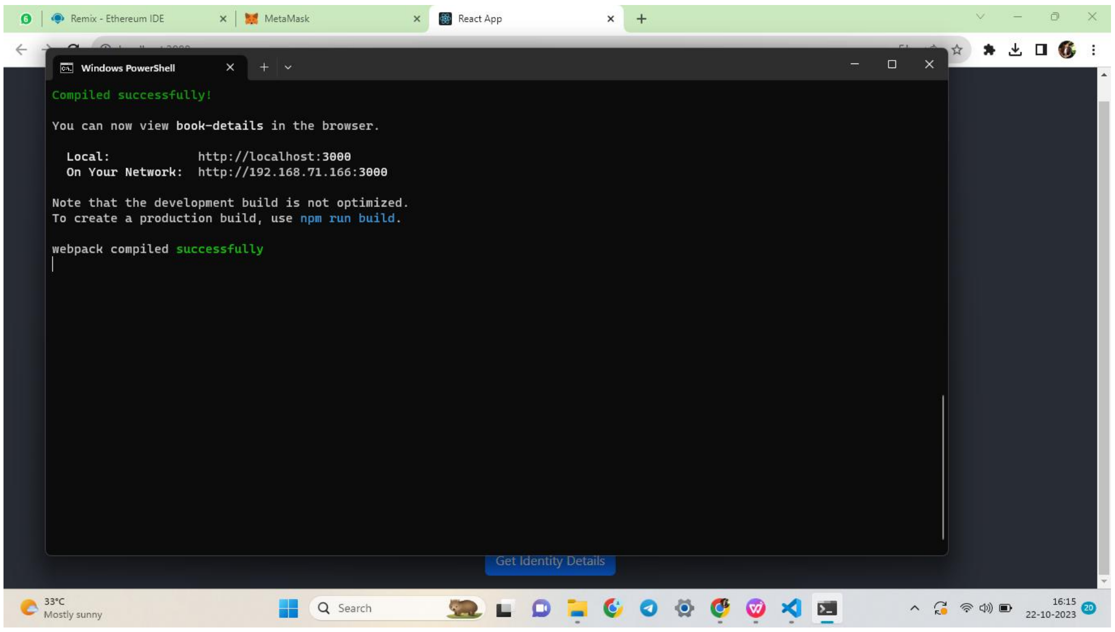
Home
Deploy
Publish to IPFS
At Address
Load contract from Address
listen on all transactions
Search with transaction hash or address
Debug

CREATINGASMART CONTRACT



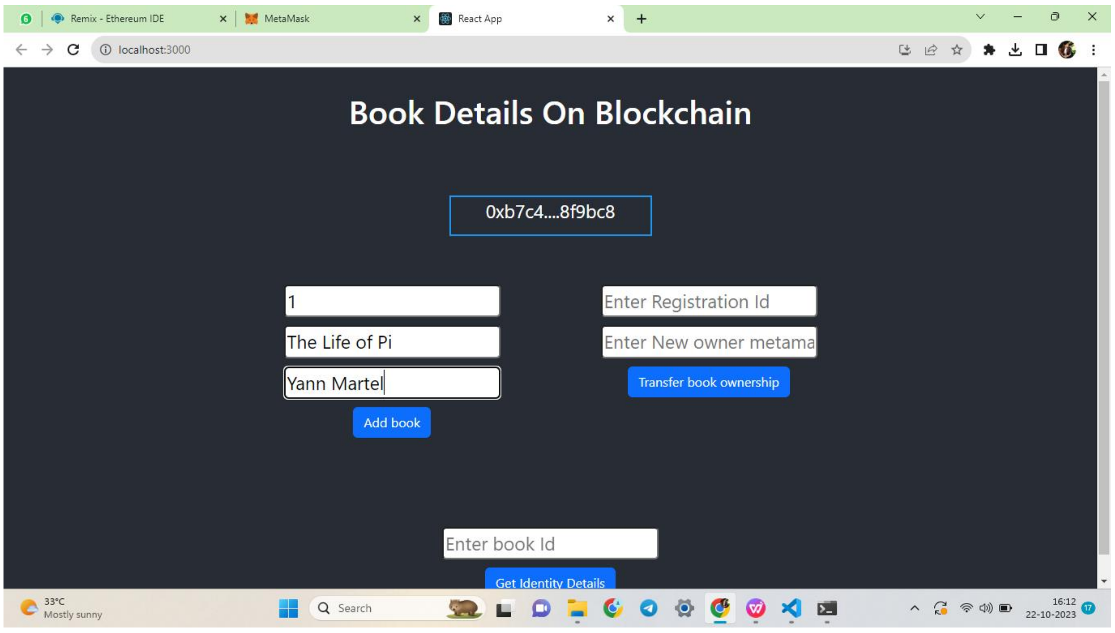
```
  run `npm fund` for details

9 vulnerabilities (2 moderate, 6 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

C:\Users\admin\Downloads\8_Problem_Statement_8_Book_details\Book-details\book-details\src\Page>npm install bootstrap

up to date, audited 1569 packages in 7s

278 packages are looking for funding
  run `npm fund` for details

9 vulnerabilities (2 moderate, 6 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

C:\Users\admin\Downloads\8_Problem_Statement_8_Book_details\Book-details\book-details\src\Page>
```

INSTALLING DEPENDENCIES



```
Compiled successfully!

You can now view book-details in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.71.166:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

HOSTING THE SITE LOCALLY



Book Details On Blockchain

0xb7c4....8f9bc8

1

The Life of Pi

Yann Martel

OUTPUT SCREEN

10. ADVANTAGES AND DISADVANTAGES

10.1 Advantages

Enhanced Security: Blockchain's decentralized and immutable ledger ensures that library records are secure and tamper-proof, reducing the risk of data breaches and fraud.

Digital Identity and Access Control: Blockchain can be used for managing user identities and access to library resources. It allows for more robust authentication and authorization mechanisms.

Smart Contracts for Transactions: Smart contracts can automate and streamline processes like borrowing and returning books, fines, and reservations, reducing administrative overhead.

Interlibrary Loans: Blockchain can facilitate interlibrary loans by providing a transparent and secure platform for tracking borrowed materials across multiple libraries.

Supply Chain for Library Materials: Libraries could use blockchain to track the procurement and distribution of materials, ensuring a streamlined supply chain.

10.2 Disadvantages

Scalability: Blockchain's scalability limitations may make it less suitable for managing large and complex code libraries.

Complexity: Implementing blockchain for library management can be technically complex, requiring expertise in blockchain development.

Cost: Running a blockchain network can be costly in terms of both development and ongoing maintenance.

Speed: Blockchain transactions can be slower compared to traditional database systems, which might not be ideal for rapid development and updates.

Adoption: Blockchain technology is still not widespread, and finding developers with expertise in blockchain development can be challenging.

11. CONCLUSION

Blockchain technology holds immense potential for revolutionizing library management in multiple ways. Its decentralized and immutable ledger ensures the security and integrity of library records, minimizing the risk of data breaches and fraud. Blockchain's role in managing digital identities and access control offers robust authentication and authorization mechanisms for users. The automation capabilities of smart contracts simplify processes like book borrowing, returns, fines, and reservations, reducing administrative burdens. Interlibrary loans benefit from blockchain's transparency and security, enabling efficient tracking of borrowed materials across various libraries. Additionally, blockchain can be instrumental in managing copyright and licensing agreements, guaranteeing fair compensation for content creators. The technology also brings transparency to donation tracking, helping libraries monitor the use of funds. For rare and historical materials, blockchain aids in establishing provenance and authenticity. Furthermore, it streamlines the supply chain for library materials, enhancing overall efficiency. However, successful implementation requires thoughtful planning to address the unique needs and challenges of library operations.

In conclusion, implementing blockchain in library management can improve security, transparency, and efficiency. However, it's important to carefully plan and design the system to meet the specific needs and challenges of the library environment.

12. FUTURE SCOPE

Data Security: Blockchain can ensure the security of sensitive library data, such as patron records and digital resources. In the future, as cybersecurity threats evolve, blockchain can play a crucial role in protecting this data.

Digital Rights Management (DRM): With the increasing prevalence of digital books and media in libraries, blockchain can be used to manage digital rights and access control. It can ensure that only authorized users have access to specific digital content.

Interlibrary Loans: Blockchain can streamline interlibrary loans, making it easier for libraries to share resources securely and efficiently. Future developments may include the use of smart contracts to automate the borrowing and lending process.

Provenance and Authentication: Blockchain can be used to verify the authenticity and provenance of rare and historical items in a library's collection. This can be crucial for preserving cultural heritage.

Decentralized Cataloging: Blockchain can facilitate decentralized cataloging systems where libraries worldwide can collaboratively maintain a single, authoritative catalog. This could make it easier for users to find resources.

Digital Preservation: Blockchain can play a role in ensuring the long-term preservation of digital assets in libraries. This includes ensuring the integrity of digital collections for future generations.

Smart Contracts for Acquisitions: Libraries could use smart contracts to automate the acquisition of new materials, ensuring that copyright and licensing agreements are adhered to automatically.

Public Record Transparency: In government and public libraries, Blockchain can enhance transparency and accountability in record-keeping, ensuring that records are tamper-proof and publicly accessible.

Tokenization and Microtransactions: In the future, libraries may explore tokenization to enable microtransactions for accessing content or services, allowing users to pay for what they use with digital tokens.

User Privacy: As concerns about user privacy continue to grow, libraries may use blockchain to give patrons more control over their personal data, including what information is shared for library services

To implement a blockchain project in library management effectively, it's crucial to consider the specific needs of the library, its user base, and the evolving technological landscape. Additionally, collaborating with blockchain experts and staying updated on blockchain developments is vital for a successful project.

12. APPENDIX

Source code

BookDetails.Sol(Smart Contract)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract BookRegistry {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can perform this action");
        _;
    }

    struct Book {
        string title;
        string author;
        address currentOwner;
    }

    mapping(uint256 => Book) public books;
    uint256 public bookCount;

    event BookAdded(uint256 indexed bookId, string title, string author, address
indexed owner);
    event OwnershipTransferred(uint256 indexed bookId, address indexed
previousOwner, address indexed newOwner);

    function addBook(uint256 registration, string memory _title, string memory
_author) external onlyOwner {

        books[registration] = Book(_title, _author, owner);
        bookCount++;
        emit BookAdded(registration, _title, _author, owner);
    }

    function transferOwnership(uint256 registrationId, address _newOwner) external {
        require(_newOwner != address(0), "Invalid address");
        require(_newOwner != books[registrationId].currentOwner, "The new owner is
the same as the current owner");
```

```
        require(msg.sender == books[registrationId].currentOwner, "Only the current
owner can transfer ownership");

        address previousOwner = books[registrationId].currentOwner;
        books[registrationId].currentOwner = _newOwner;

        emit OwnershipTransferred(registrationId, previousOwner, _newOwner);
    }

    function getBookDetails(uint256 registrationId) external view returns (string
memory, string memory, address) {

        Book memory book = books[registrationId];
        return (book.title, book.author, book.currentOwner);
    }
}
```

Connector.js

```
const { ethers } = require("ethers");

const abi = [
 {
  "inputs": [],
  "stateMutability": "nonpayable",
  "type": "constructor"
 },
 {
  "anonymous": false,
  "inputs": [
   {
    "indexed": true,
    "internalType": "uint256",
    "name": "bookId",
    "type": "uint256"
   },
  {
   "indexed": false,
   "internalType": "string",
   "name": "title",
   "type": "string"
  },
  {
   "indexed": false,
   "internalType": "string",
   "name": "author",
   "type": "string"
  },
  {
   "indexed": true,
   "internalType": "address",
   "name": "owner",
   "type": "address"
  }
 ],
 "name": "BookAdded",
 "type": "event"
},
{
 "anonymous": false,
 "inputs": [
  {
   "indexed": true,
   "internalType": "uint256",
   "name": "bookId",
   "type": "uint256"
  },
  {
   "indexed": true,
   "internalType": "address",
   "name": "previousOwner",
   "type": "address"
  },
  {
   "indexed": true,
   "internalType": "address",
   "name": "newOwner",
   "type": "address"
  }
 ],
 "name": "OwnershipTransferred",
 "type": "event"
},
{
 "inputs": [
  {
   "internalType": "uint256",
   "name": "registration",
   "type": "uint256"
  },
  {
   "internalType": "string",
   "name": "_title",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "_author",
   "type": "string"
  }
 ],
 "name": "addBook",
 "outputs": [],
 "stateMutability": "nonpayable",
"type": "function"
},
{
"inputs": [],
"name": "bookCount",
"outputs": [
 {
  "internalType": "uint256",
  "name": "",
  "type": "uint256"
 }
],
"stateMutability": "view",
"type": "function"
},
{
"inputs": [
 {
  "internalType": "uint256",
  "name": "",
  "type": "uint256"
 }
],
"name": "books",
"outputs": [
 {
  "internalType": "string",
  "name": "title",
  "type": "string"
 },
 {
  "internalType": "string",
  "name": "author",
  "type": "string"
  },
  {
   "internalType": "address",
   "name": "currentOwner",
   "type": "address"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "uint256",
   "name": "registrationId",
   "type": "uint256"
  }
 ],
 "name": "getBookDetails",
 "outputs": [
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "address",
   "name": "",
   "type": "address"
   }
  ],
  "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [],
  "name": "owner",
  "outputs": [
   {
    "internalType": "address",
    "name": "",
    "type": "address"
   }
  ],
  "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [
   {
    "internalType": "uint256",
    "name": "registrationId",
    "type": "uint256"
   },
   {
    "internalType": "address",
    "name": "_newOwner",
    "type": "address"
   }
  ],
  "name": "transferOwnership",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
 }
]

if (!window.ethereum) {
 alert('Meta Mask Not Found')
 window.open("https://metamask.io/download/")
}

export const provider = new ethers.providers.Web3Provider(window.ethereum);
export const signer = provider.getSigner();
export const address = "0xd9145CCE52D386f254917e481eB44e9943F39138"
export const contract = new ethers.Contract(address, abi, signer)
```

Home.js

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import { contract } from "./connector";

function Home() {
   const [Id, setId] = useState("");
   const [Title, setTitle] = useState("");
   const [Author, setAuthor] = useState("");
   const [TranId, setTranId] = useState("");
   const [Owner, setOwner] = useState("");
   const [BookId, setBookId] = useState("");
   const [BookDet, setBookDet] = useState("");
   const [Wallet, setWallet] = useState("");




   const handleId = (e) => {
```

```
  setId(e.target.value)
}

const handleTitle = (e) => {
  setTitle(e.target.value)
}

const handleAuthor = (e) => {
  setAuthor(e.target.value)
}

const handleAddBook = async () => {
  try {
    let tx = await contract.addBook(Id.toString(), Title, Author)
    let wait = await tx.wait()
    alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}

const handleTid = (e) => {
  setTranId(e.target.value)
}

const handleNewOwner = (e) => {
  setOwner(e.target.value)
}

const handleTransfer = async () => {
  try {
    let tx = await contract.transferOwnership(TranId.toString(), Owner)
    let wait = await tx.wait()
    console.log(wait);
    alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}

const handleBookId = (e) => {
  setBookId(e.target.value)
}

const handleBookDetails = async () => {
  try {
    let tx = await contract.getBookDetails(BookId.toString())
    let arr = []
    tx.map(e => arr.push(e))

    setBookDet(arr)
  } catch (error) {
    alert(error)
    console.log(error);
  }
}

const handleWallet = async () => {
  if (!window.ethereum) {
    return alert('please install metamask');
  }

  const addr = await window.ethereum.request({
    method: 'eth_requestAccounts',
  });

  setWallet(addr[0])
  }

 return (
  <div>
   <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Book Details On
Blockchain</h1>
      {!Wallet ?

        <Button onClick={handleWallet} style={{ marginTop: "30px", marginBottom:
"50px" }}>Connect Wallet </Button>
        :
        <p style={{ width: "250px", height: "50px", margin: "auto", marginBottom:
"50px", border: '2px solid #2096f3' }}>{Wallet.slice(0, 6)}....{Wallet.slice(-6)}</p>
      }
   <Container>
    <Row>
     <Col style={{marginRight:"100px"}}>
      <div>



      <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleId} type="number" placeholder="Enter Registration number"
value={Id} /> <br />

      <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleTitle} type="string" placeholder="Enter Book Title" value={Title}
/> <br />

                <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleAuthor} type="string" placeholder="Enter Book Author"
value={Author} /><br />

      <Button onClick={handleAddBook} style={{ marginTop: "10px" }}
variant="primary">Add book</Button>
```

```
      </div>
     </Col>
      <Col>
        <div>
                  <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleTid} type="number" placeholder="Enter Registration Id"
value={TranId} /> <br />

                  <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleNewOwner} type="string" placeholder="Enter New owner
metamask address" value={Owner} /><br />

            <Button onClick={handleTransfer} style={{ marginTop: "10px" }}
variant="primary">Transfer book ownership</Button>

        </div>
      </Col>
   </Row>
   <Row style={{marginTop:"100px"}}>
     <Col >
       <div style={{ margin:"auto" }}>
         <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleBookId} type="number" placeholder="Enter book Id"
value={BookId} /><br />

       <Button onClick={handleBookDetails} style={{ marginTop: "10px" }}
variant="primary">Get Identity Details</Button>
      {BookDet ? BookDet?.map(e => {
          return <p>{e.toString()}</p>
       }) : <p></p>}
      </div>
     </Col>
   </Row>
   </Container>

  </div>
 )
}

export default Home;
```

App.js

```
import './App.css';
import Home from './Page/Home'

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <Home />
      </header>
    </div>
  );
}

export default App;
```

App.css

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Index.css

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

GITHUB LINK:

https://github.com/kabileshkumar/Blockchain-Powered-Library-Management

DEMO VIDEO LINK:

https://drive.google.com/file/d/1x_baNL4kSQBNniaI9VIAceQpgrILPb0Y/view?usp=drivesdk