**DECENTRALIZED BUYER-SELLER MARKET FOR BOOKS**

A Hackathon Project Report Submitted   
In fulfillment of the requirements for the

## Virtusa Codelite Hackathon

## by

## Manasa Raghavaraju CMR Group of Institutions

## Manish Reddy CMR Group of Institutions

## Sai Manikanta CMR Group of Institutions

## Sameer Kasivajhula CMR Group of Institutions

**DECLARATION**

## We hereby declare that the project titled “Decentralized Buyer-Seller Market for Books” submitted to Virtusa for the Virtusa Codelite Hackathon is a result of original research carried-out in this thesis. It is further declared that the project report or any part thereof has not been previously submitted to any University or Institute or Organization.

**TEAM**

## Manasa Raghavaraju (manasaraghavraju) Backend Developer

## Manish Reddy (nmreddy1911) Frontend Developer

## Sai Manikanta (SaiManikanta1918) Backend Developer

## Sameer Kasivajhula (sameer-kousik) Full Stack Developer

Github Link for Project Repository:

##### <https://github.com/iamneowise/Blockchain-Buyer-Seller-Market-BC_Team_6>

**ABSTRACT**

The Project entitled ‘Decentralized Buyer-Seller Market for Books’ is a Website model for an e-commerce application which allows users to participate in buying or selling of books. This model helps the users to add books they own and sell them for ether. User can buy books posted online and resell after completing them. We implement this model on Blockchain to achieve decentralization and use Smart Contracts to make the transaction secure and efficient.

We use Solidity as Programming language for developing the Smart Contracts for the project. React (JavaScript Library) is used for developing the web pages. Truffle and Ganache-cli are used in development process of the backend. This model runs on Ethereum Network.

Software Requirements are:

* Any web browser with the latest version.
* Operating Systems can be Windows XP and above.
* NodeJS
* Truffle (Development Environment and Testing Framework)
* Ganache-cli (Ethereum Development Tools)
* React (JavaScript Library)
* MetaMask Extension in the Browser

Hardware Requirements are:

* Ram : 2GB Ram and above
* Hard Disk : 50GB and above
* Processor : Dual core and above

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
|  | **CONTENTS**  **1. INTRODUCTION TO PROJECT**  1.1 PURPOSE OF THE PROJECT  1.2 SCOPE OF PROJECT  1.3 FEATURES OF THE PROJECT | **PAGE NO**  **01**  01  01  01 |
|  | **2. SYSTEM ANALYSIS AND DESCRIPTION**  2.1 EXISTING SYSTEM  2.2 PROPOSED SYSTEM  2.3 MODULES  2.4 FEASIBILITY STUDY  2.5 SDLC | **02-06**  02  02  02  03  04-06 |
|  | **3. SYSTEM DESIGN**  3.1 SOFTWARE DESIGN  3.2 INPUT/ OUTPUT DESIGN  3.3 HARDWARE AND SOFTWARE REQUIREMENTS  **4**. **IMPLEMENTATION**  4.1 BLOCKCHAIN  4.2 ETHEREUM  4.3 SMART CONTRACTS  4.4 REACT  4.5 SOLIDITY FUNCTIONS  4.5 RUNNING BACKEND | **07-08**  07  07  08  **09-14**  09-11  11-13  13  13-14  14-16  16-23 |
|  | **5. TESTING**  5.1 SOFTWARE TESTING  5.2 TEST CASE DESIGN  5.3 TEST CASES  **6. SCREEN SHOTS**  **7. CONCLUSION**  **8. BIBLIOGRAPHY** | **23-25**  23  23-24  24-25  **26-29**  **30**  **30** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE.NO** | **NAME** | **PAGE.NO** |
| FIG 2.5.2 | SDLC | 05 |

**LIST OF SCREENSHOTS**

|  |  |  |
| --- | --- | --- |
| **FIGURE.NO** | **TITLE** | **PAGE.NO** |
| FIG 6.1 | Home Page | 26 |
| FIG 6.2 | Books for Sale | 26 |
| FIG 6.3 | Search Books | 27 |
| FIG 6.4 | Buy Book | 27 |
| FIG 6.5 | Order Summary | 28 |
| FIG 6.6 | Owned Books | 28 |
| FIG 6.7 | Add Book | 29 |
| FIG 6.8 | Seller Dashboard | 29 |

**Chapter 1: INTRODUCTION**

This web application is an e-commerce model in which adding, selling, buying and reselling of books can be performed. This application provides a platform to buy, sell and resell books by users on Blockchain to achieve decentralization and perform secure and efficient transactions.

**1.1 PURPOSE OF THE PROJECT**

To develop a decentralized platform using Blockchain technology to achieve secure and efficient transactions for users to add, buy, sell and resell books at their convenience.

**1.2 SCOPE OF THE PROJECT**

In the proposed system, the users can buy books using Ether (Cryptocurrency) on the Ethereum Blockchain Network. The proposed model is decentralized with Smart Contract implementation.

**1.3 FEATURES OF THE PROJECT**

1. The model uses Blockchain technology to achieve decentralization.
2. Ethereum Blockchain Network is used and Ether is used to buy books.
3. The transactions are secure and efficient. Smart Contracts are developed to implement the transactions efficiently.

**Chapter 2: SYSTEM ANALYSIS AND DESCRIPTION**

**2.1 EXISTING SYSTEM**

Almost all the dedicated bookstores online are centralized and do not accept cryptocurrency as a mode of payment.

**2.2 PROPOSED SYSTEM**

In the proposed system, Blockchain Technology is used to achieve decentralization. This system runs in Ethereum Blockchain Network thus accepts Ether (Ethereum Native Cryptocurrency) as the mode of payment.

**2.2.1 Advantages**

1. The model uses Blockchain technology to achieve decentralization.
2. Ethereum Blockchain Network is used and Ether is used to buy books.
3. The transactions are secure and efficient. Smart Contracts are developed to implement the transactions efficiently.

# 2.3 MODULES

The activities in the model are divided into modules as mentioned below:

1. User interface
2. Add Books Functionality
3. Sell Books Functionality
4. Buy Books Functionality
5. Re-sell Functionality
6. Integration

**2.4 FEASIBILITY STUDY:**

The next step in analysis is to verify the feasibility of the proposed system. “All projects are feasible given unlimited resources and infinite time “. But in reality, both resources and time are scarce. Projects should confirm to time bounce and should be optimal in their consumption of resources. This place is a constant approval of any project.

There are 3 types of feasibilities:

* Technical feasibility
* Operational feasibility
* Economic feasibility

## 2.4.1 Technical Feasibility:

To determine whether the proposed system is technically feasible, we should take into consideration the technical issues involved behind the system.

This Application uses the Blockchain technology which is decentralized and provide one of the most secure mode for transactions. This application also uses web technologies, which is rampantly employed these days worldwide. The world without the web is incomprehensible today. That proposed system is technically feasible.

**2.4.2 Operational Feasibility:**

To determine the operational feasibility of the system we should take into consideration the awareness level of the users. This system is operational feasible since the users are familiar with the technologies and hence there is no need to gear up the personnel to use the system. Also, the system is very friendly and to use.

## 2.4.3 Economic Feasibility:

To decide whether a project is economically feasible, we have to consider various factors as:

* + - * Cost benefit analysis
      * Long-term returns
      * Maintenance costs

If the proposed model runs on a public blockchain the basic capabilities would suffice as the application in decentralized. It requires average computing capabilities and access to the internet, which are very basic requirements and can be afforded by any organization hence it doesn’t incur additional economic overheads, which renders the system economically feasible.

**2.5 SDLC (SOFTWARE DEVELOPMENT LIFE CYCLE)**

**2.5.1 SDLC Methodologies**

This document plays a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It is meant for use by developers and will be the basics during the testing phase. Any changes made to the requirements in the future will have to go through a formal change approval process.

**2.5.2 SPIRAL MODEL:** It was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development. The spiral model is similar to the [incremental model](http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/), with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spirals, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral.

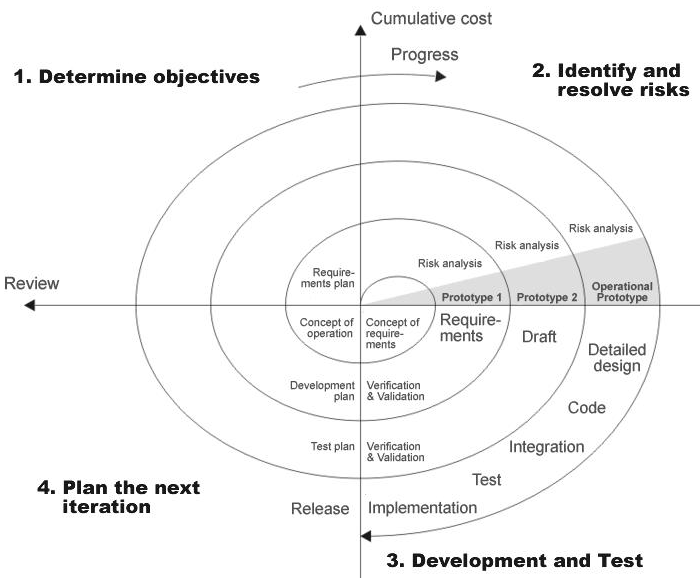
As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

* The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
* A preliminary design is created for the new system.
* A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
* A second prototype is evolved by a fourfold procedure:

1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
2. Defining the requirements of the second prototype.
3. Planning to design the second prototype.
4. Constructing and testing the second prototype.

* At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer’s judgment, result in a less-than-satisfactory final product.
* The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
* The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
* The final system is constructed, based on the refined prototype.
* The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.
* **The following diagram shows how a spiral model acts like:**



**Fig.2.5 Spiral Model**

* **Planning Phase:**Requirements are gathered during the planning phase. Requirements like ‘BRS’ that is ‘Business Requirement Specifications’ and ‘SRS’ that is ‘System Requirement specifications’.
* **Risk Analysis:** In the**risk analysis phase**, a [process](http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/) is undertaken to identify risk and alternate solutions.  A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.
* **Engineering Phase:** In this phase software is **developed**, along with [testing](http://istqbexamcertification.com/what-is-a-software-testing/) at the end of the phase. Hence in this phase the development and testing is done.
* **Evaluation phase:**This phase allows the customer to evaluate the output of the project to [date](http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/) before the project continues to the next spiral

## Chapter 3: SYSTEM DESIGN

System design is transition from a user-oriented document to programmers or data base personnel. The design is a solution, how to approach the creation of a new system. This is composed of several steps. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Designing goes through logical and physical stages of development, logical design reviews the present physical system, prepares input and output specification, details of implementation plan and prepares a logical design walkthrough.

**3.1 SOFTWARE DESIGN**

In designing the software following principles are followed:

* **Modularity and partitioning**: software is designed such that each system should consist of a hierarchy of modules and serve to partition into a separate function.
* **Coupling:** modules should have little dependence on other modules of a system.
* **Cohesion:** modules should carry out in a single processing function.
* **Shared use:** avoid duplication by allowing a single module to be called by others that need the function it provides.

**3.2 INPUT/OUTPUT DESIGN**

**3.2.1 Input Design:**

The input design has been done keeping in view that, the interaction of the user with the system being the most effective and simplified way. The users can easily upload the details of the book they own if they intend to sell it. The user can easily browse the available books and buy by providing the necessary details.

**3.2.2 Output Design:**

All the webpages of the system are designed with a view to provide the user a simpler and efficient way. Important information is emphasized on the screen. The output screen displays several webpages for adding, selling, buying or display owned books and mark them for reselling.

**3.3 HARDWARE AND SOFTWARE REQUIREMENTS:**

## 3.3.1 Software Interfaces:

We used Solidity as Programming language for developing the Smart Contracts for the project. React (JavaScript Library) is used for developing the web pages. Truffle and Ganache-cli are used in development process of the backend. This model runs on Ethereum Network.

Software Requirements are:

* Any web browser with the latest version.
* Operating Systems can be Windows XP and above.
* NodeJS
* Truffle (Development Environment and Testing Framework)
* Ganache-cli (Ethereum Development Tools)
* React (JavaScript Library)
* MetaMask Extension in the Browser.

**3.3.2 Hardware Requirements**

* Ram : 2GB Ram and above
* Hard Disk : 50GB and above
* Processor : Dual core and above

**Chapter 4: IMPLEMENTATION**

**4.1 BLOCKCHAIN**

**4.1.1 Blockchain:**

The first blockchain was conceptualized by a person (or group of people) known as Satoshi Nakamoto in 2008. Nakamoto improved the design in an important way using a Hashcash-like method to timestamp blocks without requiring them to be signed by a trusted party and introducing a difficulty parameter to stabilize rate with which blocks are added to the chain. The design was implemented the following year by Nakamoto as a core component of the cryptocurrency bitcoin, where it serves as the public ledger for all transactions on the network.

A blockchain is a decentralized, distributed, and oftentimes public, digital ledger consisting of records called blocks that is used to record transactions across many computers so that any involved block cannot be altered retroactively, without the alteration of all subsequent blocks. This allows the participants to verify and audit transactions independently and relatively inexpensively. A blockchain database is managed autonomously using a peer-to-peer network and a distributed timestamping server. They are authenticated by mass collaboration powered by collective self-interests. Such a design facilitates robust workflow where participants' uncertainty regarding data security is marginal. The use of a blockchain removes the characteristic of infinite reproducibility from a digital asset. It confirms that each unit of value was transferred only once, solving the long-standing problem of double spending. A blockchain has been described as a value-exchange protocol. A blockchain can maintain title rights because, when properly set up to detail the exchange agreement, it provides a record that compels offer and acceptance.

Logically, a blockchain can be seen as consisting of several layers:

* Infrastructure (hardware)
* Networking (node discovery, information propagation and verification)
* Consensus (proof of work, proof of stake)
* Data (blocks, transactions)
* Application (smart contracts/dapps, if applicable)

**4.1.2 Decentralization**

By storing data across its peer-to-peer network, the blockchain eliminates a number of risks that come with data being held centrally. The decentralized blockchain may use ad hoc message passing and distributed networking.

Peer-to-peer blockchain networks lack centralized points of vulnerability that computer crackers can exploit; likewise, it has no central point of failure. Blockchain security methods include the use of public-key cryptography. A public key (a long, random-looking string of numbers) is an address on the blockchain. Value tokens sent across the network are recorded as belonging to that address. A private key is like a password that gives its owner access to their digital assets or the means to otherwise interact with the various capabilities that blockchains now support. Data stored on the blockchain is generally considered incorruptible.

Every node in a decentralized system has a copy of the blockchain. Data quality is maintained by massive database replication and computational trust. No centralized "official" copy exists and no user is "trusted" more than any other. Transactions are broadcast to the network using software. Messages are delivered on a best-effort basis. Mining nodes validate transactions, add them to the block they are building, and then broadcast the completed block to other nodes. Blockchains use various time-stamping schemes, such as proof-of-work, to serialize changes. Alternative consensus methods include proof-of-stake. Growth of a decentralized blockchain is accompanied by the risk of centralization because the computer resources required to process larger amounts of data become more expensive.

**4.1.3 Types**

Currently, there are at least four types of blockchain networks — public blockchains, private blockchains, consortium blockchains and hybrid blockchains.

* **Public blockchains**: A public blockchain has absolutely no access restrictions. Anyone with an Internet connection can send transactions to it as well as become a validator (i.e., participate in the execution of a consensus protocol). Usually, such networks offer economic incentives for those who secure them and utilize some type of a Proof of Stake or Proof of Work algorithm. Some of the largest, most known public blockchains are the bitcoin blockchain and the Ethereum blockchain.
* **Private blockchains**: A private blockchain is permissioned. One cannot join it unless invited by the network administrators. Participant and validator access is restricted. To distinguish between open blockchains and other peer-to-peer decentralized database applications that are not open ad-hoc compute clusters, the terminology Distributed Ledger (DLT) is normally used for private blockchains.
* **Hybrid blockchains**: A hybrid blockchain has a combination of centralized and decentralized features. The exact workings of the chain can vary based on which portions of centralization decentralization are used.
* **Sidechains**: A sidechain is a designation for a blockchain ledger that runs in parallel to a primary blockchain. Entries from the primary blockchain (where said entries typically represent digital assets) can be linked to and from the sidechain; this allows the sidechain to otherwise operate independently of the primary blockchain (e.g., by using an alternate means of record keeping, alternate consensus algorithm, etc.).

**4.1.4 Applications**

Blockchain technology can be integrated into multiple areas. The primary use of blockchains today is as a distributed ledger for cryptocurrencies, most notably bitcoin. There are a few operational products maturing from proof of concept by late 2016. Businesses have been thus far reluctant to place blockchain at the core of the business structure. Although businesses have been reluctant to fully implement blockchain, many have begun testing the technology and are conducting low-level implementation to gauge its effects on organizational efficiency.

In 2019, it was estimated that around $2.9 billion were invested in blockchain technology, which represents an 89% increase from the year prior. Additionally, the International Data Corp has estimated that corporate investment into blockchain technology will reach $12.4 billion by 2022. Furthermore, according to PricewaterhouseCoopers (PwC), the second-largest professional services network in the world, blockchain technology has the potential to generate an annual business value of more than $3 trillion by 2030. PwC's estimate is further augmented by a 2018 study that they have conducted, in which PwC surveyed 600 business executives and determined that 84% have at least some exposure to utilizing blockchain technology, which indicts a significant demand and interest in blockchain technology.

Individual use of blockchain technology has also greatly increased since 2016. According to statistics in 2020, there were more than 40 million blockchain wallets in 2020 in comparison to around 10 million blockchain wallets in 2016.

**4.2 ETHEREUM**

Ethereum was initially described in a white paper by Vitalik Buterin, a programmer and co-founder of Bitcoin Magazine, in late 2013 with a goal of building decentralized applications. Buterin argued that Bitcoin and blockchain technology could benefit from other applications besides money and needed a scripting language for application development that could lead to attaching real-world assets, such as stocks and property, to the blockchain. In 2013, Buterin briefly worked with eToro CEO Yoni Assia on the Colored Coins project and drafted its white paper outlining additional use cases for blockchain technology. However, after failing to gain agreement on how the project should proceed, he proposed the development of a new platform with a more general scripting language that would eventually become Ethereum.

**4.2.1 Design**

Ethereum is a permissionless, non-hierarchical network of computers (nodes) which build and come to consensus on an ever-growing series of "blocks", or batches of transactions, known as the [blockchain](https://en.wikipedia.org/wiki/Blockchain). Each block contains an identifier of the block that it must immediately follow in the chain if it is to be considered valid. Whenever a node adds a block to its chain, it executes the transactions therein in their order, thereby altering the ETH balances and other storage values of Ethereum accounts. These balances and values, collectively known as the state, are maintained on the node's computer separately from the blockchain, in a Merkle tree.

Each node communicates with a relatively small subset of the network, known as its peers. Whenever a node wishes to include a new transaction in the blockchain, it sends the transaction to its peers, who then send it to their peers, and so on. In this way, it propagates throughout the network. Certain nodes, called miners, maintain a list of all of these new transactions and use them to create new blocks, which they then send to the rest of the network. Whenever a node receives a block, it checks the validity of the block and of all of the transactions therein and, if valid, adds it to its blockchain and executes all of said transactions. As the network is non-hierarchical, a node may receive competing blocks, which may form competing chains. The network comes to consensus on the blockchain by following the "longest-chain rule", which states that the chain with the most blocks at any given time is the canonical chain. This rule achieves consensus because miners do not want to expend their computational work trying to add blocks to a chain that will be abandoned by the network.

**4.2.1.1 Ether**

Ether (ETH) is the cryptocurrency generated by the Ethereum protocol as a reward to miners in a proof-of-work system for adding blocks to the blockchain. It is the only currency accepted in the payment of transaction fees, which also go to miners. The block reward together with the transaction fees provide the incentive to miners to keep the blockchain growing (i.e. to keep processing new transactions). Therefore, ETH is fundamental to the operation of the network. Each Ethereum account has an ETH balance and may send ETH to any other account. The smallest subunit of ETH is known as a Wei and is equal to 10-18 ETH. Ether is often erroneously referred to as "Ethereum".

Ether is listed on exchanges under the ticker symbol ETH. The Greek uppercase Xi character (Ξ) is sometimes used for its currency symbol.

The shift to Ethereum 2.0 may reduce the issuance rate of Ether. There is currently no implemented hard cap on the total supply of Ether.

**4.2.1.2 Gas**

Gas is a unit of account within the EVM used in the calculation of a transaction fee, which is the amount of ETH a transaction's sender must pay to the miner who includes the transaction in the blockchain.

Each type of operation which may be performed by the EVM is hardcoded with a certain gas cost, which is intended to be roughly proportional to the amount of resources (computation and storage) a node must expend to perform that operation. When creating a transaction, the sender must specify a gas limit and gas price. The gas limit is the maximum amount of gas the sender is willing to use in the transaction, and the gas price is the amount of ETH the sender wishes to pay to the miner per unit of gas used. The higher the gas price, the more incentive a miner has to include the transaction in their block, and thus the quicker the transaction will be included in the blockchain. The sender buys the full amount of gas (i.e. the gas limit) up-front, at the start of the execution of the transaction, and is refunded at the end for any gas not used. If at any point the transaction does not have enough gas to perform the next operation, the transaction is reverted but the sender still pays for the gas used. Gas prices are typically denominated in Gwei, a subunit of ETH equal to 10-9 ETH.

This fee mechanism is designed to mitigate transaction spam, prevent infinite loops during contract execution, and provide for a market-based allocation of network resources.

**4.3 SMART CONTRACTS**

A smart contract is a computer program or a transaction protocol which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement. The objectives of smart contracts are the reduction of need in trusted intermediators, arbitrations and enforcement costs, fraud losses, as well as the reduction of malicious and accidental exceptions.

Vending machines are mentioned as the oldest piece of technology equivalent to smart contract implementation. 2014's white paper about the cryptocurrency Ethereum describes the Bitcoin protocol as a weak version of the smart contract concept as defined by computer scientist, lawyer and cryptographer Nick Szabo. Since Ethereum, various cryptocurrencies support scripting languages which allow for more advanced smart contracts between untrusted parties. Smart contracts should be distinguished from smart legal contracts. The latter refers to a traditional natural language legally-binding agreement which has certain terms expressed and implemented in machine-readable code.

Smart contracts were first proposed in the early 1990s by Nick Szabo, who coined the term, using it to refer to "a set of promises, specified in digital form, including protocols within which the parties perform on these promises". In 1998, the term was used to describe objects in rights management service layer of the system The Stanford Infobus, which was a part of Stanford Digital Library Project.

**4.4 REACT**

React (also known as React.js or ReactJS) is an open-source, front end, JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

React was created by Jordan Walke, a software engineer at Facebook, who released an early prototype of React called "FaxJS". He was influenced by XHP, an HTML component library for PHP. It was first deployed on Facebook's News Feed in 2011 and later on Instagram in 2012. It was open-sourced at JSConf US in May 2013.

React Native, which enables native Android, iOS, and UWP development with React, was announced at Facebook's React Conf in February 2015 and open-sourced in March 2015.

* On April 18, 2017, Facebook announced React Fiber, a new core algorithm of React library for building user interfaces.[36] React Fiber was to become the foundation of any future improvements and feature development of the React library.
* On September 26, 2017, React 16.0 was released to the public.
* On February 16, 2019, React 16.8 was released to the public. The release introduced React Hooks.
* On August 10, 2020, the React team announced the first release candidate for React v17.0, notable as the first major release without major changes to the React developer-facing API.

**4.5 Solidity Functions**

* **addBook():** Function used to add book to the block.
  + Input:
    - title, string
    - isbn, string
    - author, string
    - category, string
    - price, unit256
    - forSale, bool
    - image, string
  + Output: None
* **sellBook():** Function used to mark book for sale.
  + Input:
    - id, bytes32
  + Output: None
* **unSellBook():** Function used to unmark book for sale.
  + Input:
    - id, bytes32
  + Output: None
* **buyBook():** Function used by the user to buy book.
  + Input:
    - id, bytes32
    - name, string
    - deliveryAddress, string
    - postalCode, unit256
    - phone, unit256
  + Output: None
* **markOrderCompleted():** Function used by the seller to mark the book as delivered.
  + Input:
    - id, bytes32
  + Output: None
* **getBooksIds():** Function used to retrieve the available Book IDs
  + Input:
    - limit, int256
  + Output:
    - List of Book IDs, bytes32[]
* **getBook():** Function used to retrieve the details of a particular book\
  + Input:
    - id, bytes32
  + Output:
    - bytes32 id, bytes32
    - title, string
    - isbn, string
    - author, string
    - category, string
    - owner, address
    - price, unit256
    - forSale, bool
    - sold, boo;
    - image, string
* **getBooksForSale():** Function returns the IDs of the books that are available for sale.
  + Input:
    - limit, unit256
  + Output:
    - List of Book IDs, bytes32[]
* **getOrdersLists():** Function returns the list of order IDs.
  + Input:
    - None
  + Output:
    - List of Order IDs, bytes32[]
* **getOrderById():** Function returns the details of the selected order.
  + Input:
    - id, bytes32
  + Output:
    - bytes32 orderId, bytes32
    - name, string
    - deliveryAddress, string
    - postalCode, unit256
    - phone, unit256
    - state, string
    - customer, address
    - bookName, string
    - seller, address

**4.5 RUNNING BACKEND LOCALLY**

## *Steps to Run the Backend locally*

### Install Truffle and Ganache-Cli on the machine

(Assuming that Node is already installed)

npm install -g truffle   
npm install -g ganache-cli

### Cloning Latest Updates

git clone https://github.com/iamneowise/Blockchain-Buyer-Seller-Market-BC\_Team\_6.git && cd Blockchain-Buyer-Seller-Market-BC\_Team\_6

### Deploying in your Machine

Start Ganache CLI

ganache-cli

Start a new terminal in the same directory and go into Truffle Console by

truffle console

Use the migrate command to run migrations(if not done before)

truffle(development)> migrate

#### Output

truffle(development)> migrate  
  
Compiling your contracts...  
===========================  
> Everything is up to date, there is nothing to compile.  
  
  
  
Starting migrations...  
======================  
> Network name: 'develop'  
> Network id: 5777  
> Block gas limit: 6721975 (0x6691b7)  
  
  
2\_deploy\_contracts.js  
=====================  
  
 Deploying 'BookToken'

transaction hash: 0x2701a840946a812cca8f10b02b1ea7750fee5b11cf2600d7d78e3c3819570969  
Blocks: 0 Seconds: 0  
contract address: 0xC886d8cA384E2B4703De5d2ED2222718d9b00594  
block number: 8  
block timestamp: 1617003799  
account: 0x5016D8Da9317425ac10F4D3e902A4b553f53cCd5  
balance: 99.79787398  
gas used: 1368005 (0x14dfc5)  
gas price: 20 gwei  
value sent: 0 ETH  
total cost: 0.0273601 ETH

Deploying 'BookStore'

transaction hash: 0x81fcc17a2dded6593465f11da9c47f0d86b4204c25b9ca03f04f833439bbbb02  
Blocks: 0 Seconds: 0  
contract address: 0xC3b86441236139d1dba352f23A2BfF9030Fc3d1B  
block number: 9  
block timestamp: 1617003800  
account: 0x5016D8Da9317425ac10F4D3e902A4b553f53cCd5  
balance: 99.69461994  
gas used: 5162702 (0x4ec6ce)  
gas price: 20 gwei  
value sent: 0 ETH  
total cost: 0.10325404 ETH

Is set? true  
Deployed both!

Saving migration to chain.  
Saving artifacts

Total cost: 0.13061414 ETH

Summary

Total deployments: 2  
Final cost: 0.13061414 ETH

* Blocks: 0 Seconds: 0
* Blocks: 0 Seconds: 0
* Saving migration to chain.

### Create a BookStore Contract instance.  
```bash  
truffle(development)> BookStore.deployed().then(function(instance){app=instance})

You can use the following command to verify

truffle(development)> app

### Now that you have the Contract instance, you can call the functions in the contract as follows

e.g. Adding A Book to the Block

truffle(development)> app.addBook("Theory of Everything","ISBN0129","Stephen Hawking","Science",12367,false,"https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FMetaMask&psig=AOvVaw01YKJ1Z0HUnd4AxTrJCq9u&ust=1617105710033000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMCQg6261e8CFQAAAAAdAAAAABAD");

#### Output

{  
 tx: '0x9b687183a31d9543b0a14a63d28935ceb65fd06f3f7300033f985ac67f42f1f8',  
 receipt: {  
 transactionHash: '0x9b687183a31d9543b0a14a63d28935ceb65fd06f3f7300033f985ac67f42f1f8',  
 transactionIndex: 0,  
 blockHash: '0xfe6448628e969a51b773d9466660d2709c216a49a1e8d53bef5aefb189596415',  
 blockNumber: 12,  
 from: '0x5016d8da9317425ac10f4d3e902a4b553f53ccd5',  
 to: '0xc5f6ea947cfa075f876d3960c6f3fc323c3a1ba3',  
 gasUsed: 410699,  
 cumulativeGasUsed: 410699,  
 contractAddress: null,  
 logs: [],  
 status: true,  
 logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',  
 rawLogs: []  
 },  
 logs: []  
}

### Get the IDs of the books

"app.getBooksIds(\_limit)"  
Note: Limit is the count of the IDs that will be returned. If the limit is greater than the available books, all the available books will be returned.

truffle(development)> truffle(development)> app.getBooksIds(1)

#### Output

[  
 '0x10f53c90c690b75262c8f5ed075e8d95cb39fe714c4032a24b79f992dd1e63a7'  
]

#### Storing the first Book ID for future use.

truffle(development)> app.getBooksIds(1).then(result => {book = result[0]});

### Get Details of a Book

truffle(development)> app.getBook(book);

#### Output

Result {  
 '0': '0x10f53c90c690b75262c8f5ed075e8d95cb39fe714c4032a24b79f992dd1e63a7',  
 '1': 'Theory of Everything',  
 '2': 'ISBN0129',  
 '3': 'Stephen Hawking',  
 '4': 'Science',  
 '5': '0x5016D8Da9317425ac10F4D3e902A4b553f53cCd5',  
 '6': BN {  
 negative: 0,  
 words: [ 12367, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 '7': false,  
 '8': false,  
 '9': 'https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FMetaMask&psig=AOvVaw01YKJ1Z0HUnd4AxTrJCq9u&ust=1617105710033000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMCQg6261e8CFQAAAAAdAAAAABAD',  
 id: '0x10f53c90c690b75262c8f5ed075e8d95cb39fe714c4032a24b79f992dd1e63a7',  
 title: 'Theory of Everything',  
 isbn: 'ISBN0129',  
 author: 'Stephen Hawking',  
 category: 'Science',  
 owner: '0x5016D8Da9317425ac10F4D3e902A4b553f53cCd5',  
 price: BN {  
 negative: 0,  
 words: [ 12367, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 forSale: false,  
 sold: false,  
 image: 'https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FMetaMask&psig=AOvVaw01YKJ1Z0HUnd4AxTrJCq9u&ust=1617105710033000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMCQg6261e8CFQAAAAAdAAAAABAD'  
}

### Mark the Book for Sale

truffle(development)> app.sellBook(book);

### Saving Truffle Test Accounts in a variable for future use

truffle(development)> web3.eth.getAccounts().then(function(result){accounts = result});

### Buying a Book

truffle(development)> app.buyBook(book,"Sameer","MyHomeAddress",400291,93211121,{from: accounts[2],value: 12367});

### Accessing Orders List and Saving the latest order for future use

truffle(development)> app.getOrdersLists().then(result=>{order1=result[0]});

### Get Order Details

truffle(development)> app.getOrderById(order1);

#### Output

Result {  
 '0': '0x1547b1c5fc5c7661799cdb13136f0185a9aee8c25bdde984c5aa614dfe56c086',  
 '1': 'Sameer',  
 '2': 'MyHomeAddress',  
 '3': BN {  
 negative: 0,  
 words: [ 400291, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 '4': BN {  
 negative: 0,  
 words: [ 934567, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 '5': 'pending',  
 '6': '0xC59Ae61257Da5cB29654c73958F0ad594885d4D2',  
 '7': 'Theory of Everything',  
 '8': '0x5016D8Da9317425ac10F4D3e902A4b553f53cCd5',  
 orderId: '0x1547b1c5fc5c7661799cdb13136f0185a9aee8c25bdde984c5aa614dfe56c086',  
 name: 'Sameer',  
 deliveryAddress: 'MyHomeAddress',  
 postalCode: BN {  
 negative: 0,  
 words: [ 400291, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 phone: BN {  
 negative: 0,  
 words: [ 934567, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 state: 'pending',  
 customer: '0xC59Ae61257Da5cB29654c73958F0ad594885d4D2',  
 bookName: 'Theory of Everything',  
 seller: '0x5016D8Da9317425ac10F4D3e902A4b553f53cCd5'  
}

### Mark the Order as Completed (Seller Exclusive)

truffle(development)> app.markOrderCompleted(order1);

##### ***Updated Order Info:***

Result {  
 '0': '0x1547b1c5fc5c7661799cdb13136f0185a9aee8c25bdde984c5aa614dfe56c086',  
 '1': 'Sameer',  
 '2': 'MYhOME',  
 '3': BN {  
 negative: 0,  
 words: [ 400291, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 '4': BN {  
 negative: 0,  
 words: [ 934567, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 '5': 'completed',  
 '6': '0xC59Ae61257Da5cB29654c73958F0ad594885d4D2',  
 '7': 'Theory of Everything',  
 '8': '0x5016D8Da9317425ac10F4D3e902A4b553f53cCd5',  
 orderId: '0x1547b1c5fc5c7661799cdb13136f0185a9aee8c25bdde984c5aa614dfe56c086',  
 name: 'Sameer',  
 deliveryAddress: 'MYhOME',  
 postalCode: BN {  
 negative: 0,  
 words: [ 400291, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 phone: BN {  
 negative: 0,  
 words: [ 934567, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 state: 'completed',  
 customer: '0xC59Ae61257Da5cB29654c73958F0ad594885d4D2',  
 bookName: 'Theory of Everything',  
 seller: '0x5016D8Da9317425ac10F4D3e902A4b553f53cCd5'  
}

##### ***Updated Book Info***

Result {  
 '0': '0x10f53c90c690b75262c8f5ed075e8d95cb39fe714c4032a24b79f992dd1e63a7',  
 '1': 'Theory of Everything',  
 '2': 'ISBN0129',  
 '3': 'Stephen Hawking',  
 '4': 'Science',  
 '5': '0xC59Ae61257Da5cB29654c73958F0ad594885d4D2',  
 '6': BN {  
 negative: 0,  
 words: [ 12367, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 '7': false,  
 '8': false,  
 '9': 'https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FMetaMask&psig=AOvVaw01YKJ1Z0HUnd4AxTrJCq9u&ust=1617105710033000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMCQg6261e8CFQAAAAAdAAAAABAD',  
 id: '0x10f53c90c690b75262c8f5ed075e8d95cb39fe714c4032a24b79f992dd1e63a7',  
 title: 'Theory of Everything',  
 isbn: 'ISBN0129',  
 author: 'Stephen Hawking',  
 category: 'Science',  
 owner: '0xC59Ae61257Da5cB29654c73958F0ad594885d4D2',  
 price: BN {  
 negative: 0,  
 words: [ 12367, <1 empty item> ],  
 length: 1,  
 red: null  
 },  
 forSale: false,  
 sold: false,  
 image: 'https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FMetaMask&psig=AOvVaw01YKJ1Z0HUnd4AxTrJCq9u&ust=1617105710033000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMCQg6261e8CFQAAAAAdAAAAABAD'  
}

**Chapter 5: TESTING**

5.1 SOFTWARE TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation.

## 5.1.1 Testing Objectives

* + To ensure that during operation the system will perform as per specification.
  + To make sure that system meets the user requirements during operation
  + To make sure that during the operation, incorrect input, processing and output will be detected
  + To see that when correct inputs are fed to the system the outputs are correct
  + To verify that the controls incorporated in the same system as intended
  + Testing is a process of executing a program with the intent of finding an error
  + A good test case is one that has a high probability of finding an as yet undiscovered error.

The software developed has been tested successfully using the following testing strategies and any errors that are encountered are corrected and again the part of the program or the procedure or function is put to testing until all the errors are removed. A successful test is one that uncovers an as yet undiscovered error.

Note that the result of the system testing will prove that the system is working correctly. It will give confidence to system designer, users of the system, prevent frustration during implementation process etc.,

## 5.2 TEST CASE DESIGN:

## 5.2.1 White box testing

White box testing is a testing case design method that uses the control structure of the procedure design to derive test cases. All independent paths in a module are exercised at least once, all logical decisions are exercised at once, execute all loops at boundaries and within their operational bounds exercise internal data structure to ensure their validity. Here the customer is given three chances to enter a valid choice out of the given menu. After which the control exits the current menu.

## 5.2.2 Black Box Testing

Black Box Testing attempts to find errors in following areas or categories, incorrect or missing functions, interface error, errors in data structures, performance error and initialization and termination error. Here all the input data must match the data type to become a valid entry.

The following are the different tests at various levels:

**5.2.3 Unit Testing:**

Unit testing is essentially for the verification of the code produced during the coding phase and the goal is to test the internal logic of the module/program. In the Generic code project, the unit testing is done during the coding phase of data entry forms whether the functions are working properly or not. In this phase all the drivers are tested whether they are rightly connected or not.

**5.2.4 Integration Testing:**

All the tested modules are combined into sub systems, which are then tested. The goal is to see if the modules are properly integrated, and the emphasis being on the testing interfaces between the modules. In the generic code integration testing is done mainly on table creation module and insertion module.

## 5.2.5 Validation Testing

This testing concentrates on confirming that the software is error-free in all respects. All the specified validations are verified and the software is subjected to hard-core testing. It also aims at determining the degree of deviation that exists in the software designed from the specification; they are listed out and are corrected.

## 5.2.6 System Testing

This testing is a series of different tests whose primary is to fully exercise the computer-based system. This involves:

* Implementing the system in a simulated production environment and testing it.
* Introducing errors and testing for error handling.

**5.3 TEST CASES**

**5.3.1 Test Case 1: Adding A Book, Marking it for Sale**

When the user uploads the details and the image of the book, the book is uploaded into the block and when the user marks it for sale, it is added into list of books to be sold.

**5.3.2 Test Case 2: Buying a book, Retrieving Owned Books, Reselling the Book.**

When the user selects the buy book option, user needs to upload the delivery address and approve the request through Metamask. When the transaction is completed successfully, the book is added into the list of books owned by the user and it is unmarked for sale. The books owned by the user are displayed in a separate webpage. Any of the owned books can be marked for sale again.

**Chapter 6: SCREEN SHOTS**

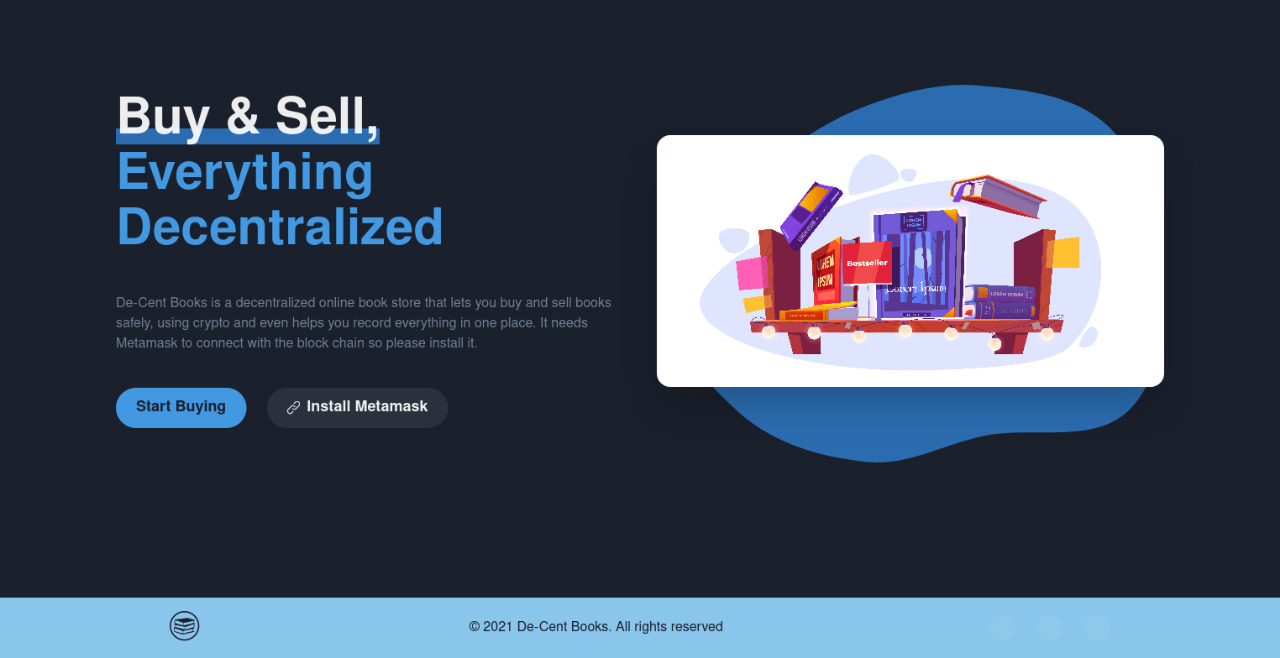
****

Fig.6.1 Home page.

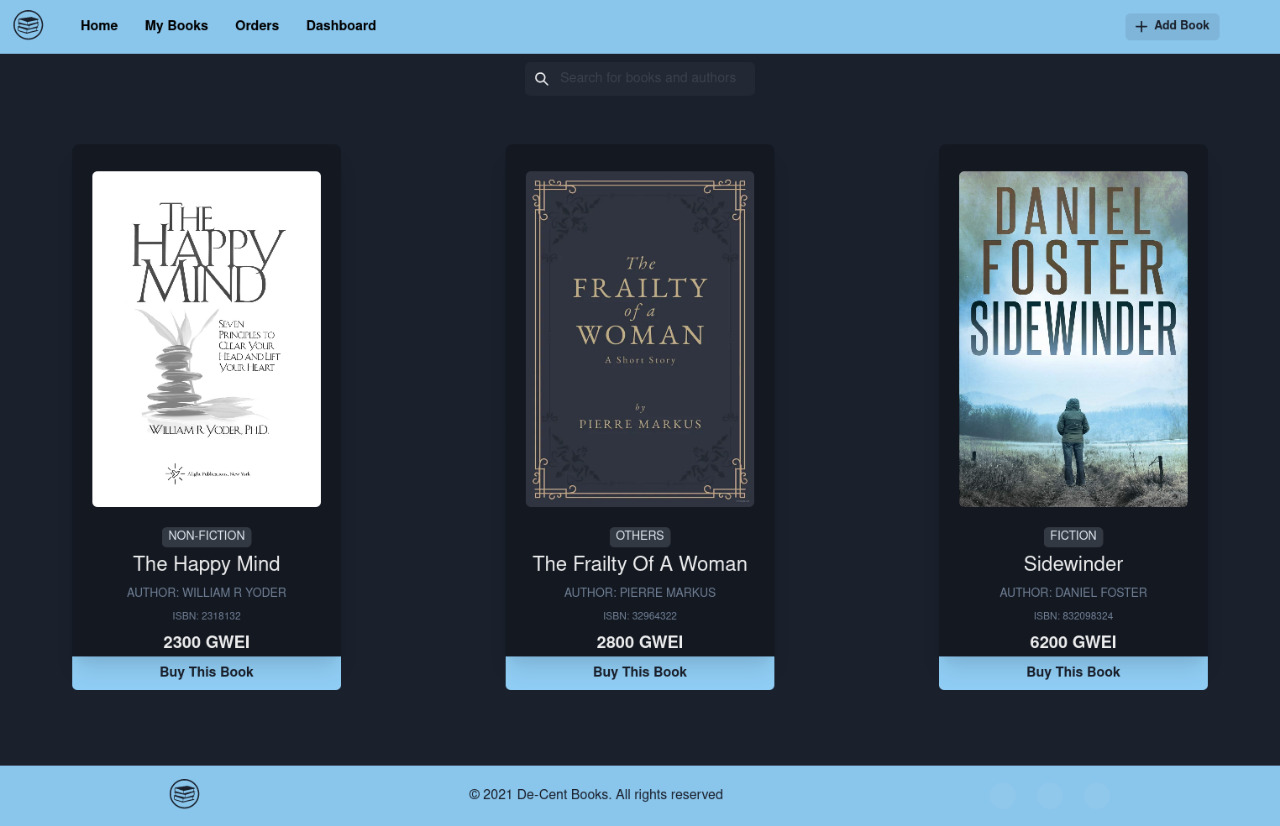
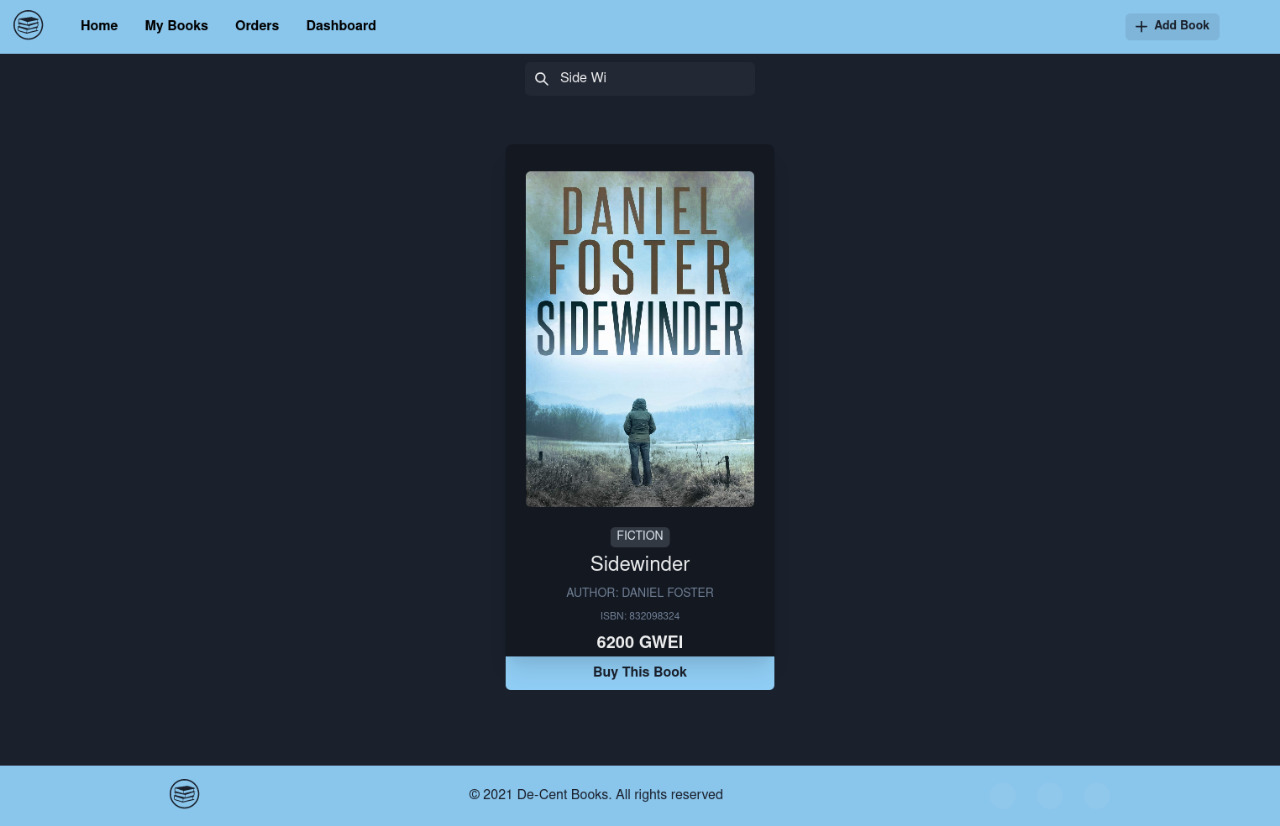


Fig.6.2 Books for Sale

 Fig.6.3 Search Books.

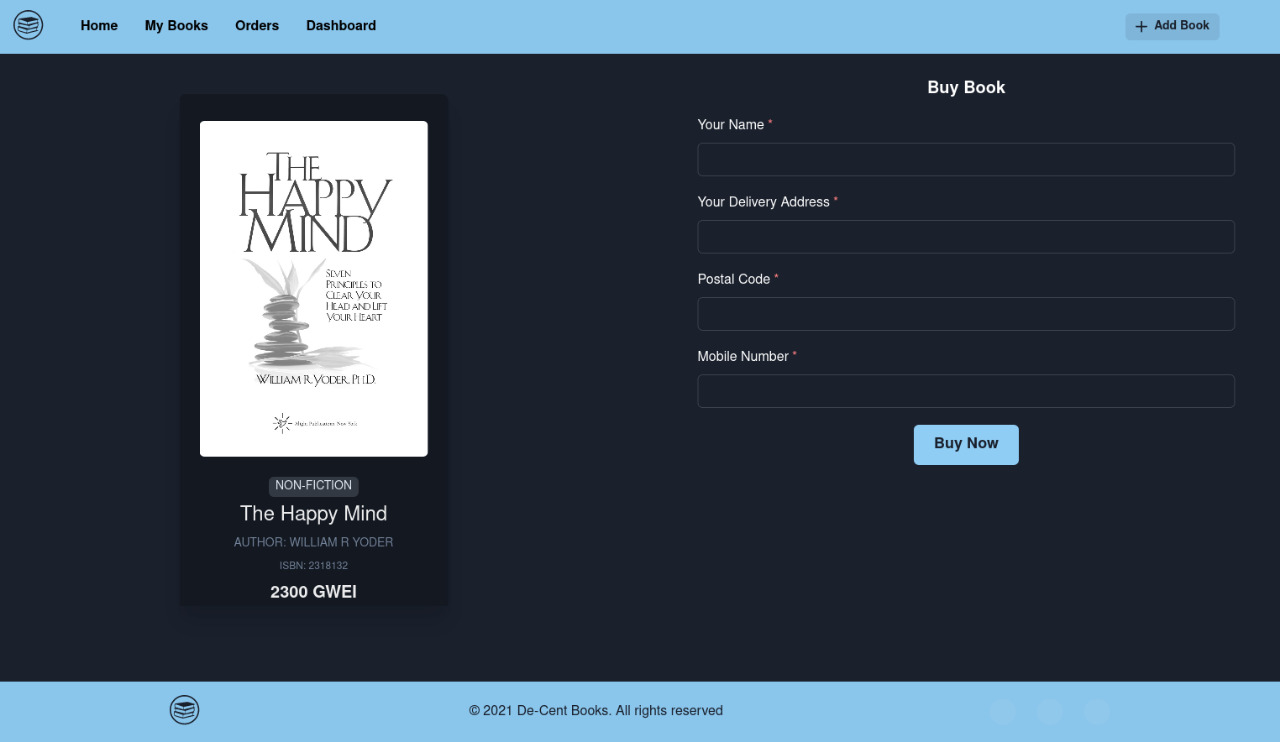


Fig.6.4 Buy Book.

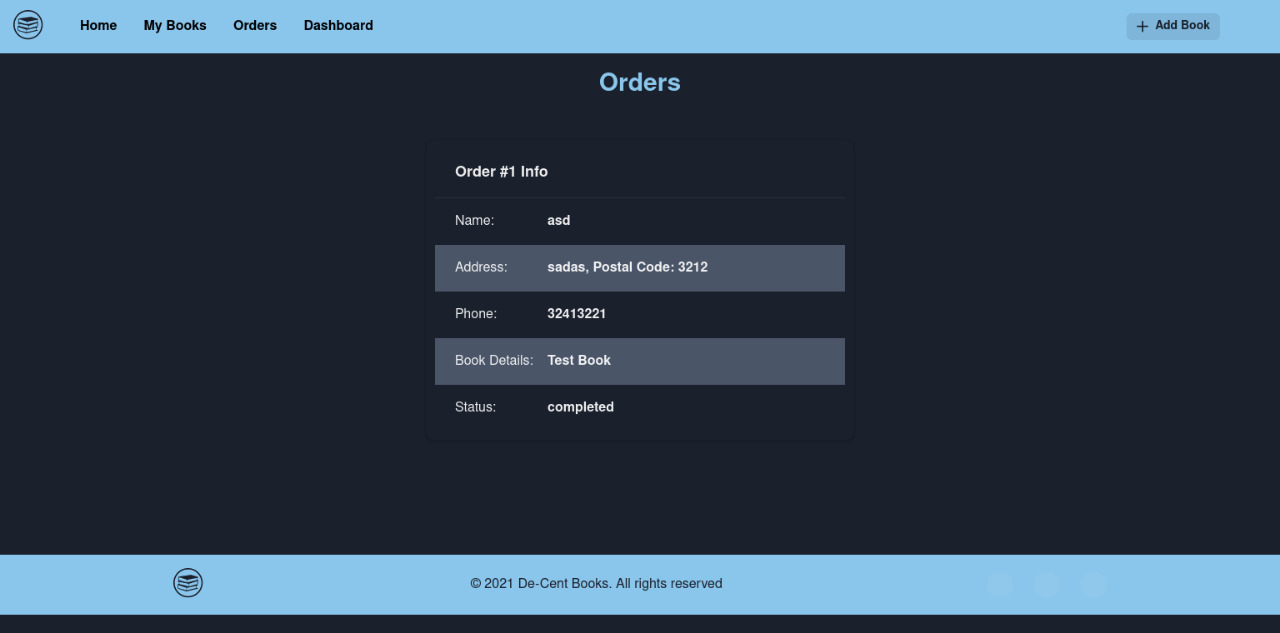


Fig.6.5 Order Summary.

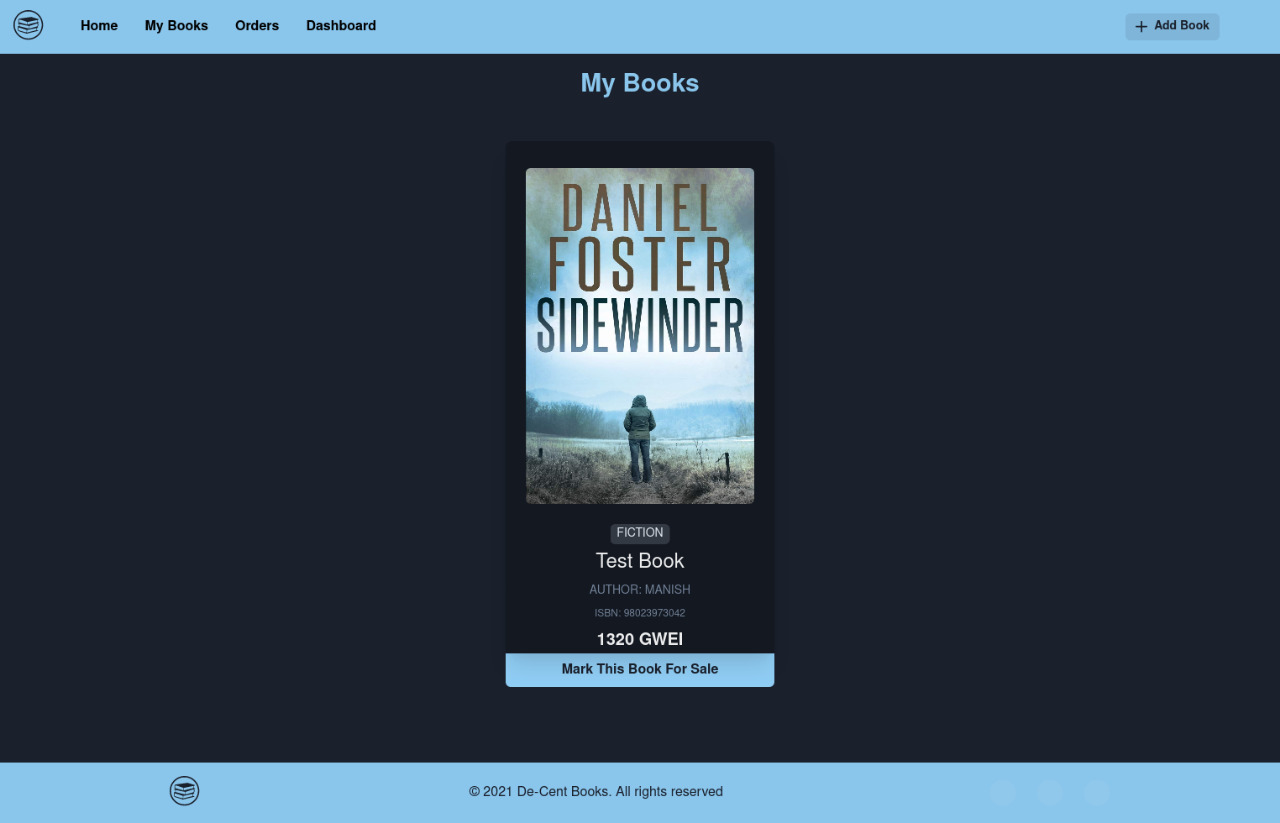


Fig.6.6 Owned Books.

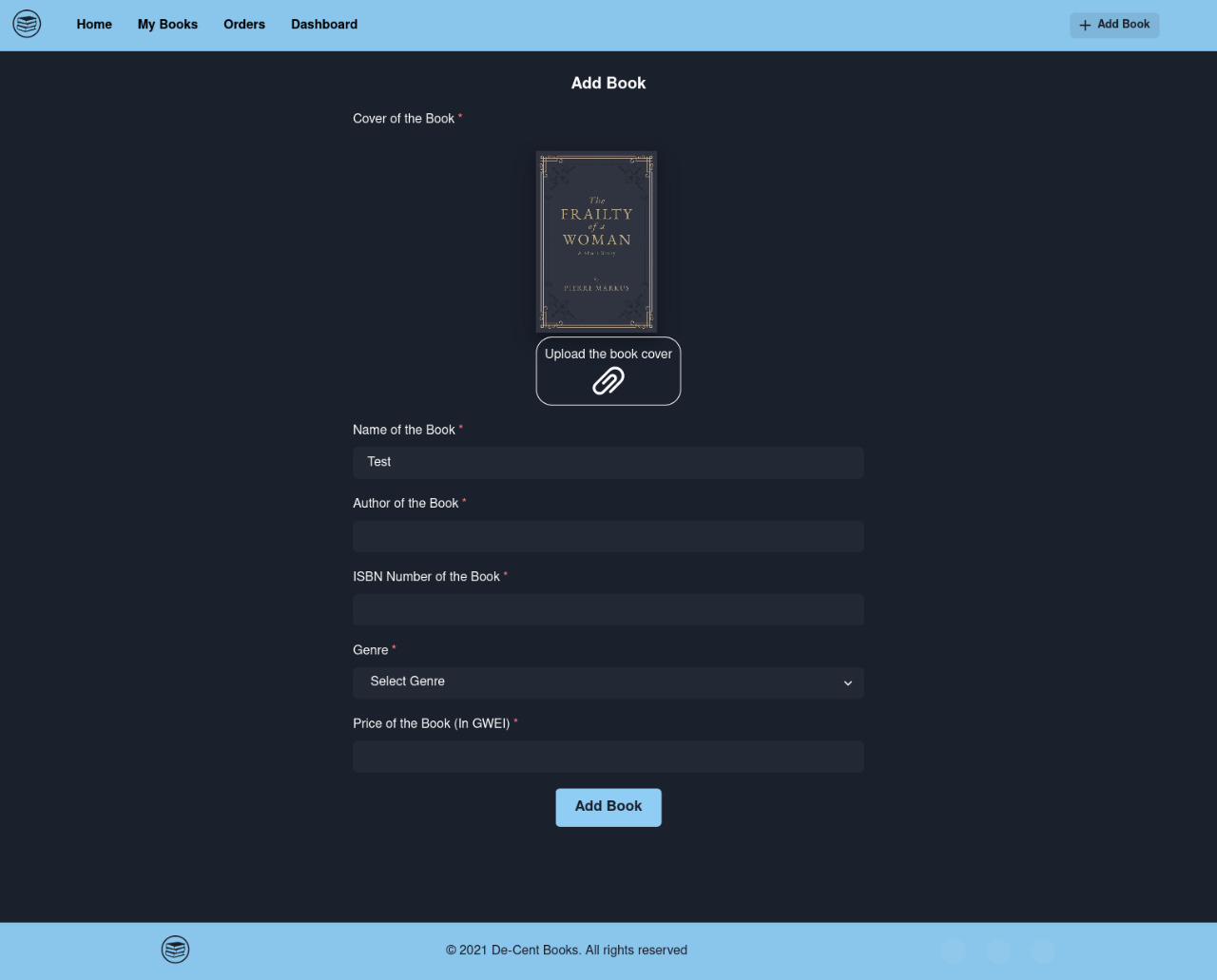


Fig.6.7 Add Book.

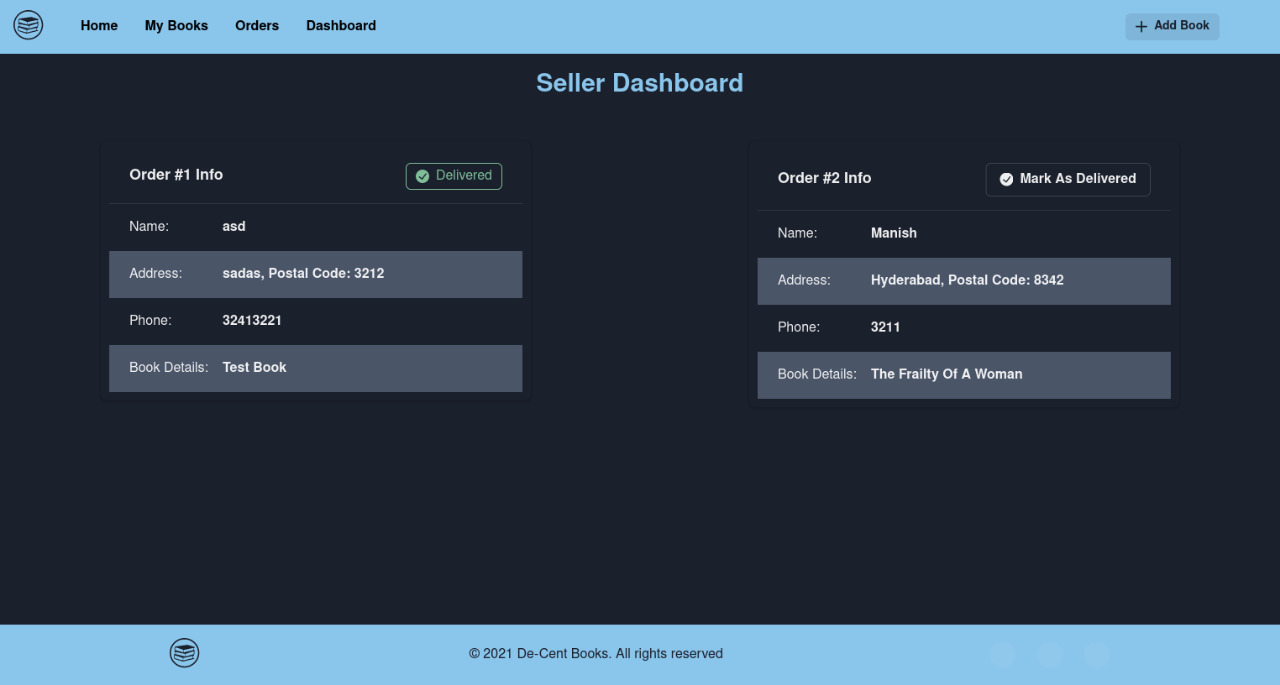


Fig.6.8 Seller Dashboard

**Chapter 7:** **CONCLUSION**

Through this web app, users can add, sell, buy and resell owned books using Ether on the Ethereum Blockchain Network.

Using this model, decentralization is implemented thus providing secure and efficient transactions to the users.

**Future Scope:**

This model can also be implemented for e-Books using ERC721 Non-Fungible Tokens which provide one NFT to each book. The owner of the token will be the owner of the book. The Ether transactions can be put on hold until the buyer receives the book thus implementing the Smart Contracts to their full extent.

**Chapter 8: BIBLIOGRAPHY**

Books used for collecting the information are:

1. Mastering Ethereum by Merunas Grincalaitis

The following websites are used for collecting the information:

[www.wikipedia.com](http://www.wikipedia.com)

<http://stackoverflow.com/>