DAYANANDA SAGAR UNIVERSITY

KUDLU GATE, BANGALORE – 560068

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING

MAJOR PROJECT PHASE II REPORT
ON
"DECENTRALIZED E-COMMERCE APPLICATION"

Submitted by:
Abhijeet Kumar (ENG19CS0009)
Amisha Asrani (ENG19CS0029)
Anjali Sharma (ENG19CS0035)
Habibul Bashar Ahmed (ENG19CS0107)

Under the supervision of
Prof. Nandini K
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING,
SCHOOL OF ENGINEERING
DAYANANDA SAGAR UNIVERSITY,
BANGALORE

(2022-2023)

DAYANANDA SAGAR UNIVERSITY

School of Engineering
Department of Computer Science & Engineering
Kudlu Gate, Bangalore-560068
Karnataka, India

CERTIFICATE

This is to certify that the Major Project Phase II work entitled "**Decentralised E-Comm Web app**" is carried out by **Abhijeet Kumar (ENG19CS0009), Amisha Asrani (ENG19CS0029), Anjali Sharma (ENG19CS0035) and Habibul Bashar Ahmed (ENG19CS0107)**, bonafide students of Bachelor of Technology in Computer Science & Engineering at the School of Engineering, Dayananda Sagar University, Bangalore in the partial fulfillment for the award of degree in Bachelor of Technology in Computer Science & Engineering, during **2022-2023.**

Prof. Nandini K	**Dr. Girisha G S**	**Dr. Udaya Kumar Reddy K R**
Assistant Professor	Chairman CSE	Dean
Dept. of CS&E,	School of Engineering,	School of Engineering,
School of Engineering,	Dayananda Sagar University	Dayananda Sagar University
Dayananda Sagar University		
Date:	Date:	Date:

Name of the Examiner	**Signature of Examiner**
1.	
2.	

DECLARATION

We, **Abhijeet Kumar (ENG19CS0009), Amisha Asrani (ENG19CS0029), Anjali Sharma (ENG19CS0035), Habibul Bashar Ahmed (ENG19CS0107)**, are students of eighth semester B.Tech in **Computer Science and Engineering**, at School of Engineering, **Dayananda Sagar University**, hereby declare that the Major Project Phase-II entitled -"**Decentralised E-Commerce Web app**", has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year 2022-2023.

Students	**Signature**
Abhijeet Kumar (ENG19CS0009)	
Amisha Asrani (ENG19CS0029)	
Anjali Sharma (ENG19CS0035)	
Habibul Bashar Ahmed (ENG19CS0107)	

Place: Bangalore

Date:

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.

First, we take this opportunity to express our sincere gratitude to School of Engineering & Technology, Dayananda Sagar University for providing us with a great opportunity to pursue our Bachelor's degree in this institution.

We would like to thank ***Dr. Udaya Kumar Reddy K R, Dean, School of Engineering & Technology, Dayananda Sagar University*** *for his constant encouragement and expert advice.*

It is a matter of immense pleasure to express our sincere thanks to ***Dr. Girisha G S, Department Chairman, Computer Science and Engineering, Dayananda Sagar University****, for providing the right academic guidance that made our task possible.*

We would like to thank our guide ***Prof. Nandini K, Assistant Professor, Dept. of Computer Science and Engineering, Dayananda Sagar University****, for sparing her valuable time to extend help in every step of our project work, which paved the way for smooth progress and fruitful culmination of the project.*

We would like to thank our ***Project Coordinator Dr. Meenakshi Malhotra and Dr. Pramod Kumar Naik, Department of Computer Science and Engineering****, and all the staff members of Computer Science and Engineering for their support.*

We are also grateful to our family and friends who provided us with every requirement throughout the course.

We would like to thank one and all who directly or indirectly helped us in the Project work.

Signature of Students

Abhijeet Kumar (ENG19CS0009)

Amisha Asrani (ENG19CS0029)

Anjali Sharma (ENG19CS0035)

Habibul Bashar Ahmed (ENG19CS0107)

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

dApp	Decentralized Application
HTML	Hypertext Markup Language
EVM	Ethereum Virtual Machine
API	Application Programming Interface

LIST OF FIGURES

ABSTRACT

Centralized e-commerce solutions are the mainstream consumer choice for online trade which leads to large companies owning big portions of our private data. The decentralization of e-commerce would provide a solution to the middleman, as business would be performed on a peer-to-peer basis. In the long term it might help everyone avoid tyranny by providing more freedoms, less censorship and more ownership of our own data, at cheaper costs.

The aim of this project is to explore a way to remove the middleman from our online commerce while still retaining the features and characteristics of centralized e-commerce platforms. Also, to evaluate and find out the appropriate ways to reduce the data breaching issues of e-commerce platforms by Blockchain technology.

This project will be able to provide a user-friendly application that implements Blockchain technology for database systems with its unique characteristics that can increase the level of security of e-commerce organizations' data and detect unauthorized access. Furthermore, Blockchain helps to prove the authenticity of products by ensuring the originality of goods in every step of the supply chain. This will ultimately help an e-commerce business acquire customers' loyalty and trust.

CHAPTER 1 INTRODUCTION

At present, centralized e-commerce solutions are the dominant preference of consumers for online trading, which results in large corporations holding significant amounts of our personal data. A decentralized e-commerce model would eliminate the need for intermediaries, as transactions would occur between individuals directly. In the future, this could reduce censorship, increase personal freedom, and enable us to have greater control over our data while being more cost-effective.

1.1. INTRODUCTION OF THE PROJECT:

The advent of blockchain technology has revolutionized the way we conduct transactions and exchange information. The decentralized and secure nature of blockchain makes it a suitable technology for various applications, including e-commerce. With the rapid growth of e-commerce, there is a growing need for a secure and transparent platform that can address the challenges faced by traditional e-commerce systems. This research paper aims to address this issue by proposing a blockchain-based e-commerce application that uses 'Hardhat' framework. Blockchain technology offers numerous benefits to the e-commerce industry, such as secure transactions, transparent record keeping, and reduced fraud. However, the adoption of blockchain technology in e-commerce has been slow due to the lack of a suitable platform that can support the complex requirements of e-commerce systems. The Hardhat framework is a powerful tool that can help developers streamline their development process and increase productivity when building Ethereum-based applications.

1.2. OBJECTIVE OF THE PROJECT:

- To explore a way to remove the middleman from our online commerce while still retaining the features and characteristics of centralized e-commerce platforms
- To evaluate and find out the appropriate ways to reduce the data breaching issues of e-commerce platforms by Blockchain technology.

1.3. SCOPE OF THE PROJECT:

- To be able to provide a user-friendly application that implements Blockchain technology for database systems with its unique characteristics that can increase the level of security of e-commerce organizations' data and detect unauthorized access.
- Also, Blockchain helps to prove the authenticity of products by ensuring the originality of goods in every step of the supply chain. This will ultimately help an e-commerce business acquire customers' loyalty and trust.

CHAPTER 2 PROBLEM DEFINITION

This project aims to implement Blockchain technology for database systems with its unique characteristics that can increase the level of security of e-commerce organizations’ data and detect unauthorized access.

Because dApps are decentralized, they are free from the control and interference of a single authority. The application aims to ensure the safeguarding of user privacy, the lack of censorship, and the flexibility of development.

CHAPTER 3 LITERATURE REVIEW

Sl. No.	Author's Name/ Paper Title	Year of Publication	Methodology	Key Findings	Research Gap
01.	Mohammad Monirujjaman Khan , Nesat Tasneem RoJa, Faris A. Almalki , and Maha Aljohani Revolutionizing E-Commerce Using Blockchain Technology and Implementing Smart Contract	May 2022	JavaScript, Postman, Smart contract, HyperText Markup Language (HTML), Cascading Style Sheets (CSS), Structured Query Language (SQL), and Hypertext Preprocessor (PHP).	They have used an algorithm called proof-ofwork (PoW). Proof-of-work is a decentralized consensus process that allows each node of the network to spend time.	Only implements smart contract for providing transactional privacy. Smart contracts are written privately without cryptographic algorithms.

Sl. No.	Author's Name/ Paper Title	Year of Publication	Methodology	Key Findings	Research Gap
02.	Teo Min Xuan, Maen T. Alrashdan, Qusay Al-Maatouk BLOCKCHAIN TECHNOLOGY IN ECOMMERCE PLATFORM	Oct 2020	Private blockchain is selected to implement in the system. CHAINSQL has been used to enhance the Blockchain network	This paper evaluates and finds out the appropriate ways to reduce the data breaches issues of e-commerce platforms by Blockchain technology.	Expensiveness should be improved. System requires highly skilled technical teams to support the system.

Sl. No.	Author's Name/ Paper Title	Year of Publication	Methodology	Key Findings	Research Gap
03.	Xiang Hong Li Blockchain-based Cross-border E-business Payment Model	2021	Multi Chain Structure, consists of 3 chains which are Account Chain IoT Chain and Transaction Chain	Cross-border e-business payment platform mainly includes registration transaction payment of three business functional modules.	Scalability - throughput and latency. Password service of blockchain technology.

Sl. No.	Author's Name/ Paper Title	Year of Publication	Methodology	Key Findings	Research Gap
04.	Gulshan Kumar, Rahul Saha, William J Buchanan Decentralized Accessibility of e-commerce Products through Blockchain	June 2020	Identity management and key generation. Rating based consensus process called Proof of Accomplishment (PoA).	The solution is beneficial for improving the traceability of the products Merging of value chain and supply chain into a single transparent blockchain-based solution.	Time consumption increases with increasing number of blocks. Reduced success rates with increasing number of blocks.

Sl. No.	Author's Name/ Paper Title	Year of Publication	Methodology	Key Findings	Research Gap
05.	Ji Jiang and Jin Chen Framework of Blockchain-Supported E-Commerce Platform for Small and Medium Enterprises	July 2021	Chain structure Encryption algorithm Smart contract	Chain structure of the blockchain guarantees that the data are real, tamper-resistant, and traceable, which is instrumental in following up information in real time.	Authenticity of data before they are recorded on the blockchain, which implies that all nodes are confronted with the threat of source data deception.

Sl. No.	Author's Name/ Paper Title	Year of Publication	Methodology	Key Findings	Research Gap
06.	Samer Shorman, Mohammad Allaymoun, Omer Hamid DEVELOPING THE E-COMMERCE MODEL A CONSUMER TO CONSUMER USING BLOCKCHAIN NETWORK TECHNIQUE	May 2019	Needed 3 important factors to put blockchain in implementation, include Decentralization Transparency Immutability	By using blockchain in a C2C business, effective mechanisms have been created to achieve quality control and payment guarantees.	A detailed verification of technical aspects and remittances.

CHAPTER 4 PROJECT DESCRIPTION

4.1 SYSTEM DESIGN:

There are many E-commerce applications like Amazon, Flipkart but a Decentralized e-commerce application would protect its user's data from unauthorized access and provide secured transactions without any intermediaries. A decentralized e-commerce ensures that the products are ethically sourced, which is an important issue for many ecommerce businesses.

The clients experience the problem of expensive and long payment processes. The payment process included some obstructive steps to settle a transaction and extra fees to pay for payment gateways. To cut this long route short, we have started the blockchain-based project that consisted of the following parts:

1. **Blockchain Node:** This is the main node that connects to the blockchain network and stores the transaction data. The node maintains a copy of the blockchain and validates transactions before adding them to the blockchain.
2. **Smart Contract Layer:** This layer contains the smart contracts that execute on the blockchain. Smart contracts are self-executing code that can automate the transfer of assets or data between parties. In an e-commerce app, smart contracts can be used to automate the payment and delivery process.
3. **E-commerce Application:** This is the frontend application that interacts with the smart contract layer to enable e-commerce transactions. The application allows users to browse and purchase products, view their order history, and manage their payments.

In this architecture, the blockchain node stores all the transaction data, ensuring that transactions are secure and transparent. The smart contract layer automates the payment and delivery process, reducing the need for intermediaries and increasing the efficiency of the transaction. Finally, the e-commerce application provides a user-friendly interface for customers to browse and purchase products.

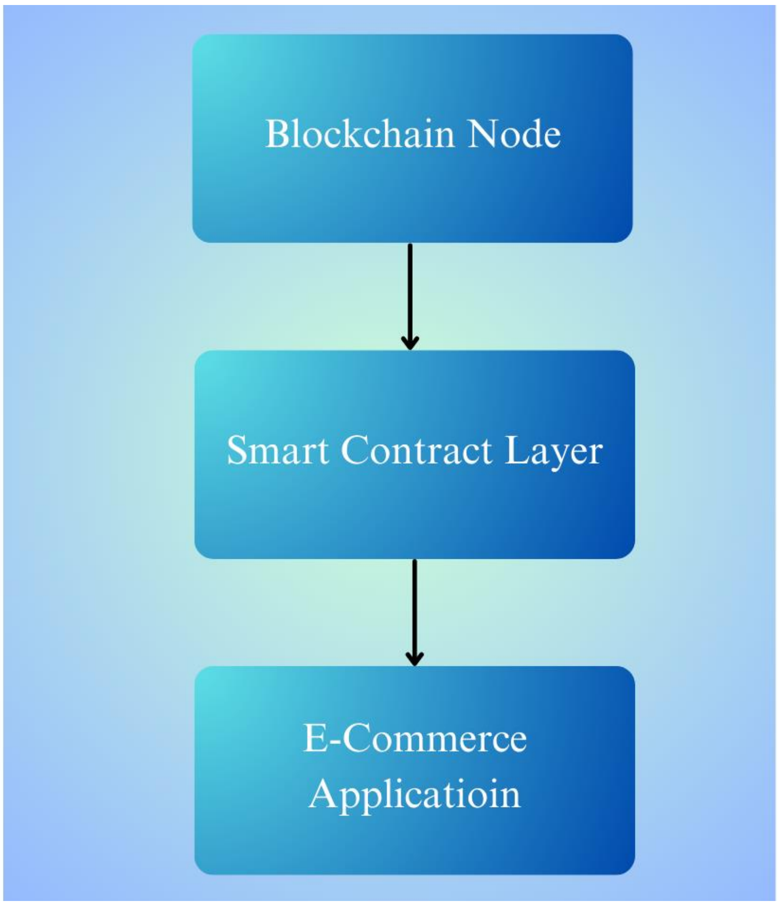


Fig.4.1 System Design

4.2 ASSUMPTIONS & DEPENDENCIES:

4.2.1 ASSUMPTIONS:

- The web app will rely on blockchain technology to provide decentralized storage, processing, and transaction capabilities.
- The users of the web app will have access to a reliable and secure internet connection.
- The users of the web app will have access to the necessary hardware and software to interact with the blockchain technology, such as a digital wallet.
- It will be able to provide a seamless and user-friendly experience for both buyers and sellers.
- It will comply with relevant regulations and legal requirements.

4.1.2 DEPENDENCIES:

- The web app will depend on a blockchain network that is scalable and secure enough to handle the volume of transactions and data associated with e-commerce.
- It will depend on a community of developers, miners, and users who are committed to the success and growth of the blockchain network.
- It will depend on a network of payment processors and gateways that can facilitate transactions between buyers and sellers using traditional fiat currencies as well as cryptocurrencies.
- The web app will depend on a system of reputation and dispute resolution mechanisms that can protect buyers and sellers from fraudulent or unreliable actors.
- It will depend on the adoption and acceptance of cryptocurrencies as a legitimate form of payment by consumers and merchants alike.

CHAPTER 5 REQUIREMENTS

5.1 FUNCTIONAL REQUIREMENTS:

- 3rd Party Integrations
- Mobile Responsive
- Product Attributes
- Order & Checkout Flow
- Social Sharing

5.2 NON-FUNCTIONAL REQUIREMENTS:

- Usability
- Security
- Performance
- Maintainability
- Scalability

5.3 SOFTWARE/SYSTEM REQUIREMENTS:

5.3.1 SOFTWARE REQUIREMENTS:

- Solidity
- JavaScript
- Hardhat
- Ether.js
- React.js

5.3.2 HARDWARE REQUIREMENTS:

- Processor- core i3 or more
- Hard disk- Minimum 5GB
- Memory- Minimum 4GB RAM

CHAPTER 6 METHODOLOGY

6.1 WORKFLOW OF A DECENTRALIZED APP:

In determining how to create dApps, the workflow of a dApp is similar to other traditional apps, but with some minor differences. Usually, there are three main components: front-end user interface, smart contract, and a backend server.

6.1.1 A FRONT-END USER INTERFACE

The front-end can take many forms, but in its simplest form, it is an HTML page that uses JavaScript to make calls to the smart contract. This step is very similar to building a regular web application. We have used React to design the Frontend. In developing a dApp, to make the front-end communicate with the smart contract and blockchain, we have used Ethers.js.

6.1.2 A SMART CONTRACT

This is where the magic happens. A smart contract is simply a program that runs on the EVM. The smart contract contains the business logic, which is equivalent to the code that runs on traditional servers. For example, an online crowdfunding platform uses a smart contract to control how funds are collected and dispersed. We have used Solidity to write Smart Contracts and Tests.

6.1.3 A BACKEND SERVER

In how to create a decentralized application, this server could be written in Ruby, NodeJS, or anything else you’re comfortable with. Your backend server could run a local copy of an Ethereum client, such as Geth or Parity, to receive events from the blockchain and forward them to the client. Alternatively, it might use an API service provider such as Infura to connect to the Ethereum network without needing to run a local client.

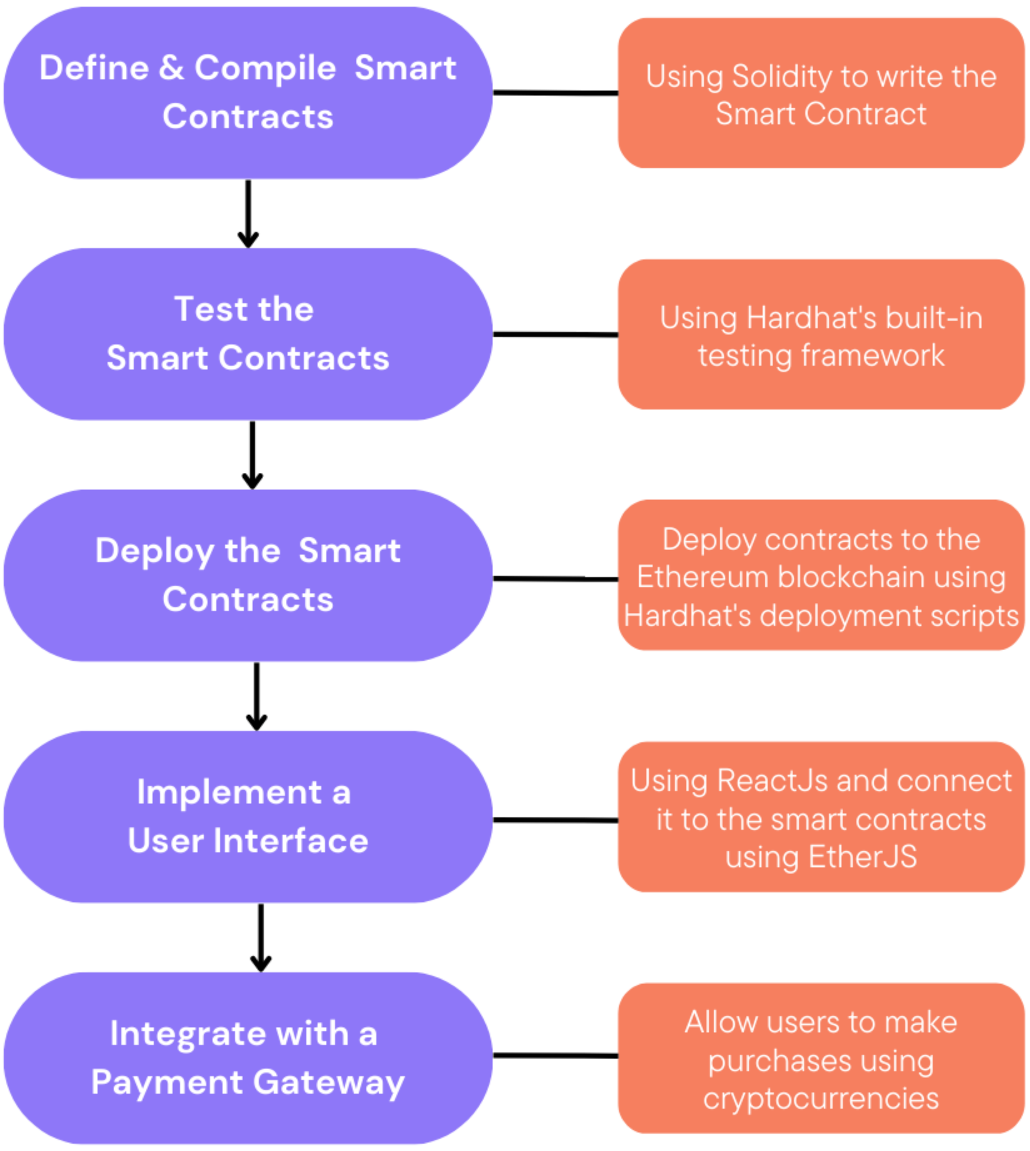

Define & Compile Smart Contracts
Using Solidity to write the Smart Contract
Test the Smart Contracts
Using Hardhat's built-in testing framework
Deploy the Smart Contracts
Deploy contracts to the Ethereum blockchain using Hardhat's deployment scripts
Implement a User Interface
Using ReactJs and connect it to the smart contracts using EtherJS
Integrate with a Payment Gateway
Allow users to make purchases using cryptocurrencies

Fig.6.1. Workflow of a decentralized E-Commerce Web App

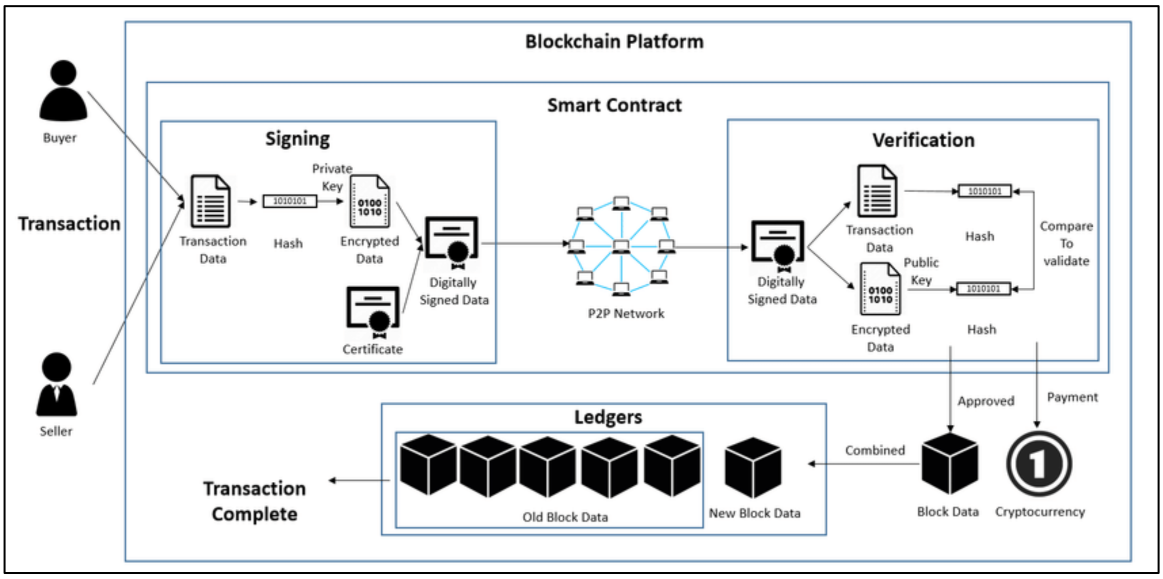


Fig.6.2. Block Diagram of a Blockchain Transaction

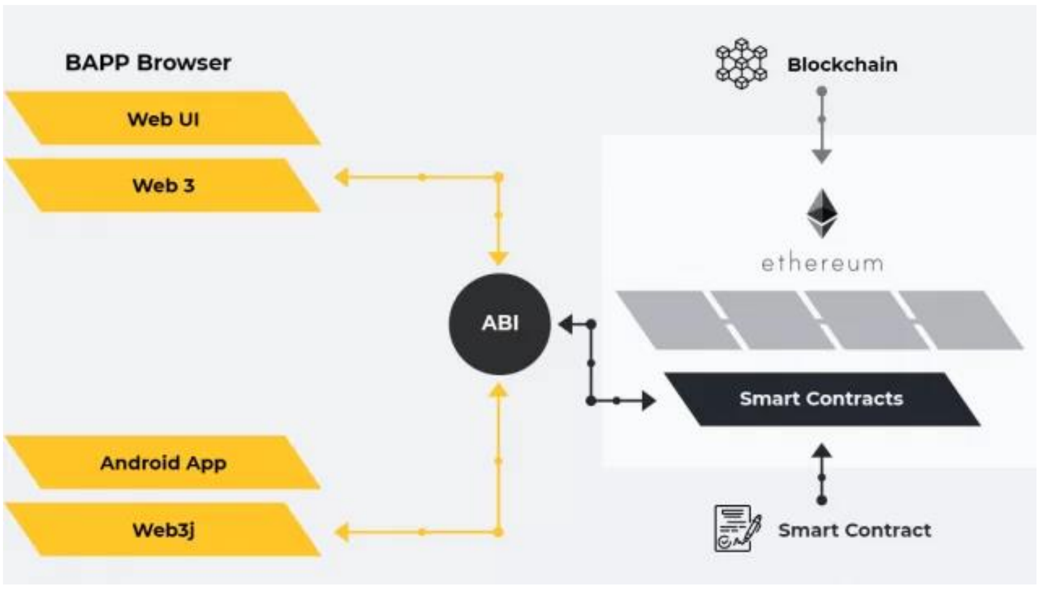


Fig.6.3. Simple architecture of a dApp

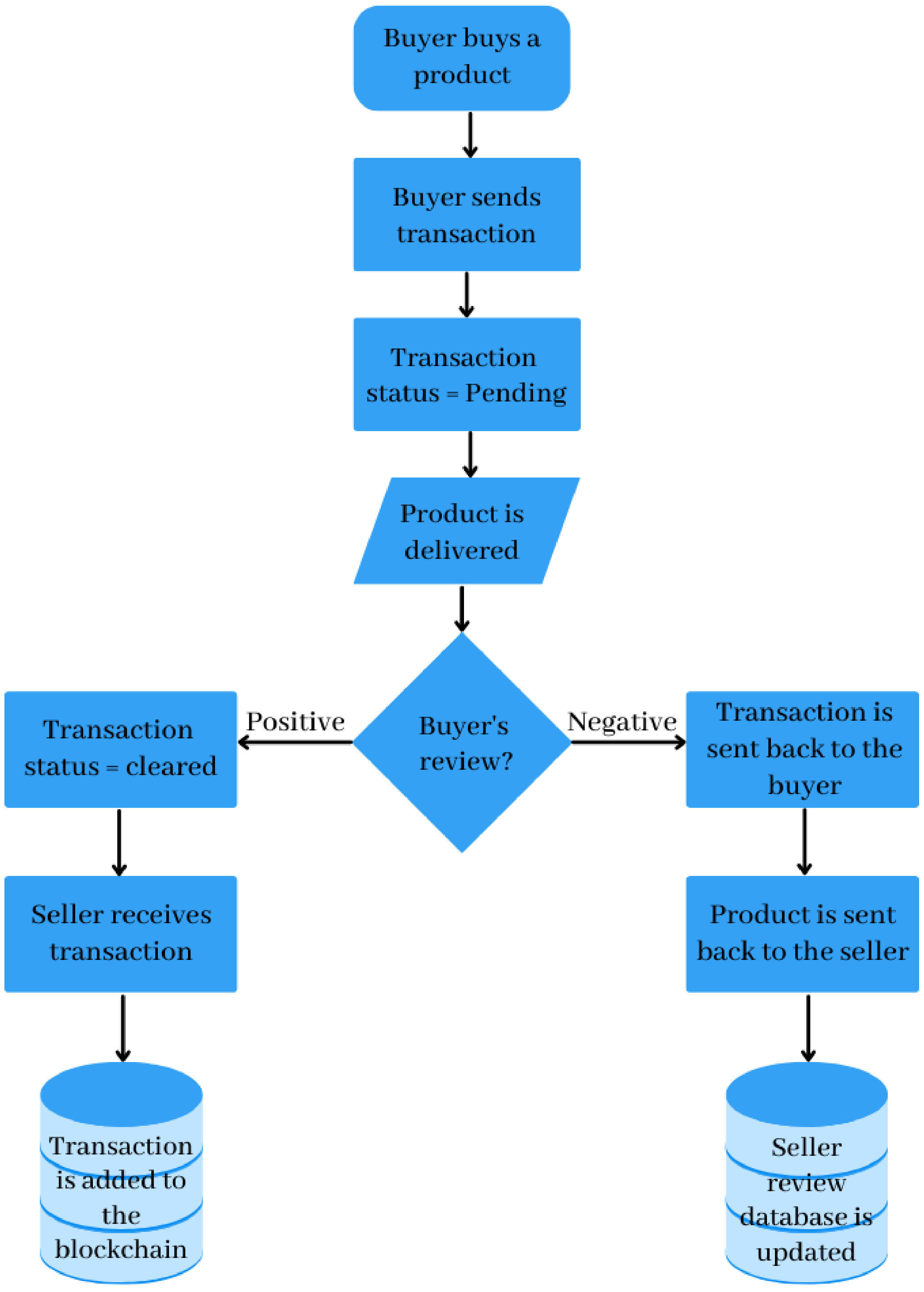

Buyer buys a product
Buyer sends transaction
Transaction status = Pending
Product is delivered
Buyer's review?
Positive
Negative
Transaction status = cleared
Seller receives transaction
Transaction is added to the blockchain
Transaction is sent back to the buyer
Product is sent back to the seller
Seller review database is updated

Fig.6.4. Flowchart of a complete Blockchain Transaction

CHAPTER 7 EXPERIMENTATION

7.1 SOFTWARE DEVELOPMENT:

This smart contract allows buyers to place orders for products by calling the placeOrder function. Here's how the function works:

1. The function takes two parameters: _productName and _quantity, which represent the name of the product and the quantity being ordered.

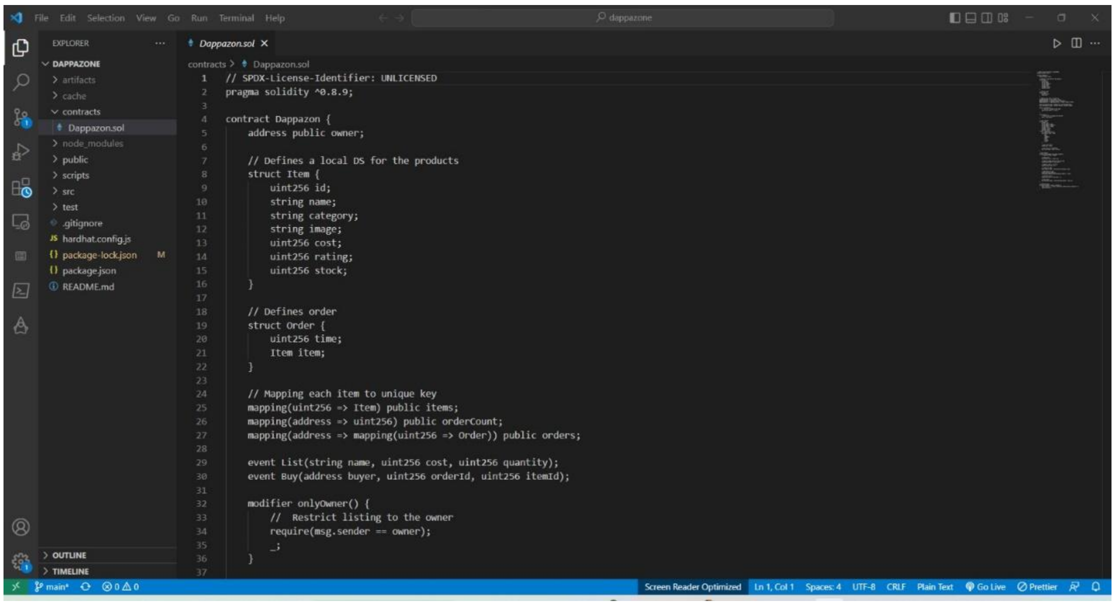
2. The function generates a unique order ID by hashing together several values, including the current block timestamp, the address of the buyer, the product name, and the quantity.

3. The function creates a new Order struct with the details of the order, including the order ID, the buyer's address, the product name, the quantity, and a boolean flag indicating whether the order has been shipped.

4. The function adds the new order to the orders mapping, with the order ID as the key and the Order struct as the value.

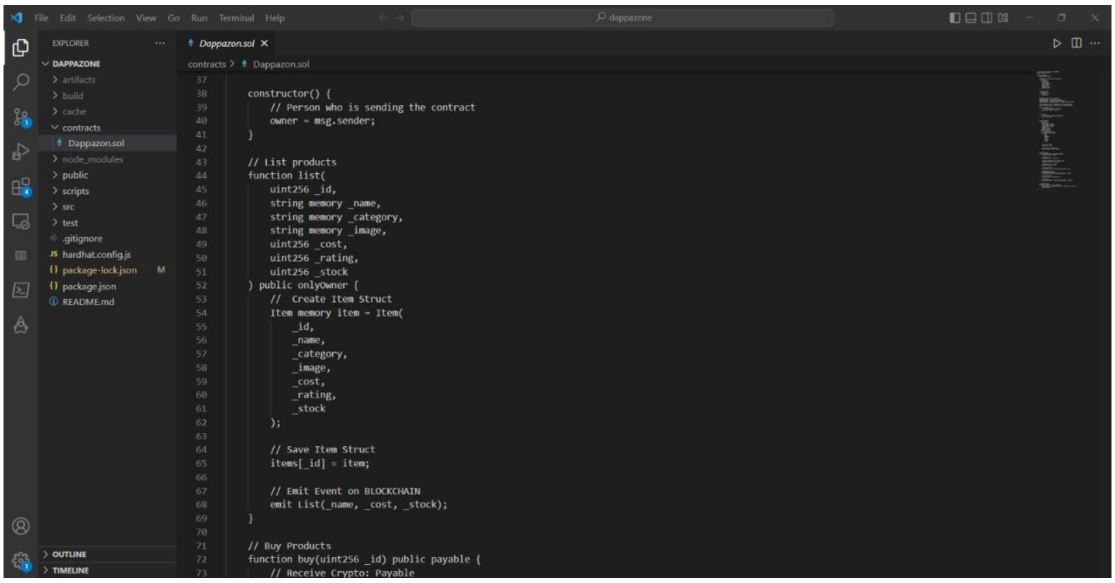
5. The function emits a NewOrder event with the details of the new order, including the order ID, the buyer's address, the product name, and the quantity.

6. Buyers can call the placeOrder function to place a new order, which will be stored in the orders mapping. The unique order ID ensures that each order is unique and can be easily tracked. The NewOrder event allows anyone to listen for new orders and take action based on the details of the order.



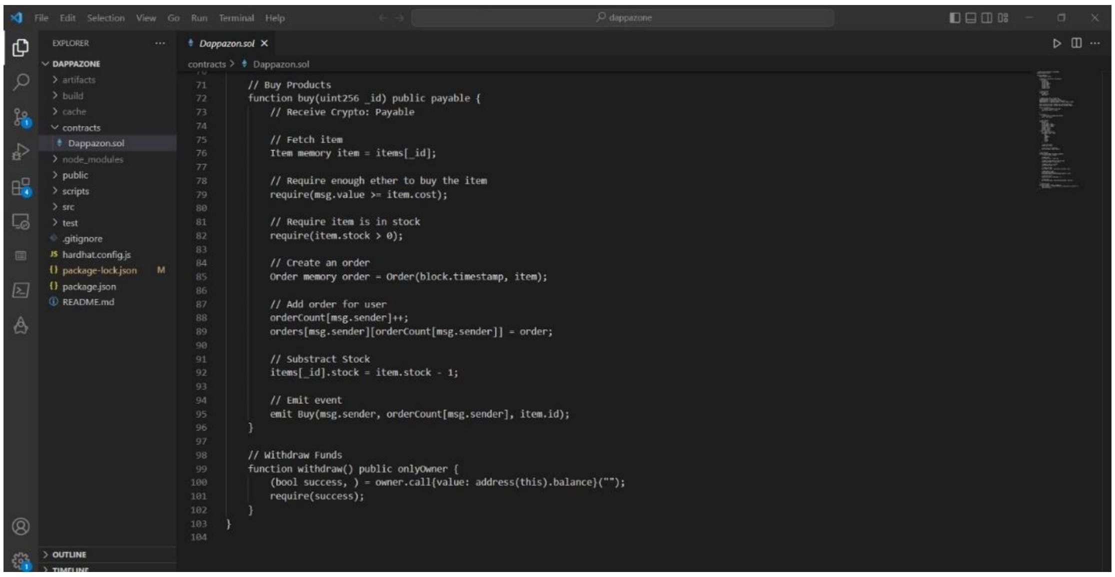
```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.9;

contract Dappazon {
    address public owner;

    // Defines a local DS for the products
    struct Item {
        uint256 id;
        string name;
        string category;
        string image;
        uint256 cost;
        uint256 rating;
        uint256 stock;
    }

    // Defines order
    struct Order {
        uint256 time;
        Item item;
    }

    // Mapping each item to unique key
    mapping(uint256 => Item) public items;
    mapping(address => uint256) public orderCount;
    mapping(address => mapping(uint256 => Order)) public orders;

    event List(string name, uint256 cost, uint256 quantity);
    event Buy(address buyer, uint256 orderId, uint256 itemId);

    modifier onlyOwner() {
        //  Restrict listing to the owner
        require(msg.sender == owner);
        _;
    }

```

Fig.7.1.1 Implementation of Smart Contract



```solidity

    constructor() {
        // Person who is sending the contract
        owner = msg.sender;
    }

    // List products
    function list(
        uint256 _id,
        string memory _name,
        string memory _category,
        string memory _image,
        uint256 _cost,
        uint256 _rating,
        uint256 _stock
    ) public onlyOwner {
        //  Create Item Struct
        Item memory item = Item(
            _id,
            _name,
            _category,
            _image,
            _cost,
            _rating,
            _stock
        );

        // Save Item Struct
        items[_id] = item;

        // Emit Event on BLOCKCHAIN
        emit List(_name, _cost, _stock);
    }

    // Buy Products
    function buy(uint256 _id) public payable {
        // Receive Crypto: Payable
```

Fig.7.1.2. Implementation of Smart Contract



```solidity
    // Buy Products
    function buy(uint256 _id) public payable {
        // Receive Crypto: Payable

        // Fetch item
        Item memory item = items[_id];

        // Require enough ether to buy the item
        require(msg.value >= item.cost);

        // Require item is in stock
        require(item.stock > 0);

        // Create an order
        Order memory order = Order(block.timestamp, item);

        // Add order for user
        orderCount[msg.sender]++;
        orders[msg.sender][orderCount[msg.sender]] = order;

        // Substract Stock
        items[_id].stock = item.stock - 1;

        // Emit event
        emit Buy(msg.sender, orderCount[msg.sender], item.id);
    }

    // Withdraw Funds
    function withdraw() public onlyOwner {
        (bool success, ) = owner.call{value: address(this).balance}("");
        require(success);
    }
}
```

Fig.7.1.3. Implementation of Smart Contract

This Solidity code defines a smart contract called `Dappazon` that allows the owner of the contract to list products for sale and for buyers to purchase those products by sending ether to the contract.

The `Dappazon` contract has two local data structures: `Item` and `Order`. The `Item` struct represents a product being sold on the platform and includes an ID, name, category, image, cost, rating, and stock count. The `Order` struct represents a purchase made by a buyer and includes a timestamp and the purchased `Item`.

The contract also has three mappings: `items`, `orderCount`, and `orders`. The `items` mapping maps item IDs to their corresponding `Item` struct. The `orderCount` mapping maps user addresses to the number of orders they have made on the platform. The `orders` mapping maps user addresses and order IDs to the corresponding `Order` struct.

The `Dappazon` contract has three functions: `list()`, `buy()`, and `withdraw()`.

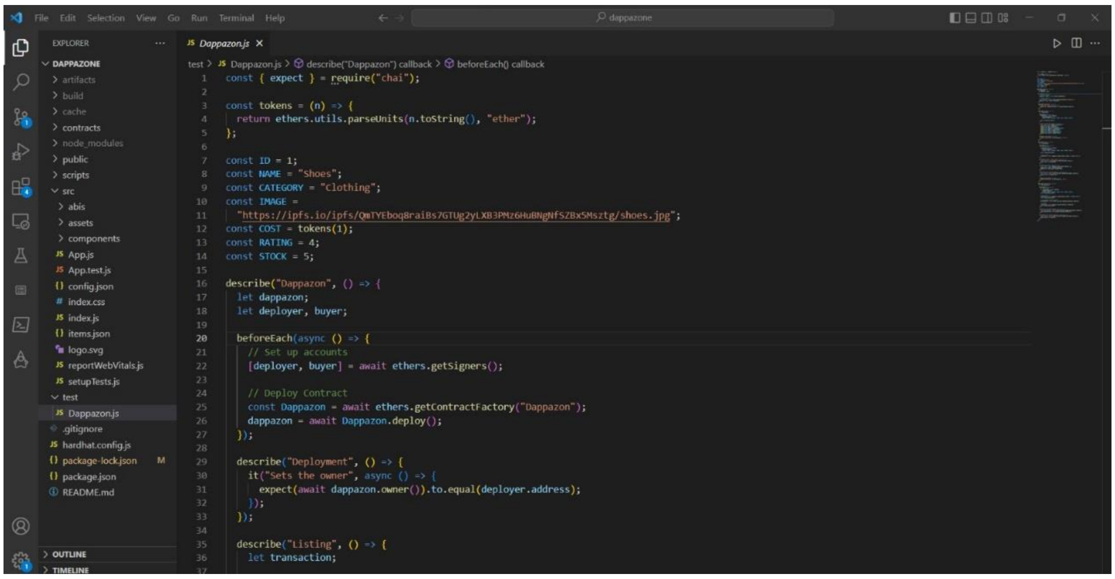
The `list()` function allows the contract owner to add a new `Item` to the `items` mapping. The owner must specify the `id`, `name`, `category`, `image`, `cost`, `rating`, and `stock` count of the item being listed.

The `buy()` function allows a user to purchase an `Item` by sending enough ether to the contract to cover the `Item` cost. The function fetches the `Item` corresponding to the provided ID, checks that the user has sent enough ether and that the `Item` is in stock, creates a new `Order` with the current timestamp and the purchased `Item`, adds the `Order` to the `orders` mapping for the user, and updates the stock count of the purchased `Item`.

The `withdraw()` function allows the contract owner to withdraw any ether stored in the contract. This function sends all the ether stored in the contract to the owner's address.

CHAPTER 8 TESTING & RESULTS

8.1 TESTING:



```javascript
const { expect } = require("chai");

const tokens = (n) => {
  return ethers.utils.parseUnits(n.toString(), "ether");
};

const ID = 1;
const NAME = "Shoes";
const CATEGORY = "Clothing";
const IMAGE =
  "https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx5Msztg/shoes.jpg";
const COST = tokens(1);
const RATING = 4;
const STOCK = 5;

describe("Dappazon", () => {
  let dappazon;
  let deployer, buyer;

  beforeEach(async () => {
    // Set up accounts
    [deployer, buyer] = await ethers.getSigners();

    // Deploy Contract
    const Dappazon = await ethers.getContractFactory("Dappazon");
    dappazon = await Dappazon.deploy();
  });

  describe("Deployment", () => {
    it("Sets the owner", async () => {
      expect(await dappazon.owner()).to.equal(deployer.address);
    });
  });

  describe("Listing", () => {
    let transaction;
```

Fig.8.1.1. Testing using Hardhat

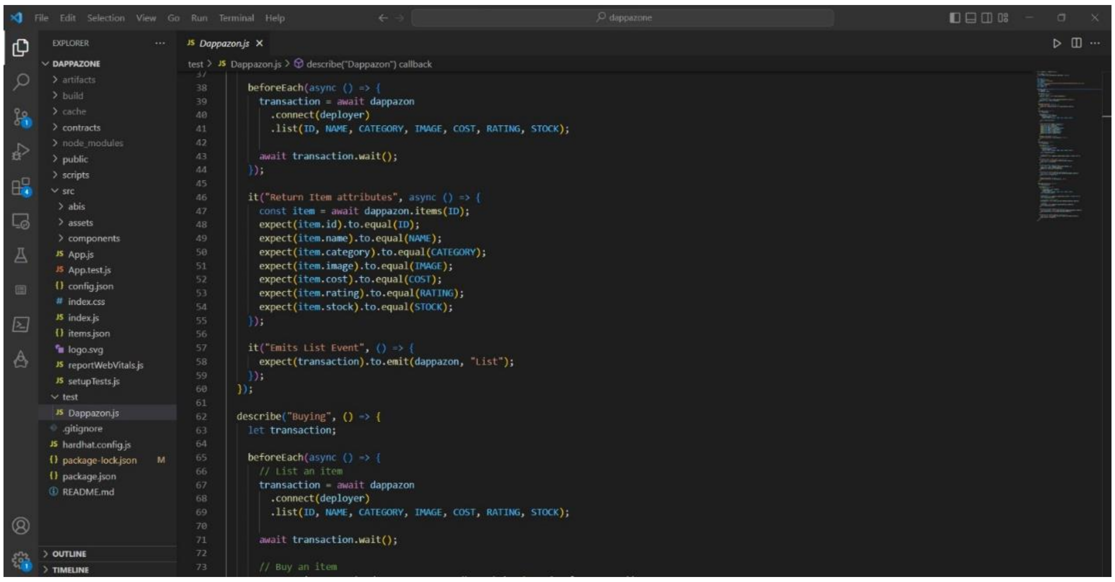
This is a JavaScript code that defines a set of tests for a smart contract called "Dappazon". The tests are using the Chai library for assertions and the ethers library for interacting with the Ethereum network. Here's a breakdown of what's happening in the code:

The first line is importing the `expect` function from the Chai library, which will be used to make assertions in the tests.

The `tokens` function is defined to convert a number into the corresponding value in Ethereum wei (the smallest unit of ether). This function is used to set the `COST` constant.

The `ID`, `NAME`, `CATEGORY`, `IMAGE`, `COST`, `RATING`, and `STOCK` constants are defined to set up an item that will be listed and bought in the tests.

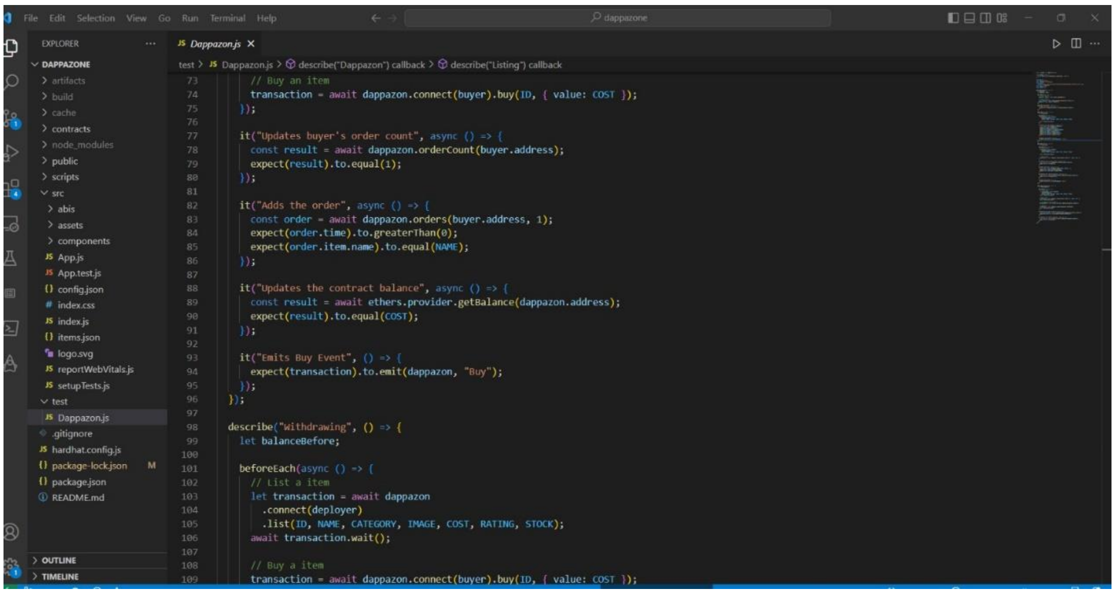
The `describe` function is used to group related tests together. The first group is for testing the contract deployment, the second group is for testing listing an item, the third group is for testing buying an item, and the fourth group is for testing withdrawing funds from the contract.



```
    beforeEach(async () => {
      transaction = await dappazon
        .connect(deployer)
        .list(ID, NAME, CATEGORY, IMAGE, COST, RATING, STOCK);

      await transaction.wait();
    });

    it("Return Item attributes", async () => {
      const item = await dappazon.items(ID);
      expect(item.id).to.equal(ID);
      expect(item.name).to.equal(NAME);
      expect(item.category).to.equal(CATEGORY);
      expect(item.image).to.equal(IMAGE);
      expect(item.cost).to.equal(COST);
      expect(item.rating).to.equal(RATING);
      expect(item.stock).to.equal(STOCK);
    });

    it("Emits List Event", () => {
      expect(transaction).to.emit(dappazon, "List");
    });
  });

  describe("Buying", () => {
    let transaction;

    beforeEach(async () => {
      // List an item
      transaction = await dappazon
        .connect(deployer)
        .list(ID, NAME, CATEGORY, IMAGE, COST, RATING, STOCK);

      await transaction.wait();

      // Buy an item
```

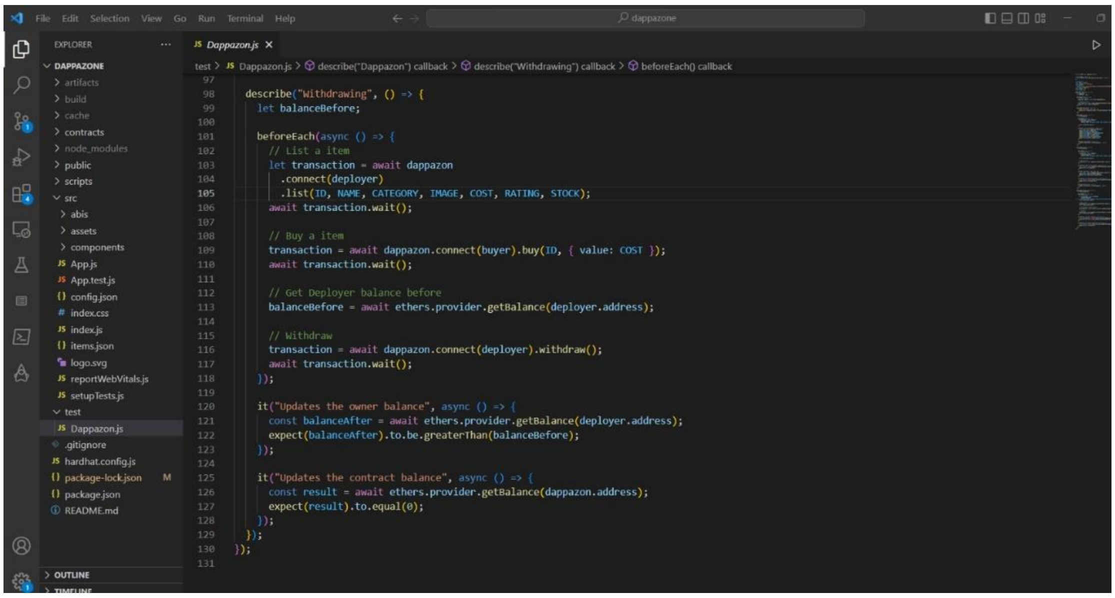
Fig.8.1.2. Testing using hardhat

The `beforeEach` function is used to set up some common state before each test in a group is run. In this case, it sets up the `deployer` and `buyer` variables by getting the signers from ethers, and deploys a new instance of the `Dappazon` contract using the `Dappazon` factory.



```
      // Buy an item
      transaction = await dappazon.connect(buyer).buy(ID, { value: COST });
    });

    it("Updates buyer's order count", async () => {
      const result = await dappazon.orderCount(buyer.address);
      expect(result).to.equal(1);
    });

    it("Adds the order", async () => {
      const order = await dappazon.orders(buyer.address, 1);
      expect(order.time).to.greaterThan(0);
      expect(order.item.name).to.equal(NAME);
    });

    it("Updates the contract balance", async () => {
      const result = await ethers.provider.getBalance(dappazon.address);
      expect(result).to.equal(COST);
    });

    it("Emits Buy Event", () => {
      expect(transaction).to.emit(dappazon, "Buy");
    });
  });

  describe("Withdrawing", () => {
    let balanceBefore;

    beforeEach(async () => {
      // List a item
      let transaction = await dappazon
        .connect(deployer)
        .list(ID, NAME, CATEGORY, IMAGE, COST, RATING, STOCK);
      await transaction.wait();

      // Buy a item
      transaction = await dappazon.connect(buyer).buy(ID, { value: COST });
```

Fig.8.1.3. Testing using hardhat



```
  describe("Withdrawing", () => {
    let balanceBefore;

    beforeEach(async () => {
      // List a item
      let transaction = await dappazon
        .connect(deployer)
        .list(ID, NAME, CATEGORY, IMAGE, COST, RATING, STOCK);
      await transaction.wait();

      // Buy a item
      transaction = await dappazon.connect(buyer).buy(ID, { value: COST });
      await transaction.wait();

      // Get Deployer balance before
      balanceBefore = await ethers.provider.getBalance(deployer.address);

      // Withdraw
      transaction = await dappazon.connect(deployer).withdraw();
      await transaction.wait();
    });

    it("Updates the owner balance", async () => {
      const balanceAfter = await ethers.provider.getBalance(deployer.address);
      expect(balanceAfter).to.be.greaterThan(balanceBefore);
    });

    it("Updates the contract balance", async () => {
      const result = await ethers.provider.getBalance(dappazon.address);
      expect(result).to.equal(0);
    });
  });
});
```

Fig.8.1.4. Testing using hardhat

The first test in the deployment group checks that the owner of the contract is the `deployer` account.

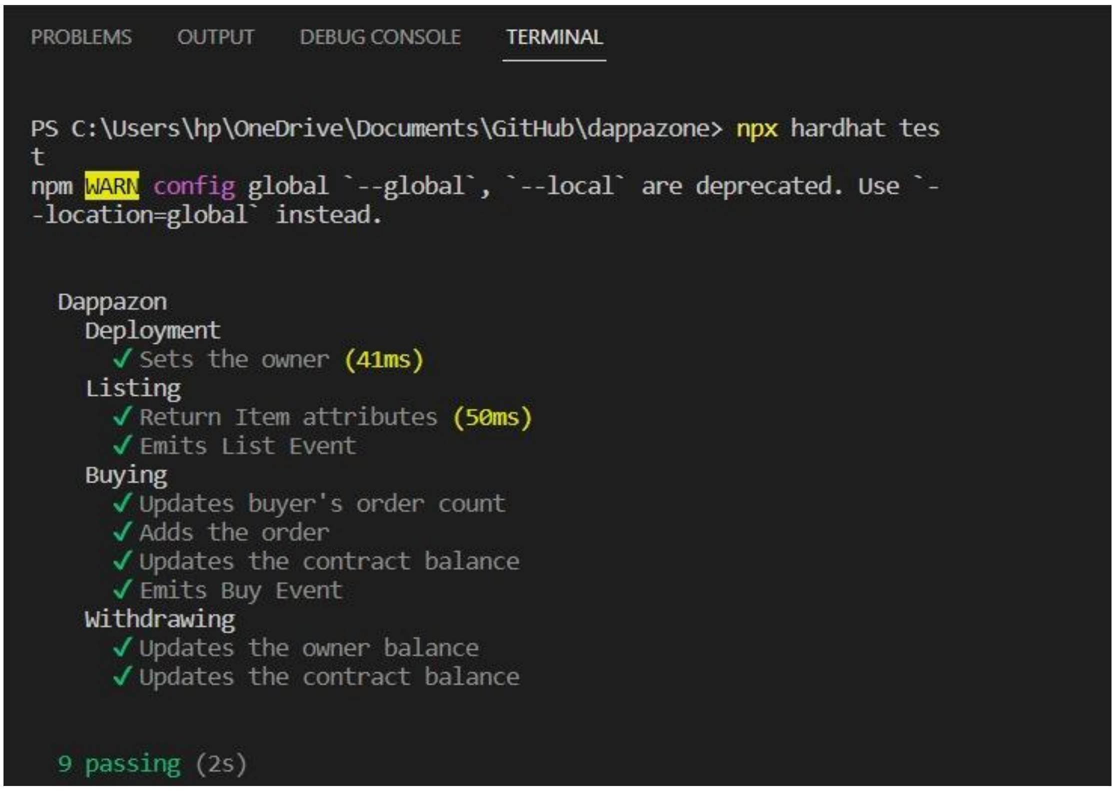
The first test in the listing group lists an item using the `list` function of the contract, and then checks that the item attributes are set correctly using the `items` function of the contract. It also checks that a `List` event is emitted when the item is listed.

The first test in the buying group lists an item, buys it using the `buy` function of the contract, and then checks that the buyer's order count is updated, the order is added to the orders mapping of the contract, the contract's balance is updated, and a `Buy` event is emitted when the item is bought.

The first test in the withdrawing group lists an item, buys it, withdraws the funds from the contract using the `withdraw` function of the contract, and then checks that the owner's balance is updated and the contract's balance is set to zero.

Overall, this code defines a set of tests to verify that the Dappazon contract works correctly for listing and buying items, and withdrawing funds from the contract.

8.2 RESULTS:



```
PS C:\Users\hp\OneDrive\Documents\GitHub\dappazone> npx hardhat tes
t
npm WARN config global `--global`, `--local` are deprecated. Use `-
-location=global` instead.


  Dappazon
    Deployment
      ✓ Sets the owner (41ms)
    Listing
      ✓ Return Item attributes (50ms)
      ✓ Emits List Event
    Buying
      ✓ Updates buyer's order count
      ✓ Adds the order
      ✓ Updates the contract balance
      ✓ Emits Buy Event
    Withdrawing
      ✓ Updates the owner balance
      ✓ Updates the contract balance


  9 passing (2s)
```

Fig.8.2.1. Testing smart contract Results

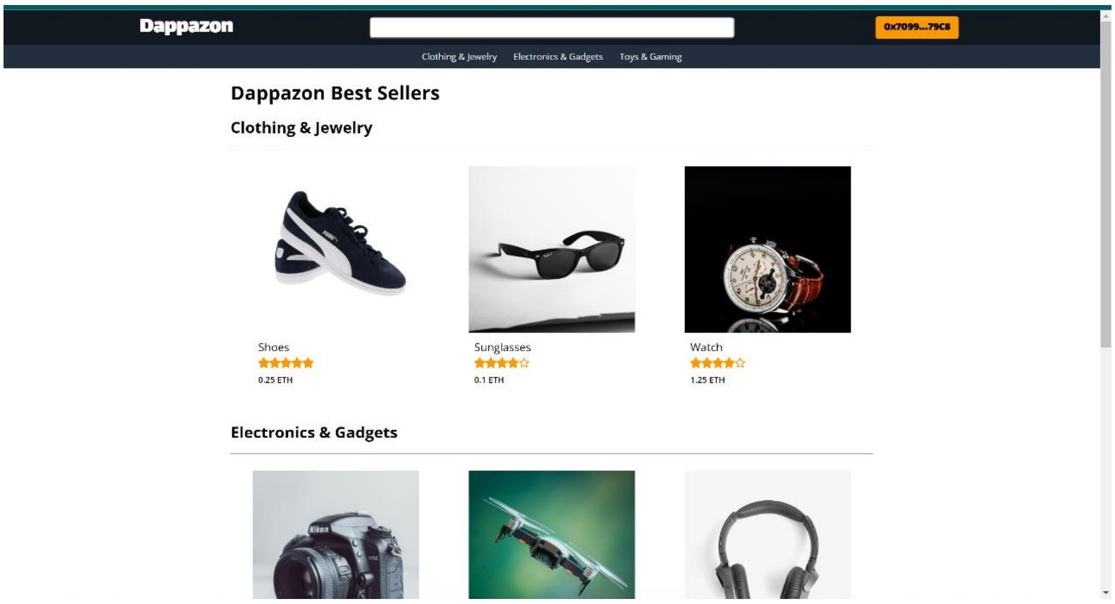
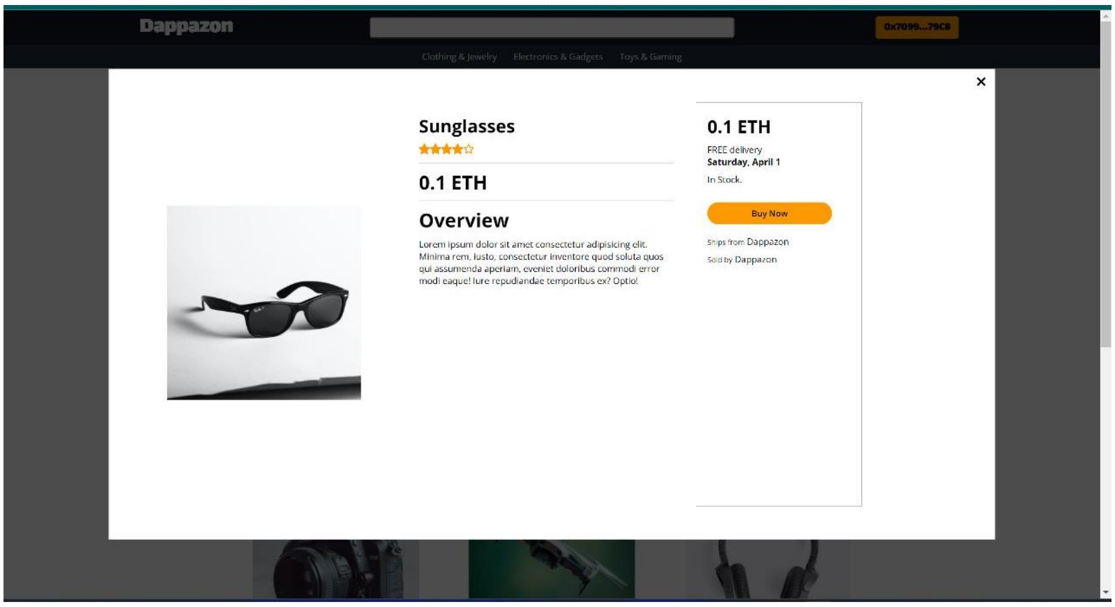


Fig.8.2.2. User Home page

This is the Home page of the web app, wherein the user can see all the listed products. The products are listed based on categories. The price and the ratings of the products are shown below the products.



Sunglasses

0.1 ETH

Overview

Lorem ipsum dolor sit amet consectetur adipisicing elit. Minima rem, iusto, consectetur inventore quod soluta quos qui assumenda aperiam, eveniet doloribus commodi error modi eaque! Iure repudiandae temporibus ex? Optio!

0.1 ETH

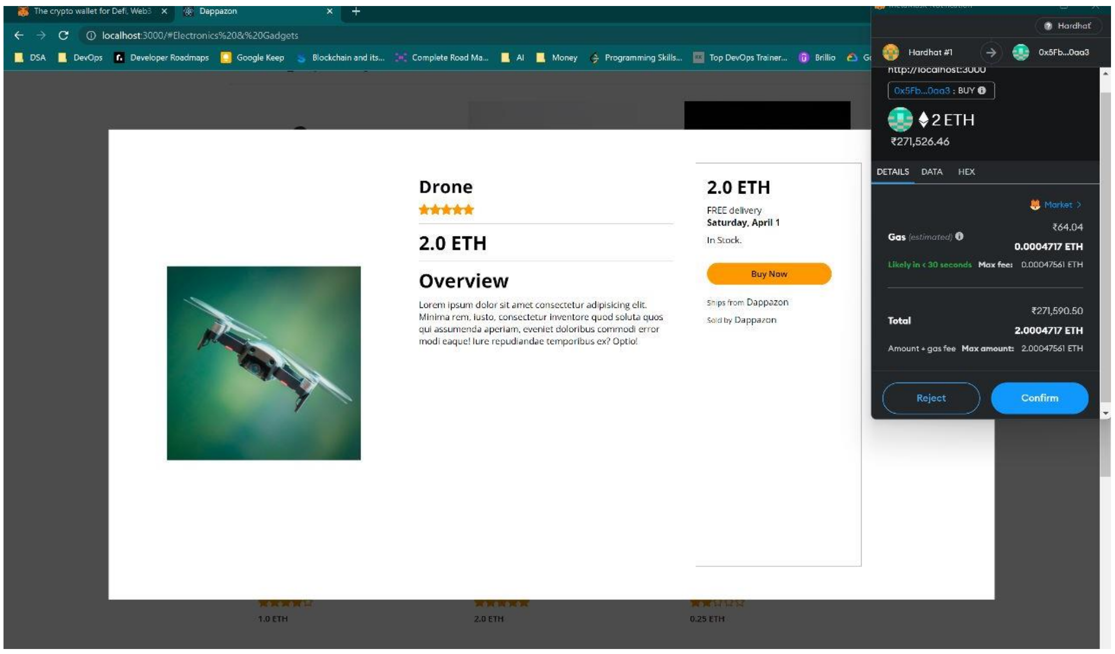
FREE delivery
Saturday, April 1

In Stock.

Ships from Dappazon
Sold by Dappazon

Fig.8.2.3. Product Description

On clicking the product, the user will be displayed the description of the products. The user gets the ‘Buy Now’ option here. On clicking which, the user will be directed to the Payment page to initiate the transaction.



Drone

2.0 ETH

Overview

Lorem ipsum dolor sit amet consectetur adipisicing elit. Minima rem, iusto, consectetur inventore quod soluta quos qui assumenda aperiam, eveniet doloribus commodi error modi eaque! Iure repudiandae temporibus ex? Optio!

2.0 ETH

FREE delivery
Saturday, April 1

In Stock.

Ships from Dappazon
Sold by Dappazon

1.0 ETH
2.0 ETH
0.25 ETH

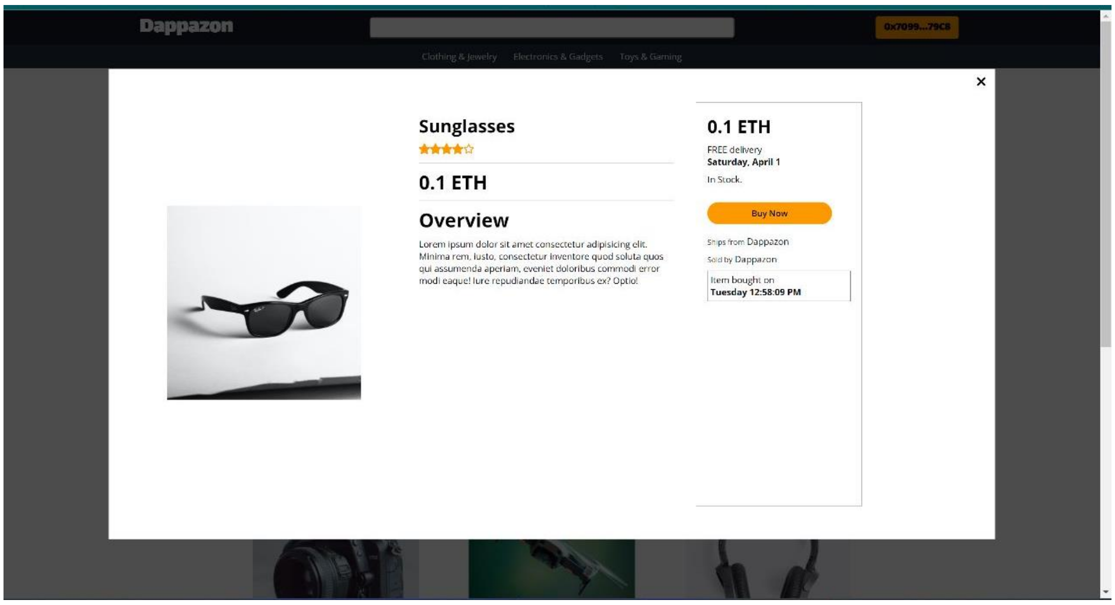
Hardhat
Hardhat #1 → 0x5Fb...0aa3
http://localhost:3000
0x5Fb...0aa3 : BUY
2 ETH
₹271,526.46

Gas (estimated) ₹64.04
0.0004717 ETH
Likely in < 30 seconds Max fee: 0.00047561 ETH

Total ₹271,590.50
2.0004717 ETH
Amount + gas fee Max amount: 2.00047561 ETH

Fig.8.2.4. Transaction initiated

This is the payment page from where the user can make the payment to the seller.



Sunglasses

★★★★☆

0.1 ETH

Overview

Lorem ipsum dolor sit amet consectetur adipisicing elit. Minima rem, iusto, consectetur inventore quod soluta quos qui assumenda aperiam, eveniet doloribus commodi error modi eaque! Iure repudiandae temporibus ex? Optio!

0.1 ETH

FREE delivery
Saturday, April 1

In Stock.

Ships from Dappazon

Sold by Dappazon

Item bought on
Tuesday 12:58:09 PM

Fig.8.2.4. Item bought

On completing the transaction, the user can see the summary of the transaction and the delivery date.

CHAPTER 9 CONCLUSION & FUTURE WORK

9.1 CONCLUSION:

It takes much more than a good idea to run a successful business. In our ever-expanding digital age, new opportunities are evolving every day for entrepreneurs. Roughly, over the last decade, e-commerce has expanded rapidly, with the US Commerce Department reporting 17.7% growth rates for 2018. But in this new landscape of online commerce, the challenges facing small business owners are not the same as they were in the previous decades.

Finding the correct products to sell, maintaining a loyal customer base, and scaling business models are all challenges that have evolved rapidly in this space. But perhaps the greatest struggle is determining the medium through which customers will buy your products. The Phore Decentralized Marketplace offers entrepreneurs and sellers the opportunity to sell their products across the globe, without the hassle of maintaining your own website — without the wet blanket of fees stifling your business growth.

Merchants also are able to offer their products at a more competitive price, since the Phore Marketplace does not charge such high fees. Zero to low fees means that merchants can improve their profit margin, as well as increase their respective market share by offering lower prices than the competition that is selling on sites such as Amazon and eBay.

9.2 FUTURE WORK:

One potential area for future work is to improve the user interface and user experience of the platform. This could involve conducting user research to identify pain points and areas for improvement, as well as iterating on the design and layout of the platform based on user feedback. Additionally, incorporating machine learning and artificial intelligence technologies could help to personalize the user experience and make the platform more intuitive and user-friendly.

Finally, there is a significant opportunity to expand the scope and reach of the decentralized e-commerce web app. This could involve partnering with additional merchants and vendors to offer a wider range of products and services, or expanding into new geographical regions to reach a larger user base. Furthermore, exploring new business models and revenue streams, such as subscription-based services or advertising, could help to ensure the long-term viability and sustainability of the platform.

CHAPTER 10 REFERENCES

[1] Mohammad Monirujjaman Khan, Nesat Tasneem RoJa, Faris A. Almalki , Maha Aljohani, Revolutionizing E-Commerce Using Blockchain Technology and Implementing Smart Contract, Hindawi, 2022

[2] Teo Min Xuan, Maen T. Alrashdan, Qusay Al-Maatouk, BLOCKCHAIN TECHNOLOGY IN ECOMMERCE PLATFORM, IAEME, 2020.

[3] Xiang Hong Li , Blockchain-based Cross-border E-business Payment Model , ECIT, 2021

[4] Gulshan Kumar, Rahul Saha, William J Buchanan, Decentralized Accessibility of e-commerce Products through Blockchain, Elsevier 2020

[5] Ji Jiang, Jin Chen, Framework of Blockchain-Supported E-Commerce Platform for Small and Medium Enterprises, Sustainability 2021

[6] Samer Shorman, Mohammad Allaymoun, Omer Hamid, DEVELOPING THE E-COMMERCE MODEL A CONSUMER TO CONSUMER USING BLOCKCHAIN NETWORK TECHNIQUE, IJMIT, 2021

[7] Buterin, V. (2013), A next-generation smart contract and decentralized application platform.

[8] Swan, M. (2015), Blockchain: blueprint for a new economy, O'Reilly Media, Inc.

[9] Gerring, J. (2007), Case study research: Principles and practices, Cambridge University Press.

[10] Yin, R. K. (2009), Case study research: Design and methods, Sage publications.

[11] Knezevic, Z., & Tadić, V. (2017), Blockchain technology in e-commerce. Economic research-Ekonomska istraživanja, 30(1), 77-87.

[12] Böhme, R., Christin, N., Edelman, B., & Moore, T. (2015), Bitcoin: Under the hood, Communications of the ACM, 59(2), 104-113.

[13] Mao, Y. (2016), Blockchain: A distributed ledger technology for improved supply chain management, Journal of Business & Economics Research (JBER), 14(4), 131-144.

[14] Wüst, K., & Gervais, A. (2017), Is Bitcoin a decentralized currency? An empirical investigation. International Journal of Information Security, 16(2), 99-115.

SAMPLE CODE

App.js

```
import { useEffect, useState } from 'react'
import { ethers } from 'ethers'
// Components
import Navigation from './components/Navigation'
import Section from './components/Section'
import Product from './components/Product'
// ABIs
import Dappazon from './abis/Dappazon.json'
// Config
import config from './config.json'
function App() {
  const [provider, setProvider] = useState(null)
  const [dappazon, setDappazon] = useState(null)
  const [account, setAccount] = useState(null)
  const [electronics, setElectronics] = useState(null)
  const [clothing, setClothing] = useState(null)
  const [toys, setToys] = useState(null)
  const [item, setItem] = useState({})
  const [toggle, setToggle] = useState(false)
  const togglePop = (item) => {
    setItem(item)
    toggle ? setToggle(false) : setToggle(true)
  }
  const loadBlockchainData = async () => {
  const provider = new ethers.providers.Web3Provider
(window.ethereum)
    setProvider(provider)
    const network = await provider.getNetwork()
    const dappazon = new
ethers.Contract(config[network.chainId].dappazon.address,
Dappazon, provider)
    setDappazon(dappazon)
    const items = []
    for (var i = 0; i < 9; i++) {
      const item = await dappazon.items(i + 1)
      items.push(item)
    }
    const electronics = items.filter((item) => item.category
=== 'electronics')
    const clothing = items.filter((item) => item.category ===
'clothing')
    const toys = items.filter((item) => item.category ===
'toys')
    setElectronics(electronics)
    setClothing(clothing)
    setToys(toys)
  }
  useEffect(() => {
    loadBlockchainData()
  }, [])
  return (
    <div>
      <Navigation account={account} setAccount={setAccount} />
      <h2>Dappazon Best Sellers</h2>
      {electronics && clothing && toys && (
        <>
          <Section title={"Clothing & Jewelry"}
items={clothing} togglePop={togglePop} />
          <Section title={"Electronics & Gadgets"}
items={electronics} togglePop={togglePop} />
          <Section title={"Toys & Gaming"} items={toys}
togglePop={togglePop} />
        </>
      )}
      {toggle && (
        <Product item={item} provider={provider}
account={account} dappazon={dappazon} togglePop={togglePop} />
      )}
    </div>
  );
}
export default App;
```

App.test.js

```
import { render, screen } from '@testing-library/react';
import App from './App';
test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

Config.json

```
{
    "31337": {
        "dappazon": {
            "address":
"0x5FbDB2315678afecb367f032d93F642f64180aa3"
        }
    }
}
```

Index.css

```
@import
url('https://fonts.googleapis.com/css2?family=Lalezar&family=O
pen+Sans:wght@300;400;500;600;700;800&display=swap');
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
:root {
  --clr-white: #FFFFFF;
  --clr-black: #000000;
  --clr-grey: #707070;
  --clr-bunker: #131921;
  --clr-ebony-clay: #232f3e;
  --clr-orange: #FF9900;
  --clr-orange-dark: #cc8111;
}
body {
  margin: 0;
  font-family: "Open Sans";
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
h2 {
  max-width: 1200px;
  margin: 20px auto 0;
  font-size: 2.15em;
}
h2:not(.product__overview h2) {
  padding-left: 20px;
}


/* ---------------------------------------------------- */
/* -- NAVIGATION -- */
nav {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  align-items: center;
  background-color: var(--clr-bunker);
}
.nav__brand {
  display: flex;
  justify-content: center;
  align-items: center;
}
.nav__brand img {
  max-width: 125px;
  height: auto;
}
.nav__brand h1 {
  color: var(--clr-white);
  font-family: "Lalezar";
  font-size: 2.5em;
}
.nav__links {
  background-color: var(--clr-ebony-clay);
  grid-column: 1 / span 3;
  padding: 10px;
  display: flex;
  justify-content: center;
  align-items: center;
  list-style: none;
}
.nav__links li {
  margin: 0 15px;
}
.nav__links li a {
  text-decoration: none;
  color: var(--clr-white);
}
.nav__links li a:hover {
  color: var(--clr-orange);
}
.nav__search {
  padding: 10px;
  border-radius: 6px;
  min-width: 50%;
}
.nav__search:hover {
  outline: 1px solid var(--clr-orange);
}
.nav__connect {
  width: 150px;
  height: 40px;
  margin: 0 auto;
  background-color: var(--clr-orange);
  color: var(--clr-black);
  border: none;
  border-radius: 4px;
  font-family: "Lalezar";
  font-size: 1.25em;
  font-weight: 600;
  cursor: pointer;
  transition: all 250ms ease;
}
```

```
.nav__connect:hover {
  background-color: var(--clr-orange-dark);
}
/* ------------------------------------------------------ */
/* -- CARDS -- */
.cards__section {
  max-width: 1200px;
  margin: 0 auto 50px;
  padding: 0 20px;
}
.cards__section h3 {
  font-size: 1.75em;
  margin: 20px 0;
}
.cards {
  display: grid;
  gap: 10px;
  grid-template-columns: repeat(auto-fit, minmax(min(100%,
350px), 1fr));
  margin-top: 20px;
}
.card {
  width: 300px;
  height: 400px;
  margin: 10px auto;
  cursor: pointer;
  position: relative;
  transition: all 250ms ease;
}
.card:hover {
  box-shadow: 0 0 5px var(--clr-grey);
}
.card__image {
  position: absolute;
  width: 100%;
  z-index: -1;
}
.card__image img {
  max-width: 100%;
}
.card__info {
  width: 100%;
  background-color: var(--clr-white);
  padding: 5px 10px;
  position: absolute;
  bottom: 0;
  left: 0;
}
.card__info h4 {
  font-family: "Open Sans";
  font-size: 1.25em;
  font-weight: 400;
}
.card__info .rating {
  color: yellow;
  margin: 5px 0;
}
.card__info p {
  font-family: "Open Sans";
  font-size: 0.95em;
  font-weight: 600;
  margin-top: 5px;
}
/* ---------------------------------------------------------- */
/* -- PRODUCT -- */
.product {
  width: 100vw;
  height: 100vh;
  background-color: rgba(0, 0, 0, 0.7);
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
.product__details {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(min(100%,
250px), 1fr));
  align-items: center;
  width: 80%;
  height: 80%;
  margin: 0 auto;
  padding: 20px;
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  z-index: 2;
  background: var(--clr-white);
}
.product__image {
  max-width: 500px;
  margin: 20px auto;
}
.product__image img {
  max-width: 100%;
}
.product__close {
  position: absolute;
  top: 10px;
  right: 10px;
  width: 30px;
  height: 30px;
  background: transparent;
  border: none;
  cursor: pointer;
}
.product__close img {
  width: 25px;
  height: 25px;
}
.product__overview {
  height: 90%;
  padding: 20px 40px 20px 20px;
}
.product__overview h2,
.product__overview hr,
.product__overview .rating {
  margin: 10px auto;
}
.product__overview ul {
  margin-left: 40px;
}
.product__order {
  max-width: 300px;
  height: 90%;
  border: 1px solid var(--clr-grey);
  padding: 20px;
}
.product__order h1,
.product__order p {
  margin-bottom: 10px;
}
.product__buy {
  width: 225px;
  height: 40px;
  margin: 20px 0;
  background-color: var(--clr-orange);
  color: var(--clr-black);
  border: none;
  border-radius: 24px;
  font-family: "Open Sans";
  font-size: 0.95em;
  font-weight: 600;
  cursor: pointer;
  transition: all 250ms ease;
}
```

```
.product__buy:hover {
  background-color: var(--clr-orange-dark);
}
.product__bought {
  padding: 5px;
  border: 1px solid var(--clr-grey);
}
@media screen and (max-width: 992px) {
  .product__details {
    height: 95%;
    width: 90%;
  }
  .product__order {
    max-width: 100%;
    height: 100%;
    margin-top: 20px;
    grid-column: span 2;
  }
}
@media screen and (max-width: 768px) {
  nav {
    grid-template-columns: repeat(2, 1fr);
  }
  .nav__links {
    display: none;
  }
  .nav__search {
    display: none;
  }
}
@media screen and (max-width: 576px) {
  .product__details {
    height: 95%;
    overflow-y: scroll;
  }
  .product__image {
    grid-column: span 2;
  }
}
```

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.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
  <meta name="viewport" content="width=device-width, initial-
scale=1" />
  <meta name="theme-color" content="#000000" />
  <meta name="description" content="Web site created using
create-react-app" />
  <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png"
/>
  <!--
      manifest.json provides metadata used when your web app
is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-
manifest/
    -->
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
  <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder
during the build.
      Only files inside the `public` folder can be referenced
from the HTML.
      Unlike "/favicon.ico" or "favicon.ico",
"%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-
root public URL.
      Learn how to configure a non-root public URL by running
`npm run build`.
    -->
  <title>Dappazon</title>
</head>
```

```
<body>
  <noscript>You need to enable JavaScript to run this
app.</noscript>
  <div id="root"></div>
  <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an
empty page.
      You can add webfonts, meta tags, or analytics to this
file.
      The build step will place the bundled scripts into the
<body> tag.
      To begin the development, run `npm start` or `yarn
start`.
      To create a production bundle, use `npm run build` or
`yarn build`.
    -->
</body>
</html>
```

Items.json

```
{
  "items": [
    {
      "id": 1,
      "name": "Camera",
      "category": "electronics",
      "image":
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/camera.jpg",
      "price": "1",
      "rating": 4,
      "stock": 10
    },
    {
      "id": 2,
      "name": "Drone",
      "category": "electronics",
      "image":
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/drone.jpg",
      "price": "2",
      "rating": 5,
      "stock": 6
    },
    {
      "id": 3,
      "name": "Headset",
      "category": "electronics",
      "image":
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/headset.jpg",
      "price": "0.25",
      "rating": 2,
      "stock": 24
    },
    {
      "id": 4,
      "name": "Shoes",
      "category": "clothing",
      "image":
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/shoes.jpg",
      "price": "0.25",
      "rating": 5,
      "stock": 3
    },
    {
      "id": 5,
      "name": "Sunglasses",
      "category": "clothing",
      "image":
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/sunglasses.jpg",
      "price": "0.10",
      "rating": 4,
      "stock": 12
    },
    {
      "id": 6,
      "name": "Watch",
      "category": "clothing",
      "image":
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/watch.jpg",
      "price": "1.25",
      "rating": 4,
      "stock": 0
    },
    {
      "id": 7,
      "name": "Puzzle Cube",
      "category": "toys",
      "image":
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/cube.jpg",
      "price": "0.05",
      "rating": 4,
      "stock": 15
    },
    {
      "id": 8,
      "name": "Train Set",
      "category": "toys",
      "image":
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/train.jpg",
      "price": "0.20",
      "rating": 4,
      "stock": 0
    },
    {
      "id": 9,
      "name": "Robot Set",
      "category": "toys",
      "image":
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/robots.jpg",
      "price": "0.15",
      "rating": 3,
      "stock": 12
    }
  ]
}
```

Dappazon.json

```
[
    {
        "inputs": [],
        "stateMutability": "nonpayable",
        "type": "constructor"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": false,
                "internalType": "address",
                "name": "buyer",
                "type": "address"
            },
            {
                "indexed": false,
                "internalType": "uint256",
                "name": "orderId",
                "type": "uint256"
            },
            {
                "indexed": false,
                "internalType": "uint256",
                "name": "itemId",
                "type": "uint256"
            }
        ],
        "name": "Buy",
        "type": "event"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": false,
                "internalType": "string",
                "name": "name",
                "type": "string"
            },
            {
                "indexed": false,
                "internalType": "uint256",
                "name": "cost",
                "type": "uint256"
            },
            {
                "indexed": false,
                "internalType": "uint256",
                "name": "quantity",
                "type": "uint256"
            }
        ],
        "name": "List",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_id",
                "type": "uint256"
            }
        ],
        "name": "buy",
        "outputs": [],
        "stateMutability": "payable",
        "type": "function"
        },
        {
            "inputs": [
                {
                    "internalType": "uint256",
                    "name": "",
                    "type": "uint256"
                }
            ],
            "name": "items",
            "outputs": [
                {
                    "internalType": "uint256",
                    "name": "id",
                    "type": "uint256"
                },
                {
                    "internalType": "string",
                    "name": "name",
                    "type": "string"
                },
                {
                    "internalType": "string",
                    "name": "category",
                    "type": "string"
                },
                {
                    "internalType": "string",
                    "name": "image",
                    "type": "string"
                },
                {
                    "internalType": "uint256",
                    "name": "cost",
                    "type": "uint256"
                },
                {
                    "internalType": "uint256",
                    "name": "rating",
                    "type": "uint256"
                },
                {
                    "internalType": "uint256",
                    "name": "stock",
                    "type": "uint256"
                }
            ],
            "stateMutability": "view",
            "type": "function"
        },
        {
            "inputs": [
                {
                    "internalType": "uint256",
                    "name": "_id",
                    "type": "uint256"
                },
                {
                    "internalType": "string",
                    "name": "_name",
                    "type": "string"
                },
                {
                    "internalType": "string",
                    "name": "_category",
                    "type": "string"
                },
                {
                    "internalType": "string",
                    "name": "_image",
                    "type": "string"
                },
                {
                    "internalType": "uint256",
                    "name": "_cost",
                    "type": "uint256"
                },
                {
                    "internalType": "uint256",
                    "name": "_rating",
                    "type": "uint256"
                },
                {
                    "internalType": "uint256",
                    "name": "_stock",
                    "type": "uint256"
                }
            ],
            "name": "list",
            "outputs": [],
            "stateMutability": "nonpayable",
            "type": "function"
        },
        {
            "inputs": [
                {
                    "internalType": "address",
                    "name": "",
                    "type": "address"
                }
            ],
            "name": "orderCount",
            "outputs": [
                {
                    "internalType": "uint256",
                    "name": "",
                    "type": "uint256"
                }
            ],
            "stateMutability": "view",
            "type": "function"
        },
        {
            "inputs": [
                {
                    "internalType": "address",
                    "name": "",
                    "type": "address"
                },
                {
                    "internalType": "uint256",
                    "name": "",
                    "type": "uint256"
                }
            ],
            "name": "orders",
            "outputs": [
                {
                    "internalType": "uint256",
                    "name": "time",
                    "type": "uint256"
                },
                {
                    "components": [
                        {
                            "internalType": "uint256",
                            "name": "id",
                            "type": "uint256"
                        },
                        {
                            "internalType": "string",
                            "name": "name",
                            "type": "string"
                        },
                        {
                            "internalType": "string",
                            "name": "category",
                            "type": "string"
                        },
                        {
                            "internalType": "string",
                            "name": "image",
                            "type": "string"
                        },
                        {
                            "internalType": "uint256",
                            "name": "cost",
                            "type": "uint256"
                        },
                        {
                            "internalType": "uint256",
                            "name": "rating",
                            "type": "uint256"
                        },
                        {
                            "internalType": "uint256",
                            "name": "stock",
                            "type": "uint256"
                        }
                    ],
                    "internalType": "struct Dappazon.Item",
                    "name": "item",
                    "type": "tuple"
                }
            ],
            "stateMutability": "view",
            "type": "function"
        },
        {
            "inputs": [],
            "name": "owner",
            "outputs": [
                {
                    "internalType": "address",
                    "name": "",
                    "type": "address"
                }
            ],
            "stateMutability": "view",
            "type": "function"
        },
        {
            "inputs": [],
            "name": "withdraw",
            "outputs": [],
            "stateMutability": "nonpayable",
            "type": "function"
        }
    ]
```

Deploy.js

```
// We require the Hardhat Runtime Environment explicitly here.
This is optional
// but useful for running the script in a standalone fashion
through `node <script>`.
// You can also run a script with `npx hardhat run <script>`.
If you do that, Hardhat
// will compile your contracts, add the Hardhat Runtime
Environment's members to the
// global scope, and execute the script.
const hre = require("hardhat");
const { items } = require("../src/items.json");
const tokens = (n) => {
  return ethers.utils.parseUnits(n.toString(), "ether");
};
async function main() {
  // Setup accounts
  const [deployer] = await ethers.getSigners();
  // Deploy Dappazon
  const Dappazon = await
hre.ethers.getContractFactory("Dappazon");
  const dappazon = await Dappazon.deploy();
  await dappazon.deployed();
  console.log(`Deployed Dappazon Contract at:
${dappazon.address}\n`);
  // Listing items...
  for (let i = 0; i < items.length; i++) {
    const transaction = await dappazon
      .connect(deployer)
      .list(
        items[i].id,
        items[i].name,
        items[i].category,
        items[i].image,
        tokens(items[i].price),
        items[i].rating,
        items[i].stock
      );
    await transaction.wait();
    console.log(`Listed item ${items[i].id}:
${items[i].name}`);
```

```
  }
}
// We recommend this pattern to be able to use async/await
everywhere
// and properly handle errors.
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

Navigation.js

```
import { ethers } from 'ethers'
const Navigation = ({ account, setAccount }) => {
    const connectHandler = async () => {
        const accounts = await window.ethereum.request({
method: 'eth_requestAccounts' });
        const account = ethers.utils.getAddress(accounts[0])
        setAccount(account);
    }
    return (
        <nav>
            <div className='nav__brand'>
                <h1>Dappazon</h1>
            </div>
            <input
                type="text"
                className="nav__search"
            />
            {account ? (
                <button
                    type="button"
                    className='nav__connect'
                >
                    {account.slice(0, 6) + '...' +
account.slice(38, 42)}
                </button>
            ) : (
                <button
                    type="button"
                    className='nav__connect'
                    onClick={connectHandler}
                >
                    Connect
                </button>
            )}
            <ul className='nav__links'>
                <li><a href="#Clothing & Jewelry">Clothing &
Jewelry</a></li>
```

```
                <li><a href="#Electronics &
Gadgets">Electronics & Gadgets</a></li>
                <li><a href="#Toys & Gaming">Toys &
Gaming</a></li>
            </ul>
        </nav>
    );
}
export default Navigation;
```

Product.js

```
import { useEffect, useState } from 'react'
import { ethers } from 'ethers'
// Components
import Rating from './Rating'
import close from '../assets/close.svg'
const Product = ({ item, provider, account, dappazon,
togglePop }) => {
  const [order, setOrder] = useState(null)
  const [hasBought, setHasBought] = useState(false)
  const fetchDetails = async () => {
    const events = await dappazon.queryFilter("Buy")
    const orders = events.filter(
      (event) => event.args.buyer === account &&
event.args.itemId.toString() === item.id.toString()
    )
    if (orders.length === 0) return
    const order = await dappazon.orders(account,
orders[0].args.orderId)
    setOrder(order)
  }
  const buyHandler = async () => {
    const signer = await provider.getSigner()
    // Buy item...
    let transaction = await
dappazon.connect(signer).buy(item.id, { value: item.cost })
    await transaction.wait()
    setHasBought(true)
  }
  useEffect(() => {
    fetchDetails()
  }, [hasBought])
  return (
    <div className="product">
      <div className="product__details">
        <div className="product__image">
          <img src={item.image} alt="Product" />
        </div>
```

```
          <div className="product__overview">
            <h1>{item.name}</h1>
            <Rating value={item.rating} />
            <hr />
            <p>{item.address}</p>
            <h2>{ethers.utils.formatUnits(item.cost.toString(),
'ether')} ETH</h2>
            <hr />
            <h2>Overview</h2>
            <p>
              {item.description}
              Lorem ipsum dolor sit amet consectetur adipisicing
elit. Minima rem, iusto,
              consectetur inventore quod soluta quos qui
assumenda aperiam, eveniet doloribus
              commodi error modi eaque! Iure repudiandae
temporibus ex? Optio!
            </p>
          </div>
          <div className="product__order">
            <h1>{ethers.utils.formatUnits(item.cost.toString(),
'ether')} ETH</h1>
            <p>
              FREE delivery <br />
              <strong>
                {new Date(Date.now() +
345600000).toLocaleDateString(undefined, { weekday: 'long',
month: 'long', day: 'numeric' })}
              </strong>
            </p>
            {item.stock > 0 ? (
              <p>In Stock.</p>
            ) : (
              <p>Out of Stock.</p>
            )}
            <button className='product__buy'
onClick={buyHandler}>
              Buy Now
            </button>
            <p><small>Ships from</small> Dappazon</p>
            <p><small>Sold by</small> Dappazon</p>
            {order && (
              <div className='product__bought'>
                Item bought on <br />
                <strong>
                  {new Date(Number(order.time.toString() +
'000')).toLocaleDateString(
                    undefined,
                    {
```

```
                    weekday: 'long',
                    hour: 'numeric',
                    minute: 'numeric',
                    second: 'numeric'
                  })}
              </strong>
            </div>
          )}
        </div>
        <button onClick={togglePop}
className="product__close">
          <img src={close} alt="Close" />
        </button>
      </div>
    </div >
  );
}
export default Product;
```

Rating.js

```
import star_regular from '../assets/star-regular.svg';
import star_solid from '../assets/star-solid.svg';
const Rating = ({ value }) => {
    return (
        <div className='rating'>
            <img src={value >= 1 ? star_solid : star_regular}
width="20px" height="20px" alt="Star" />
            <img src={value >= 2 ? star_solid : star_regular}
width="20px" height="20px" alt="Star" />
            <img src={value >= 3 ? star_solid : star_regular}
width="20px" height="20px" alt="Star" />
            <img src={value >= 4 ? star_solid : star_regular}
width="20px" height="20px" alt="Star" />
            <img src={value >= 5 ? star_solid : star_regular}
width="20px" height="20px" alt="Star" />
        </div>
    );
}
export default Rating;
```

Section.js

```
import { ethers } from 'ethers'
// Components
import Rating from './Rating'
const Section = ({ title, items, togglePop }) => {
    return (
        <div className='cards__section'>
```

```
            <h3 id={title}>{title}</h3>
            <hr />
            <div className='cards'>
                {items.map((item, index) => (
                    <div className='card' key={index}
onClick={() => togglePop(item)}>
                        <div className='card__image'>
                            <img src={item.image} alt="Item"
/>
                        </div>
                        <div className='card__info'>
                            <h4>{item.name}</h4>
                            <Rating value={item.rating} />

<p>{ethers.utils.formatUnits(item.cost.toString(), 'ether')}
ETH</p>
                        </div>
                    </div>
                ))}
            </div>
        </div>
    );
}
export default Section;
```

Dappazon.js

```
const { expect } = require("chai");
const tokens = (n) => {
  return ethers.utils.parseUnits(n.toString(), "ether");
};
const ID = 1;
const NAME = "Shoes";
const CATEGORY = "Clothing";
const IMAGE =
"https://ipfs.io/ipfs/QmTYEboq8raiBs7GTUg2yLXB3PMz6HuBNgNfSZBx
5Msztg/shoes.jpg";
const COST = tokens(1);
const RATING = 4;
const STOCK = 5;
describe("Dappazon", () => {
  let dappazon;
  let deployer, buyer;
  beforeEach(async () => {
    // Set up accounts
    [deployer, buyer] = await ethers.getSigners();
    // Deploy Contract
    const Dappazon = await
ethers.getContractFactory("Dappazon");
```

```
    dappazon = await Dappazon.deploy();
  });
  describe("Deployment", () => {
    it("Sets the owner", async () => {
      expect(await
dappazon.owner()).to.equal(deployer.address);
    });
  });
  describe("Listing", () => {
    let transaction;
    beforeEach(async () => {
      transaction = await dappazon
        .connect(deployer)
        .list(ID, NAME, CATEGORY, IMAGE, COST, RATING, STOCK);
      await transaction.wait();
    });
    it("Return Item attributes", async () => {
      const item = await dappazon.items(ID);
      expect(item.id).to.equal(ID);
      expect(item.name).to.equal(NAME);
      expect(item.category).to.equal(CATEGORY);
      expect(item.image).to.equal(IMAGE);
      expect(item.cost).to.equal(COST);
      expect(item.rating).to.equal(RATING);
      expect(item.stock).to.equal(STOCK);
    });
    it("Emits List Event", () => {
      expect(transaction).to.emit(dappazon, "List");
    });
  });
  describe("Buying", () => {
    let transaction;
    beforeEach(async () => {
      // List an item
      transaction = await dappazon
        .connect(deployer)
        .list(ID, NAME, CATEGORY, IMAGE, COST, RATING, STOCK);
      await transaction.wait();
      // Buy an item
      transaction = await dappazon.connect(buyer).buy(ID, {
value: COST });
    });
    it("Updates buyer's order count", async () => {
      const result = await dappazon.orderCount(buyer.address);
      expect(result).to.equal(1);
    });
    it("Adds the order", async () => {
      const order = await dappazon.orders(buyer.address, 1);
      expect(order.time).to.greaterThan(0);
      expect(order.item.name).to.equal(NAME);
    });
    it("Updates the contract balance", async () => {
      const result = await
ethers.provider.getBalance(dappazon.address);
      expect(result).to.equal(COST);
    });
    it("Emits Buy Event", () => {
      expect(transaction).to.emit(dappazon, "Buy");
    });
  });
  describe("Withdrawing", () => {
    let balanceBefore;
    beforeEach(async () => {
      // List a item
      let transaction = await dappazon
        .connect(deployer)
        .list(ID, NAME, CATEGORY, IMAGE, COST, RATING, STOCK);
      await transaction.wait();
      // Buy a item
      transaction = await dappazon.connect(buyer).buy(ID, {
value: COST });
      await transaction.wait();
      // Get Deployer balance before
      balanceBefore = await
ethers.provider.getBalance(deployer.address);
      // Withdraw
      transaction = await
dappazon.connect(deployer).withdraw();
      await transaction.wait();
    });
    it("Updates the owner balance", async () => {
      const balanceAfter = await
ethers.provider.getBalance(deployer.address);
      expect(balanceAfter).to.be.greaterThan(balanceBefore);
    });
    it("Updates the contract balance", async () => {
      const result = await
ethers.provider.getBalance(dappazon.address);
      expect(result).to.equal(0);
    });
  });
});
```

Dappazon.sol

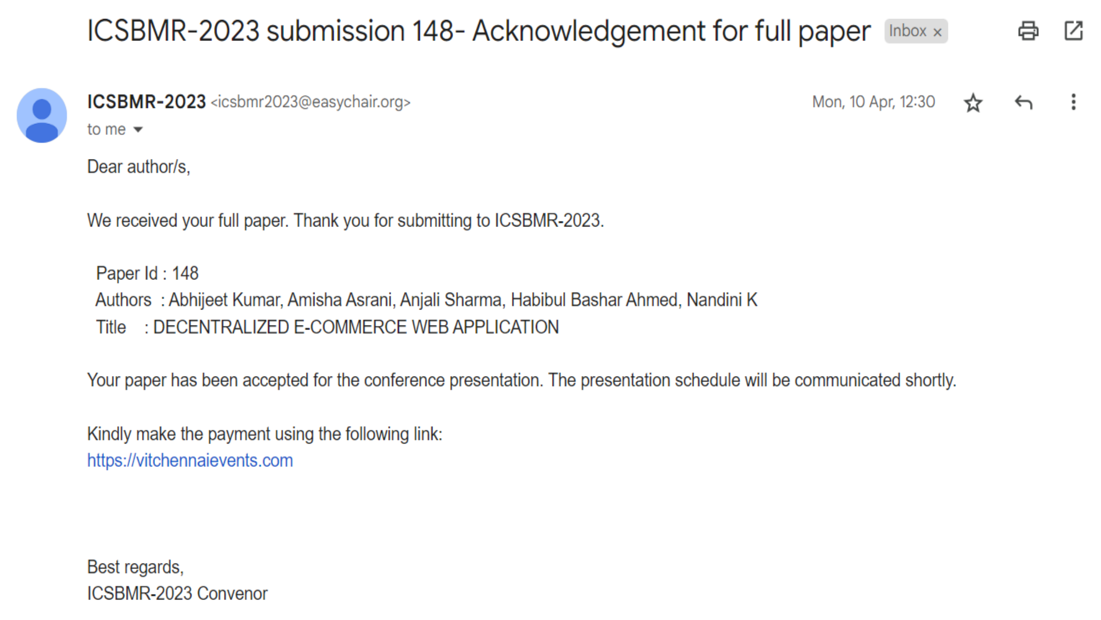
```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.9;
contract Dappazon {
    address public owner;
    // Defines a local DS for the products
    struct Item {
        uint256 id;
        string name;
        string category;
        string image;
        uint256 cost;
        uint256 rating;
        uint256 stock;
    }
    // Defines order
    struct Order {
        uint256 time;
        Item item;
    }
    // Mapping each item to unique key
    mapping(uint256 => Item) public items;
    mapping(address => uint256) public orderCount;
    mapping(address => mapping(uint256 => Order)) public
orders;
    event List(string name, uint256 cost, uint256 quantity);
    event Buy(address buyer, uint256 orderId, uint256 itemId);
    modifier onlyOwner() {
        //  Restrict listing to the owner
        require(msg.sender == owner);
        _;
    }
    constructor() {
        // Person who is sending the contract
        owner = msg.sender;
    }
    // List products
    function list(
        uint256 _id,
        string memory _name,
        string memory _category,
        string memory _image,
        uint256 _cost,
        uint256 _rating,
        uint256 _stock
    ) public onlyOwner {
        //  Create Item Struct
        Item memory item = Item(
            _id,
            _name,
            _category,
            _image,
            _cost,
            _rating,
            _stock
        );
        // Save Item Struct
        items[_id] = item;
        // Emit Event on BLOCKCHAIN
        emit List(_name, _cost, _stock);
    }
    // Buy Products
    function buy(uint256 _id) public payable {
        // Receive Crypto: Payable
        // Fetch item
        Item memory item = items[_id];
        // Require enough ether to buy the item
        require(msg.value >= item.cost);
        // Require item is in stock
        require(item.stock > 0);
        // Create an order
        Order memory order = Order(block.timestamp, item);
        // Add order for user
        orderCount[msg.sender]++;
        orders[msg.sender][orderCount[msg.sender]] = order;
        // Substract Stock
        items[_id].stock = item.stock - 1;
        // Emit event
        emit Buy(msg.sender, orderCount[msg.sender], item.id);
    }
    // Withdraw Funds
    function withdraw() public onlyOwner {
        (bool success, ) = owner.call{value: 
address(this).balance}("");
        require(success);
    }
}
```

Hardhat.config.js

```
require("@nomicfoundation/hardhat-toolbox");
/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.17",
};
```

FUNDING AND PUBLISHED PAPER DETAILS

Attached below is the screenshot of the confirmation mail we have received from the conference for our paper entitled ‘Decentralized E-commerce Web Application’. The ICSBMR-2023: International Conference on Emerging Trends in Social, Business and Management Science Research is going to be held at VIT Business School, Vellore Institute of Technology, Chennai on May 8, 2023. The paper is authored by Abhijeet Kumar, Amisha Asrani, Anjali Sharma, and Habibul Bashar Ahmed under the constant guidance of Prof. Nandini K. The research paper was accepted on April 10, 2023 for publication. The paper was selected for publication following a rigorous peer-review process and was chosen for presentation at the conference due to the novelty of the proposed approach and the potential impact on the e-commerce industry. This confirmation certificate serves as evidence of the quality and relevance of the research presented in this project report.



ICSBMR-2023 submission 148- Acknowledgement for full paper Inbox ×

ICSBMR-2023 <icsbmr2023@easychair.org> Mon, 10 Apr, 12:30
to me

Dear author/s,

We received your full paper. Thank you for submitting to ICSBMR-2023.

Paper Id : 148
Authors : Abhijeet Kumar, Amisha Asrani, Anjali Sharma, Habibul Bashar Ahmed, Nandini K
Title : DECENTRALIZED E-COMMERCE WEB APPLICATION

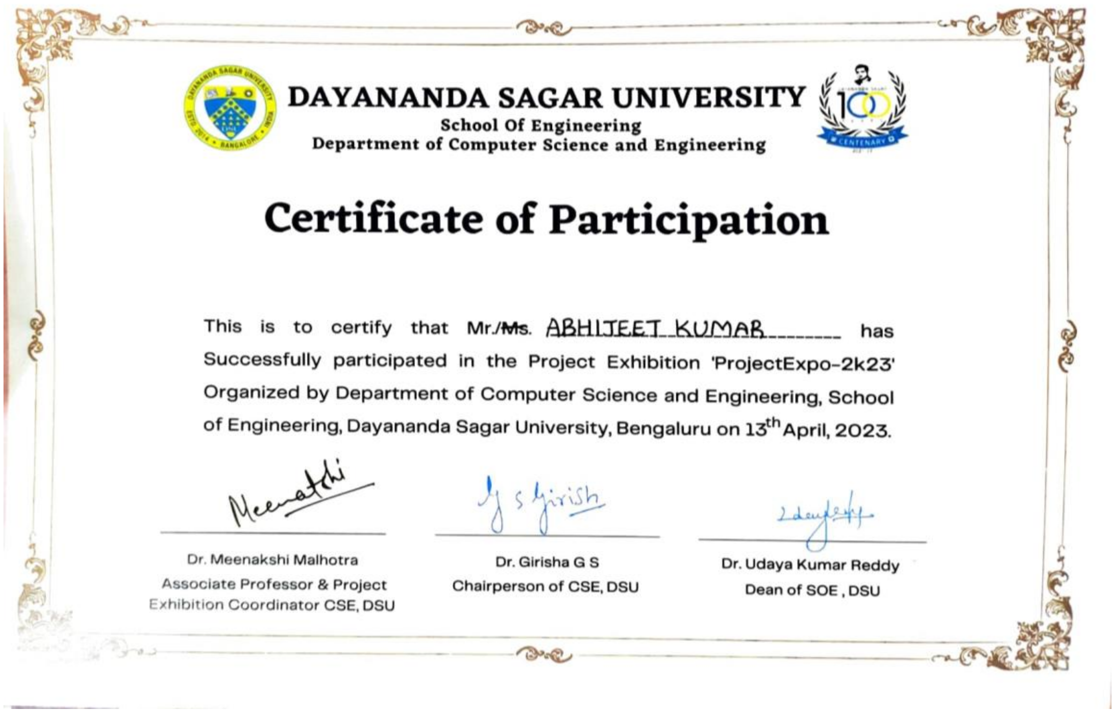
Your paper has been accepted for the conference presentation. The presentation schedule will be communicated shortly.

Kindly make the payment using the following link:
https://vitchennaievents.com

Best regards,
ICSBMR-2023 Convenor

PROJECT EXHIBITION:

Our team had been selected to present our project at the project exhibition 'ProjectExpo-2k23' organized by the Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University on 13th April 2023. Out of a total of 100 projects submitted, only a few were shortlisted, and we were honored to be among them. To top it off, our team was delighted to learn that our project had been ranked among the top 10 projects presented at the exhibition. This recognition is a testament to the hard work and dedication put in by our team in developing an innovative and impactful project. We are grateful for the opportunity to showcase our project and share it with others, and we look forward to continuing to make a positive impact in our field.


DAYANANDA SAGAR UNIVERSITY
School Of Engineering
Department of Computer Science and Engineering

Certificate of Participation

This is to certify that Mr./~~Ms.~~ ABHIJEET KUMAR has Successfully participated in the Project Exhibition 'ProjectExpo-2k23' Organized by Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bengaluru on 13th April, 2023.

Meenakshi	G S Girish	[illegible]
Dr. Meenakshi Malhotra Associate Professor & Project Exhibition Coordinator CSE, DSU	Dr. Girisha G S Chairperson of CSE, DSU	Dr. Udaya Kumar Reddy Dean of SOE , DSU

GITHUB LINK:

https://github.com/CSE-DSU/Team_55_Decentralized_E-Commerce_Web_Application