CLOUD BASED ANTIQUE AUCTIONEER USING BLOCKCHAIN

A PROJECT REPORT

Submitted by

VARUN RAJ V R - 210419104177

SAKTHI KUMAR S - 210419104166

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

CHENNAI INSTITUTE OF TECHNOLOGY, CHENNAI -600069

ANNA UNIVERSITY: CHENNAI 600025

JUNE 2022

ANNA UNIVERSITY: CHENNAI – 600025

JUNE 2022

BONAFIDE CERTIFICATE

Certified that this project report **"CLOUD BASED ANTIQUE AUCTIONEERUSING BLOCKCHAIN"** is the bonafide work of **"VARUN RAJ V R (210419104177) AND SAKTHI KUMAR S (210419104166)"** who carried out the project work under my supervision.

SIGNATURE
Dr. S. PAVITHRA, M.E Ph.D.

Associate professor
Head of the Department.

Department of Computer Science
And Engineering.
Chennai Institute of Technology.
Kundrathur, Chennai- 600 069.

SIGNATURE
Dr. S KANIMOZHI, M.E Ph.D.

Associate Professor.

Department of Computer Science
and Engineering.
Chennai Institute of Technology.
Kundrathur, Chennai- 600 069.

Submitted for the project viva examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our gratitude to our chairman **Shri. P. SRIRAM**, and all trust members of Chennai Institute of Technology for providing the facility and opportunity to do this project as a part of our undergraduate course.

We thank our Principal **Dr. A. RAMESH M.E., Ph.D.** for his valuable suggestion and guidance for the development and completion of this project

We sincerely thank our Head of the Department, **Dr. S. PAVITHRA M.E., Ph.D.** Department of Computer Science Engineering for having provided us with valuable guidance, resources and timely suggestions through our work.

We sincerely thank our project guide **Dr. S. KANIMOZHI, M.E., Ph.D.** Associate Professor, Department of Computer Science Engineering for having provided us with valuable guidance, resources and timely suggestions through our work.

We wish to extend our sincere thanks to **All Faculty members** and **Lab Instructors** of the Department of Computer Science Engineering for their valuable suggestions and their kind cooperation for the successful completion of our project.

ABSTRACT

Nowadays, anyone can sell anything on the internet. This is the era of internet and e-commerce is a huge pinnacle of this evolvement. This raises the question of integrity of the products and security of the customer. Selling art and collectibles, have always remained a huge challenge nevertheless the internet and e-commerce. There is always a challenge of finding the authenticity of the arts whilst ensuring the creative standards. This pandemic has thrown many artists out of work because of lack of mode of sale.

Also, the artists take a huge toll when they are ripped off and duplicate copies are sold. This reduces the motivation they have for the art. It is high time to expedite this.

We have developed a web application that can solve this very problem by means of auctioning the art. the customers are made happy because better quality assurance standards are met.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

S. NO	ABBREVATION	EXPANSION
1	AWS	Amazon Web Services
2	AWSN	Amazon web services network
3	SVM	Support Vector Machine
4	CLI	Command line interface
5	UI	User Interface
6	S3	Storage bin 3
7	EC2	Elastic computer 2
8	AVAX	Avalanche
9	Deft	Direct electric fund transfer
10	KYC	Know your customer
11	NFT	Non- fungible token
12	DApp	Decentralized application

CHAPTER 1

INTRODUCTION

1.1 SCOPE OF THE PROJECT

In Today's world, anybody can sell products over the internet. Unfortunately, counterfeit news gathers a lot of consideration over the web, particularly via web-based commerce media. Individuals get misdirected and don't reconsider before falling for such mis-educational pieces to the most distant part of the arrangement. Such type of activities is not good for the society as art is an important part of our culture and history. As fast as the technology is moving, on the same pace the preventive measures are required to deal with such activities. Arts have been an integral part of showcasing history, representing cultures of various cultures. Arts have been a useful component for anthropologists to understand how people lived in various eras. Recently contemporary arts have been used to represent people's emotions and have proven useful in many campaigns.

Such important arts have now become an object of possession and considered as collectibles. Commerce of these artworks and collectibles have become a problem these days. There are a lot of possibilities for customers to get hoodwinked. The authenticity and ownership issues are a huge problem that are faced by the customers. The sellers face issues like lack of options to sell. They also face a lot of issues with protecting the integrity of their artwork. Plagiarism is very frowned upon in the art community, yet, there are very little ways to report such an event of fraudulence. This is high time to expedite this raising issue.

Digital Currency is a term that refers to a specific type of electronic currency with specific properties. Digital Currency is also a term used to include the meta-group of

sub-types of digital currency, the specific meaning can only be determined within the specific legal or contextual case.

Legally and technically, there already are a myriad of legal definitions of digital currency and the many digital currency sub-types. Combining different possible properties, there exists an extensive number of implementations creating many and numerous sub-types of Digital Currency. Many governmental jurisdictions have implemented their own unique definition for digital currency, virtual currency, cryptocurrency, e-money, network money, e-cash, and other types of digital currency. Within any specific government jurisdiction, different agencies and regulators define different and often conflicting meanings for the different types of digital currency based on the specific properties of a specific currency type or sub-type.

A virtual currency has been defined in 2012 by the European Central Bank as "a type of unregulated, digital money, which is issued and usually controlled by its developers, and used and accepted among the members of a specific virtual community". The US Department of Treasury in 2013 defined it more tersely as "a medium of exchange that operates like a currency in some environments, but does not have all the attributes of real currency". The US Department of Treasury also stated that, "Virtual currency does not have legal-tender status in any jurisdiction."

According to the European Central Bank's 2015 "Virtual currency schemes – a further analysis" report, virtual currency is a digital representation of value, not issued by a central bank, credit institution or e-money institution, which, in some circumstances, can be used as an alternative to money. In the previous report of October 2012, the virtual currency was defined as a type of unregulated, digital money, which is issued and usually controlled by its developers, and used.

According to the Bank for International Settlements' November 2015 "Digital currencies" report, it is an asset represented in digital form and having some monetary characteristics. Digital currency can be denominated to a sovereign currency and issued by the issuer responsible to redeem digital money for cash. In that case, digital currency represents electronic money (e-money). Digital currency denominated in its own units of value or with decentralized or automatic issuance will be considered as a virtual currency. As such, bitcoin is a digital currency but also a type of virtual currency. Bitcoin and its alternatives are based on cryptographic algorithms, so these kinds of virtual currencies are also called cryptocurrencies.

Cryptocurrency is a sub-type of digital currency and a digital asset that relies on cryptography to chain together digital signatures of asset transfers, peer-to-peer networking and decentralization. In some cases, a proof-of-work or proof-of-stake scheme is used to create and manage the currency. Cryptocurrencies can allow electronic money systems to be decentralized. When implemented with a blockchain, the digital ledger system or record keeping system uses cryptography to edit separate shards of database entries that are distributed across many separate servers. The first and most popular system is bitcoin, a peer-to-peer electronic monetary system based on cryptography.

1.2 OBJECTIVE OF THE PROJECT

Selling art and collectibles, have always remained a huge challenge nevertheless the internet and e-commerce. There is always a challenge of finding the authenticity of the arts whilst ensuring the creative standards. We have developed a web app that can solve this very problem by means of auctioning the art. the customers are made happy because better quality assurance standards are met.

1.3 REPORT SUMMARY

The project report is organized as follows: the chapter 2 narrates the related work done in art auctioning, chapter 3 explains the fundamentals and methodologies, chapter 4 depicts the architecture and design of the proposed methodology, chapter 5 discuss the system specification and requirements, chapter 6 describes the implementation and discussion for auctioning web app, the chapter 7 details the conclusion and future work of this model, and the last section provides the references and appendix.

CHAPTER 2

RELATED WORK

2.1 INTRODUCTION

The various research works on the existing art auctioning model using blockchain.

2.2 Artfare

Artfare puts a new spin on the art fair model, using a mobile app, in-house curators and local pop-up shows and fairs to create connections between artists and collectors in local art scenes, starting with New York. Artfare promotes vibrant local art scenes by bringing artists and collectors together through messaging, studio visits and sales of art works.

2.3 Paddle8

Paddle 8 is a curated auction platform that enables a global community of buyers to discover and bid for art in real time. By digitally enabling the auction mechanic, it helps to promote artists' work and drive sales, creating a global, digital spin on a market dynamic that was previously often limited to those with privilege and access.

CHAPTER 3

RELATED ARTICLES

3.1 Art and commerce

It's always been tough to make money from art. But for big-city artists and curators, the cost of real estate for galleries and studio space has made it even tougher to build a sustainable practice.

The growing popularity of art fairs has also had an effect, by creating destination experiences that might be great for bringing people in, but not necessarily for actually selling art.

But artists willing to embrace digital technology and take charge of their own destinies have never had it so good. Social media, online platforms, transactional ecommerce platforms and mobile apps have made it possible to reach global audiences, build personal brands, and sell art on more favourable terms than ever before.

3.2 Blockchain

A blockchain is a distributed database that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format. Blockchains are best known for their crucial role in cryptocurrency systems, such as Bitcoin, for maintaining a secure and decentralized record of transactions. The innovation with a blockchain is that it guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party.

A blockchain is a growing list of records, called blocks, that are securely linked together using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree, where data

nodes are represented by leafs). The timestamp proves that the transaction data existed when the block was published to get into its hash. As blocks each contain information about the block previous to it, they form a chain, with each additional block reinforcing the ones before it. Therefore, blockchains are resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks.

Blockchains are typically managed by a peer-to-peer network for use as a publicly distributed ledger, where nodes collectively adhere to a protocol to communicate and validate new blocks. Although blockchain records are not unalterable as forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.

The blockchain was popularized by a person (or group of people) using the name Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the cryptocurrency bitcoin, based on work by Stuart Haber, W. Scott Stornetta, and Dave Bayer. The identity of Satoshi Nakamoto remains unknown to date. The implementation of the blockchain within bitcoin made it the first digital currency to solve the double-spending problem without the need of a trusted authority or central server. The bitcoin design has inspired other applications and blockchains that are readable by the public and are widely used by cryptocurrencies. The blockchain is considered a type of payment rail.

Private blockchains have been proposed for business use. Computerworld called the marketing of such privatized blockchains without a proper security model "snake oil"; however, others have argued that permissioned blockchains, if carefully designed, may be more decentralized and therefore more secure in practice than permissionless ones.

3.3 Types of Blockchain

3.3.1 Public blockchain

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.

3.3.2 Private blockchain

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.

3.3.3 Hybrid blockchain

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 and decentralization are used.

3.3.4 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.).

3.4 React.JS

React is a popular JavaScript library used for web development. React.js or ReactJS or React are different ways to represent ReactJS. Today's many large-scale companies (Netflix, Instagram, to name a few) also use React JS. There are many advantages of using this framework over other frameworks, and It's ranking under the top 10 programming languages for the last few years under various language ranking indices.

3.5 Metamask

MetaMask is a cryptocurrency wallet that can be downloaded on Chrome and Firefox as a browser extension or as an App on iOS and Android devices. It allows users to store and transact crypto through a simple interface.

MetaMask enables users to interact with DeFi ecosystems on Ethereum and other blockchain networks, such as Binance Smart Chain, Polygon, and Avalanche. This guide will teach you how to connect your MetaMask to the Avalanche network.

If you are already using the Avalanche network, note that only the C-Chain wallet is compatible with MetaMask. So to transfer AVAX from your Avalanche wallet, your tokens need to be in your C-Chain wallet. You can also transfer your AVAX to MetaMask from an exchange wallet that has integrated C-Chain, such as Binance.

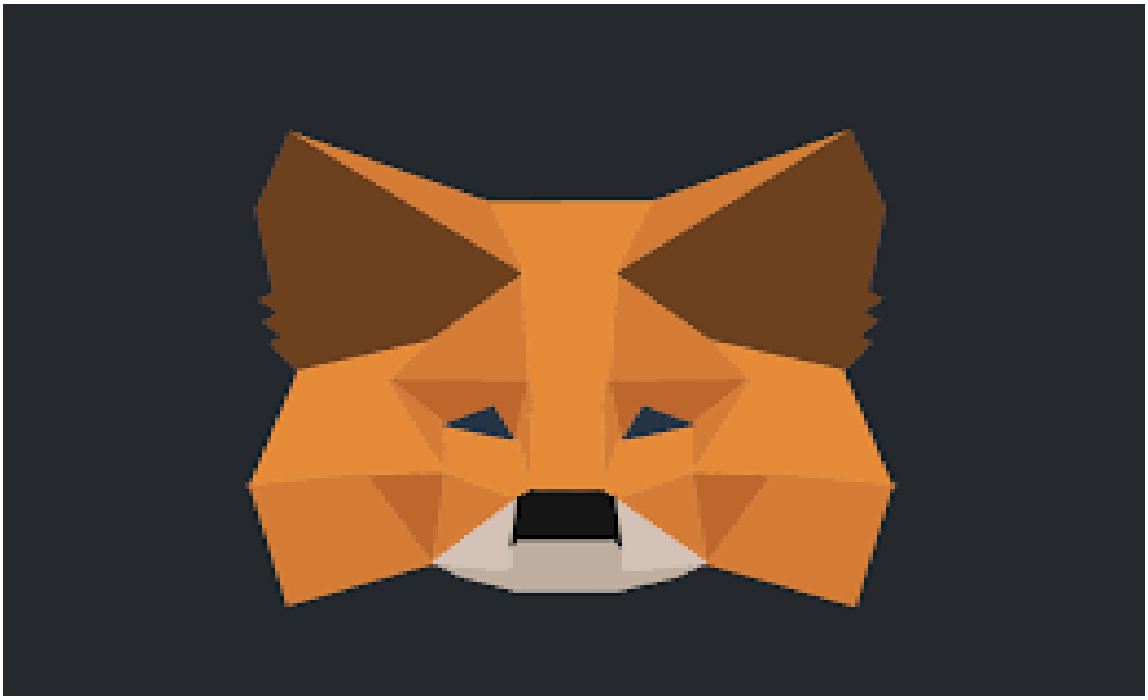


Fig. 3.1 METAMASK - a cryptocurrency supported in avax.

3.6 Avalanche (AVAX)

The Avalanche Foundation has launched Avalanche Multiverse, an up to $290M (up to 4M AVAX) incentive program focused on accelerating the adoption and growth of its novel “subnet” functionality, which enables a rich ecosystem of scalable app-specific blockchains. Initially, the program is focused on supporting new ecosystems including, but not limited to, blockchain-enabled gaming, DeFi, NFTs and institutional use cases. Additionally, Ava Labs, a core developer of the Avalanche client, will collaborate with the Aave Companies, Golden Tree Asset Management, Wintermute, Jump Crypto, Valkyrie, Securitize and others to build the first horizontally-integrated blockchain specifically engineered for Institutional DeFi with native KYC functionality. This will enable regulated institutions to leverage the power of Subnets to access DeFi primitives at scale and accelerate the institutional adoption of DeFi.

3.7 AWS

Amazon Web Services (AWS), the cloud platform offered by Amazon.com Inc (AMZN), has become a giant component of the e-commerce giant's business portfolio. AWS is made up of many different cloud computing products and services. The highly profitable division of Amazon provides servers, storage, networking, remote computing, email, mobile development, and security. AWS can be broken into three main products: EC2, Amazon’s virtual machine service, Glacier, a low-cost cloud storage service, and S3, Amazon’s storage system.

Arguably, Amazon Web Services is much more secure than a company hosting its own website or storage. AWS currently has dozens of data centers across the globe that are continuously monitored and strictly maintained. The diversification of the data centers ensures that a disaster striking one region doesn't cause permanent data loss worldwide. Imagine if Netflix were to have all of its personnel files, content, and backed-up data centralized on-site on the eve of a hurricane. Chaos would ensue.

3.8 Ganache

Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and deterministic environment. Ganache comes in two flavors: a UI and CLI. Ganache UI is a desktop application supporting both Ethereum and Corda technology. The command-line tool, ganache-cli (formerly known as the TestRPC), is available for Ethereum development. Prefer using the command-line? This documentation will focus only on the UI flavor of Ganache. Please see the Ganache CLI Readme for command-line documentation. All versions of Ganache are available for Windows, Mac, and Linux.

3.9 Truffle

Truffle Suite is a DApp development ecosystem that is divided into three sections. Truffle, Ganache, and Drizzle are the three components. This is a well-known ecosystem that a large number of developers rely on. Furthermore, the purpose of Truffle Suite is to make the development process more approachable.

A world class development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM), aiming to make life as a developer easier. With Truffle, you get:

- Built-in smart contract compilation, linking, deployment and binary management.
- Automated contract testing for rapid development.
- Scriptable, extensible deployment & migrations framework.
- Network management for deploying to any number of public & private networks.
- Package management with EthPM & NPM, using the ERC190 standard.
- Interactive console for direct contract communication.
- Configurable build pipeline with support for tight integration.
- External script runner that executes scripts within a Truffle environment.

Let's take a closer look at the components of the Truffle Suite:

- Truffle — Truffle is a development environment that runs on top of the Ethereum Virtual Machine (EVM). Truffle's slogan is "Smart Contracts Made Sweeter," implying that the ecosystem focuses on smart contract development. This environment has a variety of useful features that benefit DApp developers greatly.
- Ganache — Ganache is a tool for creating your own private Ethereum blockchain, which you may use to build and test smart contracts and decentralized applications (DApps) before releasing them on a public chain. As a result, Ganache allows developers to save money on gas during the development process.
- Drizzle — Drizzle is a set of frontend libraries built around a Redux store. Frontend development may become much more predictable and manageable using Drizzle.

Fig. 3.2 Truffle suite

CHAPTER 4

4.1 PROPOSED METHODOLOGY

METHODOLOGY

This section presents the methodology used for the Art auctioning app. Using this model, a tool is implemented for trading arts. In this method, the user can register as an owner or as a customer. The first step in this classification problem is auction initiation phase, followed by bidding, quality assurance, plagiarism assurance and then finally a safe transaction to make the sale a success. Figure [4.1] describes the proposed system methodology. The methodology is based on conducting various sales and for various users.

The user can opt to be an auctioneer or a bidder. The user can create an auction he/she chooses to sell or they can bid on an auction they would like. The auctioneer will give a promise of quality contract, which will be verified by the customer at the time of purchase. This contract is legally binding and if not matched, the auctioneer will indemnify for the same.

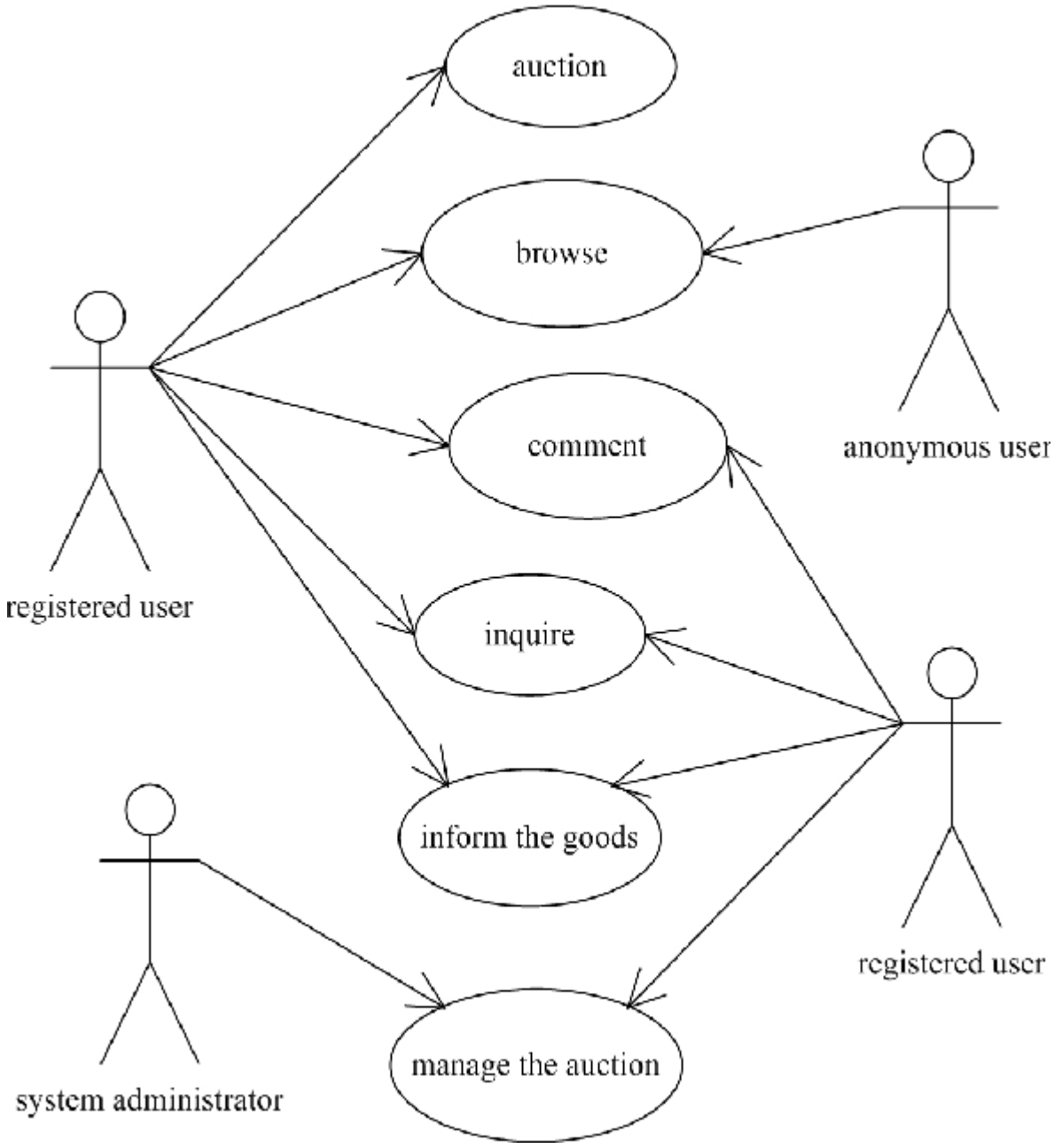


Fig. 4.1 Use case of the Model

The frontend of the website of done using react.js. React is a fully component based scripting language that was used to make the front-end very intituitve and enhance the user experience. The user experience is a very important concept of web devolopment and it was done to a decent extent with the help of comprehensive react.js framework.

The backend was done with the help of Blockchain technologies. Blockchain is a comprehensive concept and Truffle suite was used in it to link react.js and the backend. Whenever a payment request is made, Truffle needs a tool such as METAMASK, to implement the logical as well as physical link.

Finally, the project was deployed and hosted onto AWS CLI server and was implemented successfully.

CHAPTER 5

SYSTEM SPECIFICATION

5.1 Software Requirement

Table I. Software Specifications

S. No	Software	Version	URL
1	Avalanche	6.1	https://www.avax.network/
2	Metamask	10.11.3	https://metamask.io/
3	Truffle	5.5.18	https://www.npmjs.com/package/truffle
4	React.js	18.2.3	https://reactjs.org/
5	AWS	2.4	https://aws.amazon.com/

5.1.1 Truffle

Truffle Suite is a DApp development ecosystem that is divided into three sections. Truffle, Ganache, and Drizzle are the three components. This is a well-known ecosystem that a large number of developers rely on. Furthermore, the purpose of Truffle Suite is to make the development process more approachable.

5.1.2 Avalanche

The Avalanche Foundation has launched Avalanche Multiverse, an up to $290M (up to 4M AVAX) incentive program focused on accelerating the adoption and growth of its novel “subnet” functionality, which enables a rich ecosystem of scalable app-specific blockchains. Initially, the program is focused on supporting new ecosystems including, but not limited to, blockchain-enabled gaming, DeFi, NFTs and institutional use cases.

It is a symbolic math library, and is also used for machine learning applications suchas neural networks. It is used for both research and production at Google.

5.1.3 React.js

React is a popular JavaScript library used for web development. React.js or ReactJS or React are different ways to represent ReactJS. Today's many large-scale companies (Netflix, Instagram, to name a few) also use React JS.

5.1.4 Metamask

MetaMask is a cryptocurrency wallet that can be downloaded on Chrome and Firefox as a browser extension or as an App on iOS and Android devices. It allows users to store and transact crypto through a simple interface.

MetaMask enables users to interact with DeFi ecosystems on Ethereum and other blockchain networks, such as Binance Smart Chain, Polygon, and Avalanche. This guide will teach you how to connect your MetaMask to the Avalanche network.

5.1.5 AWS

Amazon Web Services (AWS), the cloud platform offered by Amazon.com Inc (AMZN), has become a giant component of the e-commerce giant's business portfolio. AWS is made up of many different cloud computing products and services. The highly profitable division of Amazon provides servers, storage, networking, remote computing, email, mobile development, and security. AWS can be broken into three main products: EC2, Amazon's virtual machine service, Glacier, a low-cost cloud storage service, and S3, Amazon's storage system.

5.2 Hardware Requirement

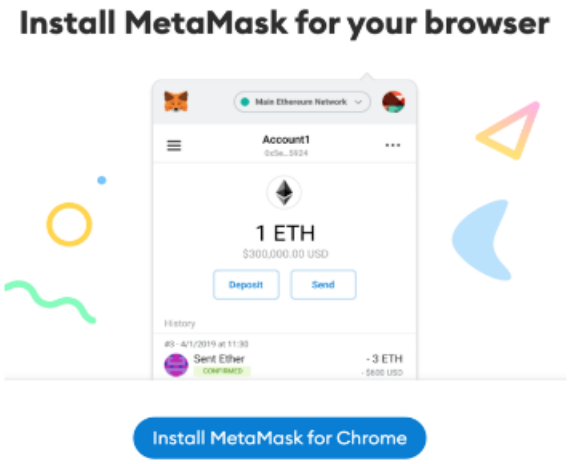
Table II. Hardware Specification

Processor	Intel Core
Operating System	Windows 10 (64-bit)
RAM	2 GB

5.3 Installation procedure

Step 1 — Install the dependencies for Windows.

Download and install the MetaMask extension on Chrome, iOS, or Android through the official MetaMask URL provided below.



2. You should see the MetaMask fox on the welcome screen after installation. Click [Get Started].

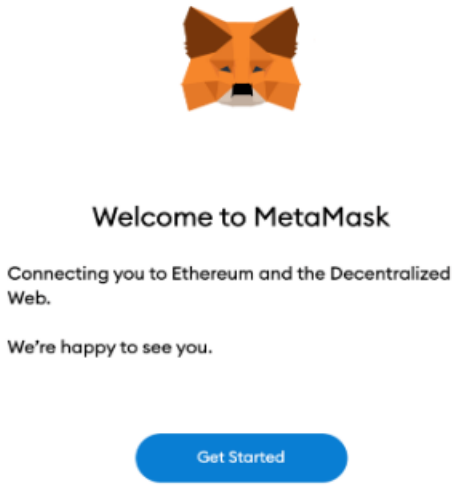


Fig. 5.1 MetaMask installation

Step 2 — Create a wallet and authenticate.

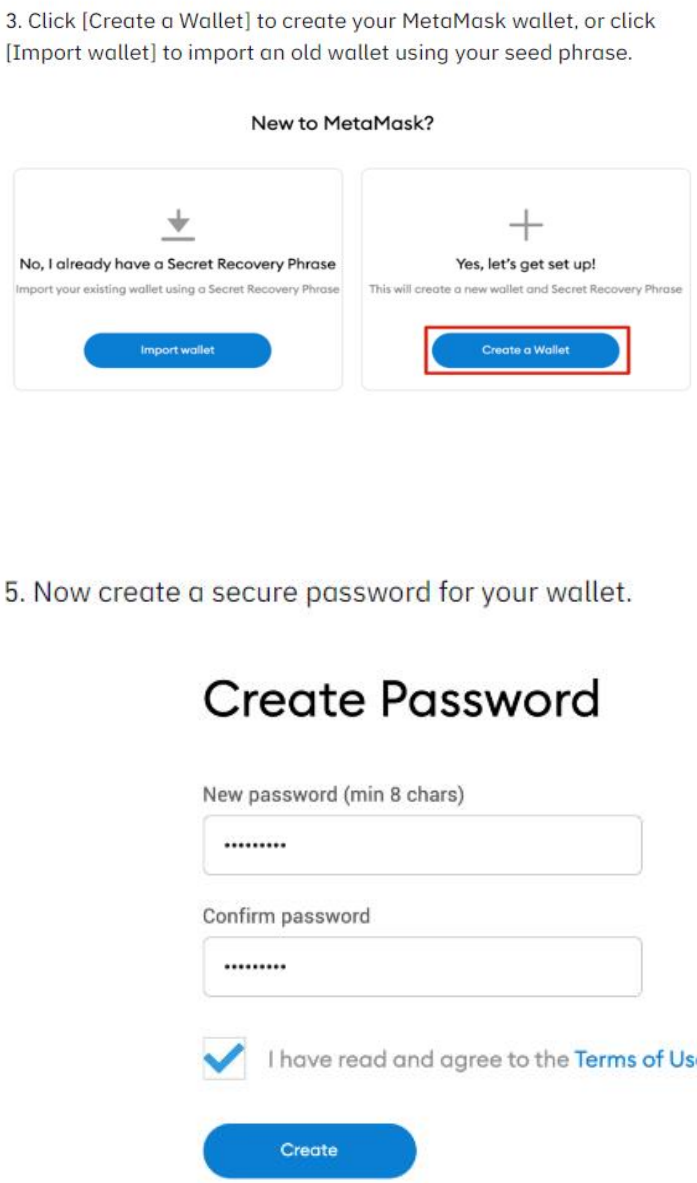


Fig. 5.2 Metamask wallet setup

Step 3 — Open your wallet extension and click the network dropdown menu and add new network

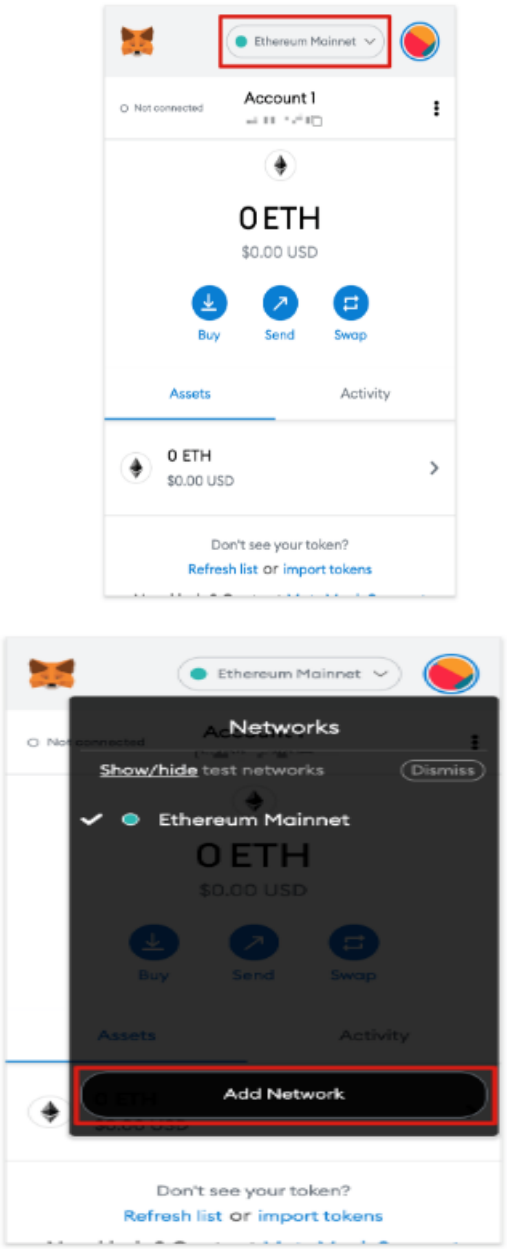


Fig. 5.3 Adding new network

Step 4 — You'll be redirected to the [Add a network] page. Copy and paste the following details and click [Save].

Network Name	Avalanche Network
New RPC URL	https://api.avax.network/ext/bc/C/rpc
Chain ID	43114 or 0xa86a
Currency Symbol	AVAX
Block Explorer URL	https://snowtrace.io/

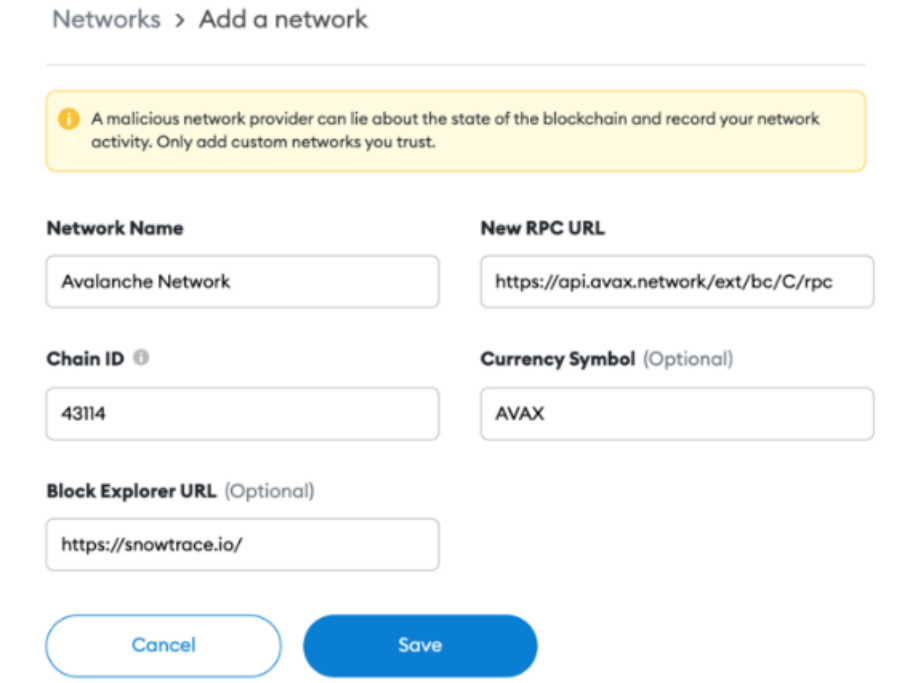


Fig. 5.4 Setup chain with credentials

CHAPTER 6

IMPLEMENTATION AND RESULTS

6.1 Opening up the environment to create the contract.

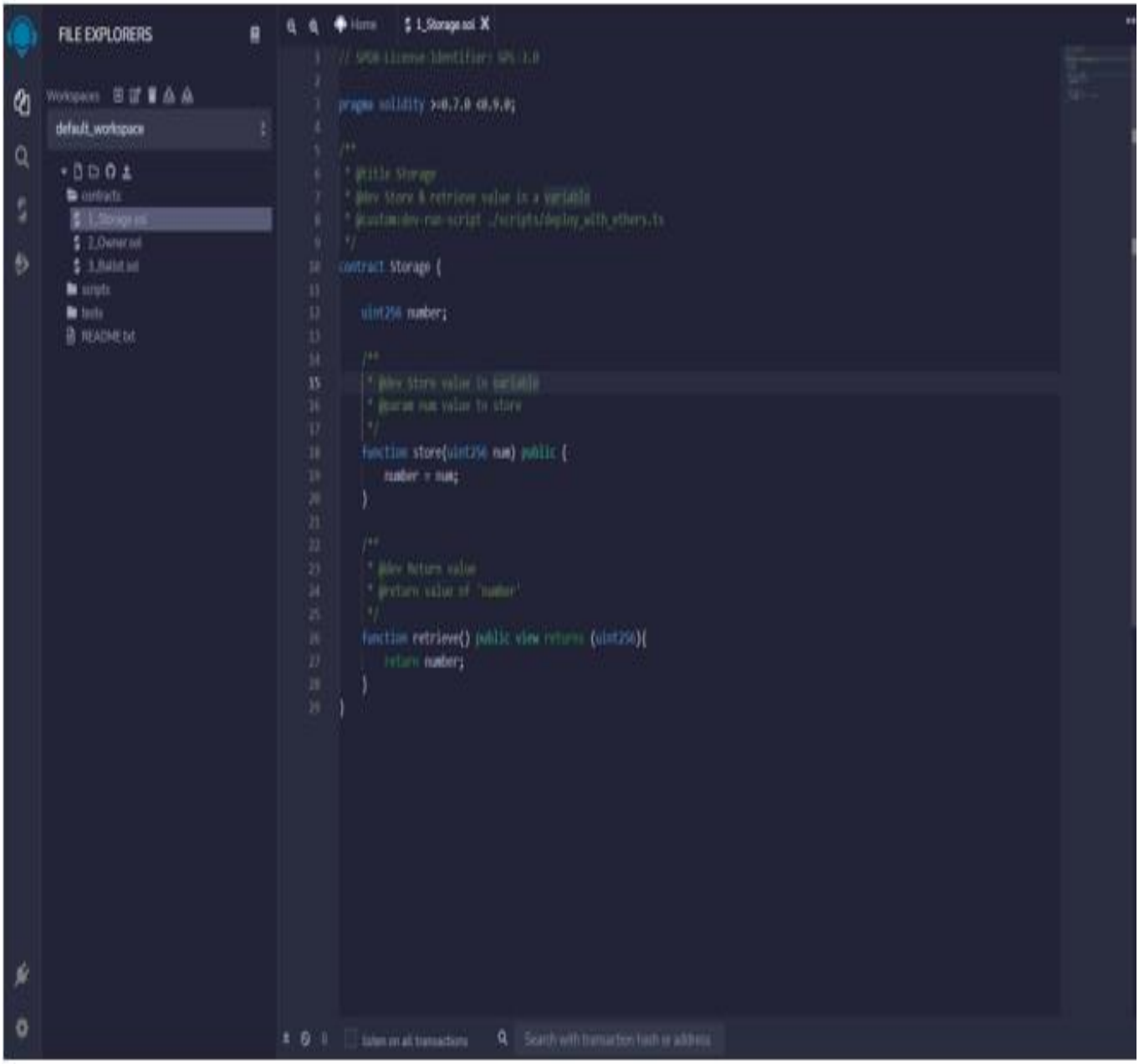


Fig .6.1 Editor setup

6.2 CODING

Codes for the frontend, backend and blockchains were written successfully and the bugs were eliminated.

The code for the contract was compiled successfully.

6.3 FRONTEND AND BACKEND CONNECTION.

The file directory was opened appropriately.

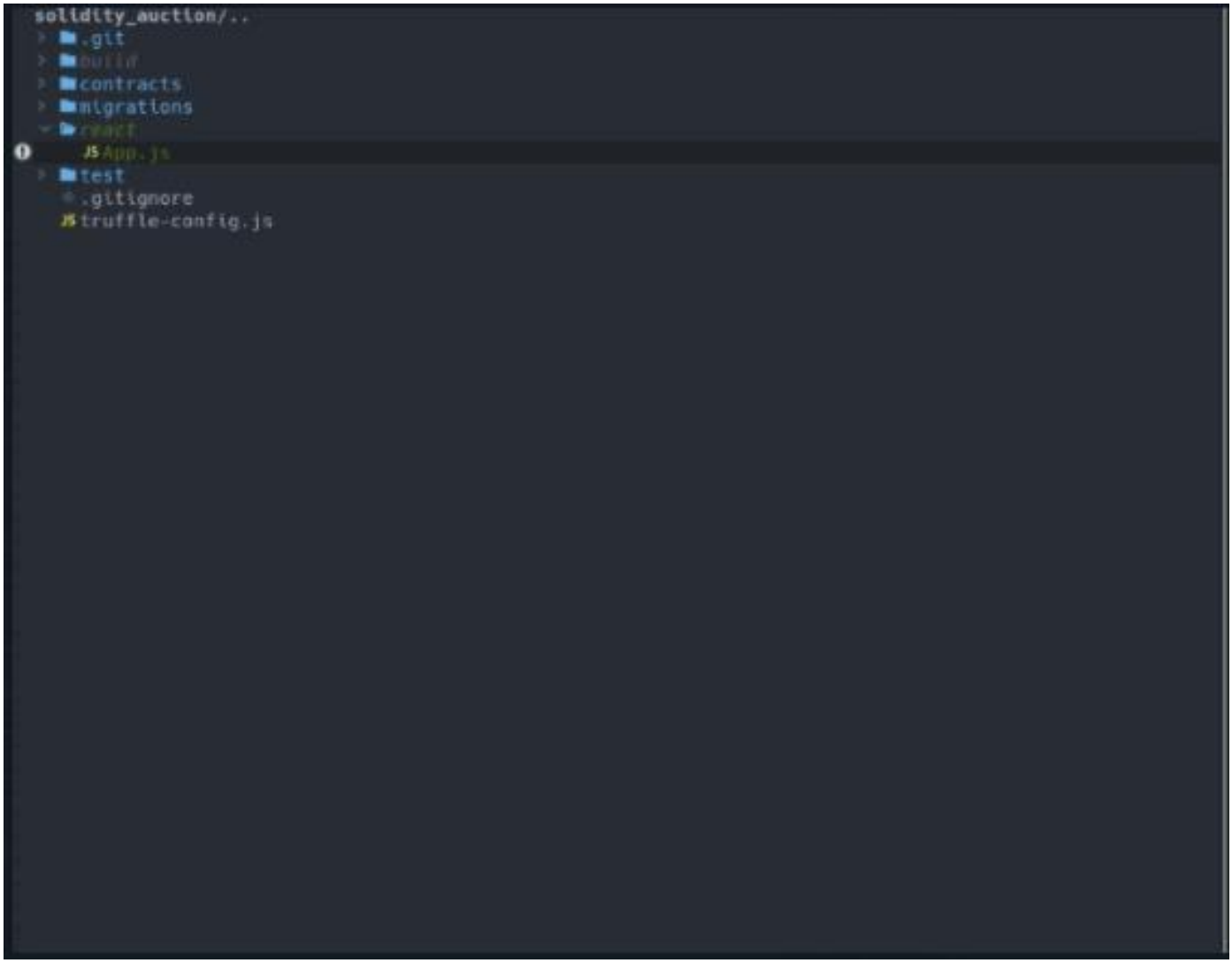
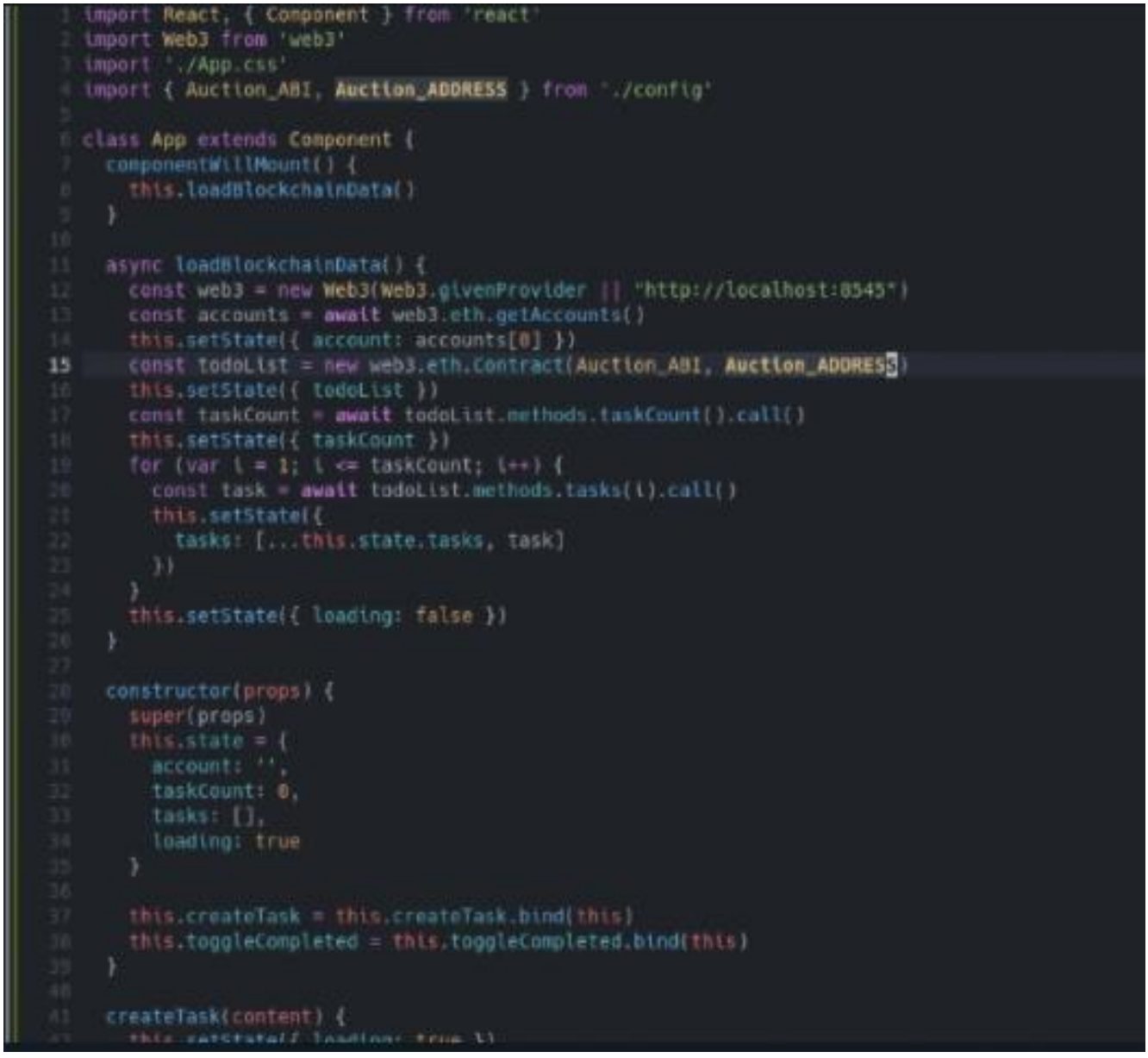


Fig. 6.2 File category.

Once the file directory was opened to app.JS; the react.js code was run successfully on the complier.



```
import React, { Component } from 'react'
import Web3 from 'web3'
import './App.css'
import { Auction_ABI, Auction_ADDRESS } from './config'

class App extends Component {
  componentWillMount() {
    this.loadBlockchainData()
  }

  async loadBlockchainData() {
    const web3 = new Web3(Web3.givenProvider || "http://localhost:8545")
    const accounts = await web3.eth.getAccounts()
    this.setState({ account: accounts[0] })
    const todoList = new web3.eth.Contract(Auction_ABI, Auction_ADDRESS)
    this.setState({ todoList })
    const taskCount = await todoList.methods.taskCount().call()
    this.setState({ taskCount })
    for (var i = 1; i <= taskCount; i++) {
      const task = await todoList.methods.tasks(i).call()
      this.setState({
        tasks: [...this.state.tasks, task]
      })
    }
    this.setState({ loading: false })
  }

  constructor(props) {
    super(props)
    this.state = {
      account: '',
      taskCount: 0,
      tasks: [],
      loading: true
    }

    this.createTask = this.createTask.bind(this)
    this.toggleCompleted = this.toggleCompleted.bind(this)
  }

  createTask(content) {
```

Fig. 6.3 React.js code for frontend

6.4 FINAL RESULT

The web app was built successfully and the auctioneering system works now with the desired result.

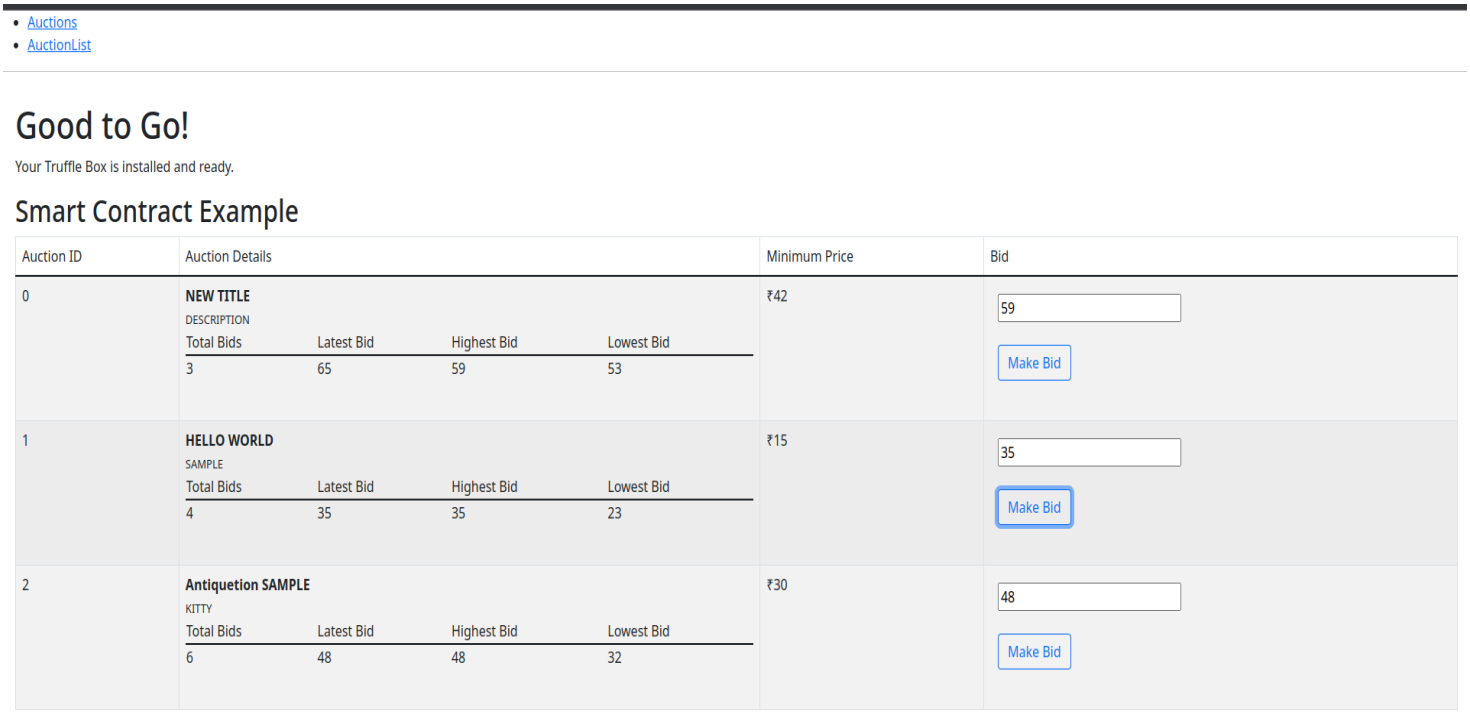


Fig. 6.4 WEB-APP model protoype.

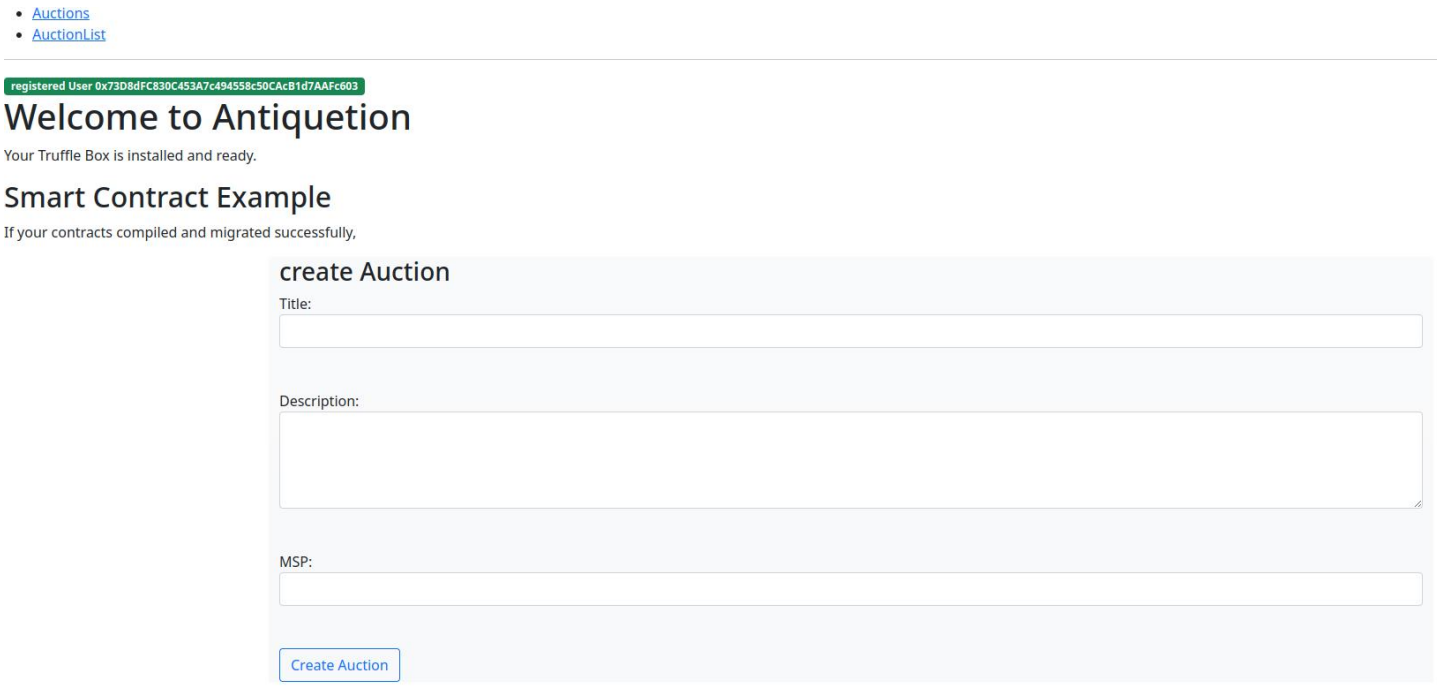


Fig. 6.5 Auction creating page

6.5 HOSTING

The website was hosted on AWS.

An EC2 instance and an S3 was created and the website was hosted on cloud.

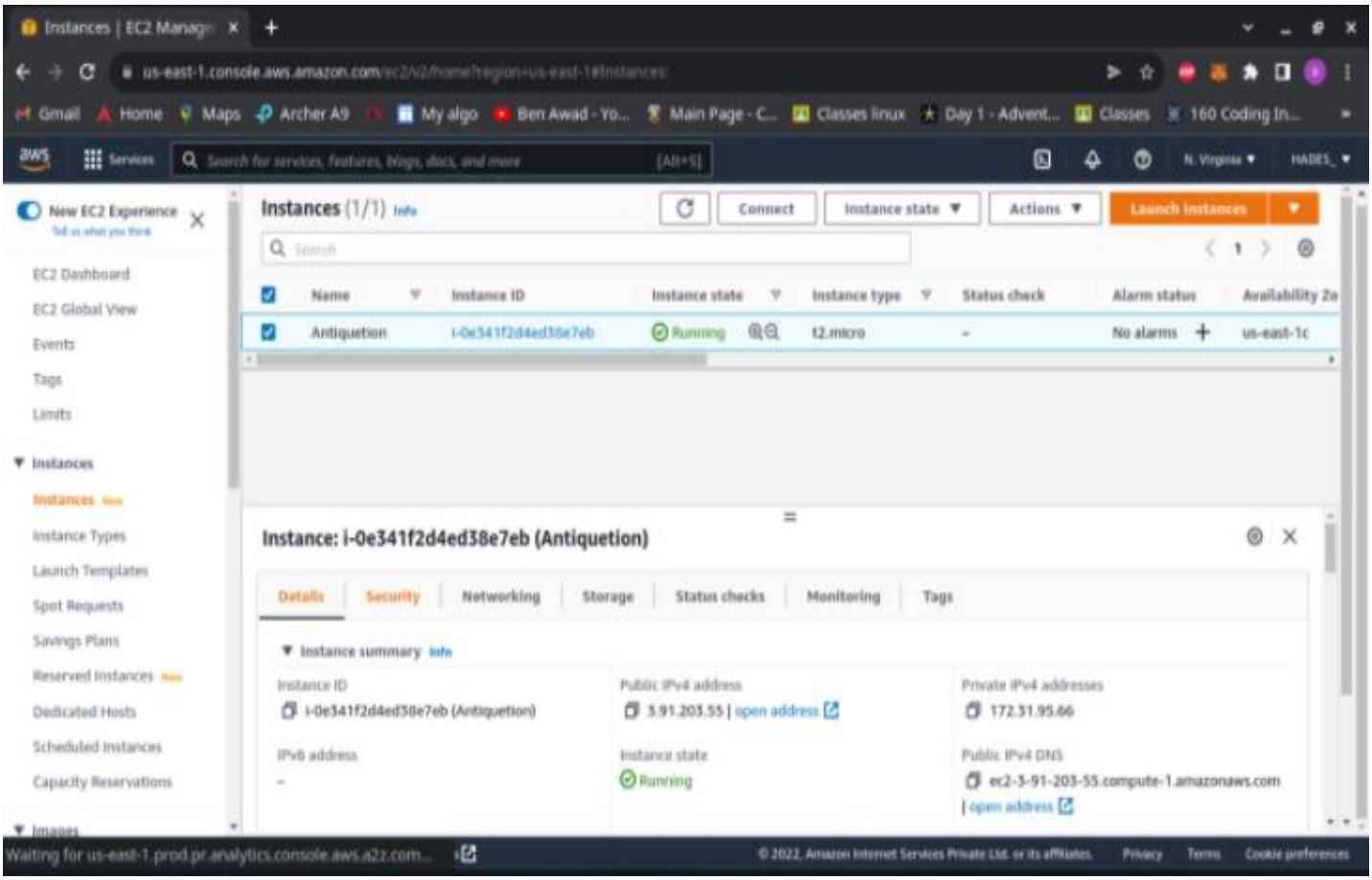


Fig. 6.6 AWS hosting image.

The Instance was created and a public IP was attached.

Instance ID: i-0e341f2d4ed38e7eb (Antiquetion)

Public IP: 3.91.203.55

CHAPTER7

CONCLUSION AND FUTURE SCOPE

In this project with the inbuilt security and integrity of the blockchain, auctioning of artworks and collectible have become much more simpler and hassle free for both the auctioneer and the buyers.

The artists can worry a little lesser as the authenticity and originality of the artwork is maintained to a better extent. This will prevent plagiarism. The smart contract is written in solidity language which is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behavior of accounts within the Ethereum state. Also, the intuitive user -interface will be able to juxtapose various artforms and buy at the price they feel it is right.

This application in the future can further be extended to sell NFT's as well. Digital arts are having a huge boom these days and we can help the market with a good auctioning app.

APPENDIX- I

SOURCE CODE

A1. SMART CONTRACT

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.8.0;

contract AuctionManager {
  uint public uId = 0;
  uint public aId = 0;
  // Structure to store user information
  struct User {
    uint userId;
    string name;
    address publicAddress;
  }

 // Structure to store bid information
  struct Bid {
    uint userId;
    uint bidPrice;
  }
  // Structure to store auction information
  struct Auction {
    uint auctionId;
```

```
    uint userId;
    string name;
    string description;
    uint msp;
    uint auctionBidId;
  }
  // Structure to store real time analytics of each
auction
  struct AuctionAnalytics {
    uint auctionId;
    uint auctionBidId;
    uint latestBid;
    uint lowestBid;
    uint highestBid;
  }

  // List of all auctions
  Auction[] public auctions;

  // Mapping for storing user info, bids and auction
analytics
  mapping (uint => User) public users;
  mapping (uint => Bid[]) public bids;
  mapping (uint => AuctionAnalytics) public
auctionAnalytics;
```

```
  // Public function to check the registration of users
(public address)
  function isRegistered(address _publicAddress)
public view returns (uint256[2] memory) {
    uint256[2] memory result = [uint256(0),
uint256(0)];
    for(uint i = 0; i < uId; i++) {
      if(_publicAddress == users[i].publicAddress) {
        result[0] = 1;
        result[1] = i;
        return result;
      }
    }
    return result;
  }

  function ReturnName(string memory name)
public pure returns (string memory){
    return name;
  }

  // Creating new users
  function createUser(string memory _name) public
returns(uint256){
    require((isRegistered(msg.sender))[0] == 0,
```

```
"User already registered!");
    users[uId] = User(uId, _name, msg.sender);
    uId++;
    return uId;
  }

  // Creating new auctions
  function createAuction(string memory _name,
string memory _description, uint _msp) public {
    require((isRegistered(msg.sender))[0] == 1,
"User not poda!!");
    uint MAX_UINT = 2 ** 256 - 1;
    auctions.push(Auction(aId,
isRegistered(msg.sender)[1], _name, _description,
_msp, 0));
    auctionAnalytics[aId] = AuctionAnalytics(aId, 0,
0, MAX_UINT, 0);
    aId++;
  }
  // Private function to update auction analytics after
the new bids
  function updateAucionAnalytics(uint _aId, uint
_latestBid) private {
    auctionAnalytics[_aId].latestBid = _latestBid;
    auctionAnalytics[_aId].auctionBidId =
auctions[_aId].auctionBidId;
```

```
    if(_latestBid <
auctionAnalytics[_aId].lowestBid) {
      auctionAnalytics[_aId].lowestBid = _latestBid;
    }
    if(_latestBid >
auctionAnalytics[_aId].highestBid) {
      auctionAnalytics[_aId].highestBid = _latestBid;
    }
  }

  // Creating new bids
  function createBid(uint _aId, uint _bidPrice)
public {
    require((isRegistered(msg.sender))[0] == 1,
"User not poda!");

bids[_aId].push(Bid((isRegistered(msg.sender))[1],
_bidPrice));
    auctions[_aId].auctionBidId++;
    updateAucionAnalytics(_aId, _bidPrice);
  }

  // Return list of all auctions
  function showAuctions() public view returns
(Auction[] memory) {
    return auctions;
```

```
        }
}
```

A2. REACT.JS WEB3 CONSTRUCTOR

```
import react, { component } from "react";
import simplestoragecontract from "./contracts/simplestorage.json";
import getweb3 from "./getweb3";

import "./app.css";

class app extends component {
  state = { storagevalue: 9, web3: null, accounts: null, contract: null };

  componentdidmount = async () => {
    try {
      // get network provider and web3 instance.
      const web3 = await getweb3();

      // use web3 to get the user's accounts.
      const accounts = await web3.eth.getaccounts();

      // get the contract instance.
      const networkid = await web3.eth.net.getid();
      const deployednetwork = simplestoragecontract.networks[networkid];

        const instance = new web3.eth.contract(
        simplestoragecontract.abi,
        deployednetwork && deployednetwork.address,
      );
```

```
    instance.options.address = "0x5731e62fba66cdc40d84fedab49f18a6d44c24bf";
    // set web3, accounts, and contract to the state, and then proceed with an
    // example of interacting with the contract's methods.
    this.setstate({ web3, accounts, contract: instance }, this.runexample);
  }
    catch (error) {
    // catch any errors for any of the above operations.
    alert(
      `failed to load web3, accounts, or contract. check console for details.`,
    );
    console.error(error);
  }

};
```

A3. DEPENDENCY LIST

```
  {
    "name": "client",
    "version": "0.1.0",
    "private": true,
    "dependencies": {
      "@babel/plugin-proposal-optional-chaining":
  "^7.17.12",
      "bootstrap": "^5.1.3",
      "react": "^17.0.2",
```

```
  "react-bootstrap": "^2.4.0",
  "react-dom": "^17.0.2",
  "react-redux": "^8.0.2",
  "react-router-dom": "^6.3.0",
  "react-scripts": "^3.4.1",
  "web3": "^1.6.1"
},
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
"eslintConfig": {
  "extends": "react-app"
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
```

```
      "last 1 safari version"
    ]
  }
}
```

A4. FRONTEND IN REACT.JS

```
import React, { Component, useContext } from
"react";
import AuctionManager from
"./contracts/AuctionManager.json";
import getWeb3 from "./getWeb3";
import 'bootstrap/dist/css/bootstrap.min.css';
import { GlobalContext, GlobalProvider } from
'./GlobalContext';

class Auction extends Component {
  state = {
    storageValue: 9, web3: null, accounts: null,
contract: null, bidAmount: 0, isRegistered: false,
title: "",
    description: "",
    msp: "",
    name: "sakthi"
  };
```

```
  componentDidMount = async () => {
    try {
      // Get network provider and web3 instance.
      const web3 = await getWeb3();

      // Use web3 to get the user's accounts.
      const accounts = await web3.eth.getAccounts();

      // Get the contract instance.
      const networkId = await web3.eth.net.getId();
      const deployedNetwork =
AuctionManager.networks[networkId];

      const instance = new web3.eth.Contract(
        AuctionManager.abi,
        deployedNetwork &&
deployedNetwork.address,
      );
      instance.options.address =
"0x5731e62Fba66cDC40d84FEdab49f18A6d44c24
Bf";
      // Set web3, accounts, and contract to the state,
and then proceed with an
      // example of interacting with the contract's
methods.
```

```
      // const response = await
contract.methods.isRegistered(accounts[0]).call();
      // if(response[0]==1){
      //   this.setState({isRegistered:true})
      // }

      this.setState({ web3, accounts, contract:
instance }, this.runExample);
    } catch (error) {
      // Catch any errors for any of the above
operations.
      alert(
        `Failed to load web3, accounts, or contract.
Check console for details.`,
      );
      console.error(error);
    }
  };

  createAuc = async () => {

    const { accounts, contract, title, description, msp
} = this.state;
```

```
    console.log("hello", "creating")
    const auction = await
contract.methods.createAuction(title, description,
msp).send({ from: accounts[0] });

    const auctionList = await
contract.methods.showAuctions().call();
    console.log(auctionList);
  }
  registerUser = async () => {
    const { accounts, contract } = this.state;
    const register = await
contract.methods.createUser(this.state.name).send(
{ from: accounts[0] });



  }

  createBid = async () => {

    const { accounts, contract, bidAmount } =
this.state;
    console.log("hello", "creating")
    const auction = await
contract.methods.createBid(0, bidAmount).send({
```

```
from: accounts[0] });
    const AuctionAnalytics = await
contract.methods.auctionAnalytics(0).call();
    console.log(AuctionAnalytics);
  }
  handleBidChange = (e) => {
    e.preventDefault();
    this.setState({ bidAmount: e.target.value });

  }
  runExample = async () => {
    const { accounts, contract } = this.state;

    console.log(contract)
    // Stores a given value, 5 by default.

    // Get the value from the contract to prove it
worked.
    console.log(accounts)
    const n = await contract.methods.uId().call();
    console.log(n);
    // for (let i = 0; i < n; i++) {
    const a = await contract.methods.users(0).call()
    console.log("user id =0", a);
```

```
    const isRegistered_ = await
contract.methods.isRegistered(accounts[0]).call();
    this.setState({ isRegistered: isRegistered_ })

    // }
    const response = await
contract.methods.isRegistered(accounts[0]).call();
    console.log(response)
    this.registerUser();
    // console.log(auctionList);
    // Update state with the result.
    // this.setState({ storageValue: auctionList });

  };
  handleAuctionTitleChange = (event) => {
    this.setState({ title: event.target.value })
  }
  handleAuctionDescriptionChange = (event) => {
    this.setState({ description: event.target.value })
  }

  handleAuctionMspChange = (event) => {
    this.setState({ msp: event.target.value })
  }
  submitNewAuction = (event) => {
```

```
    event.preventDefault();
    this.createAuc();
  }

  registerNewUser = (event) => {
    event.preventDefault();
    this.registerUser()
  }
  handleAuctionNameChange = (event) => {
    this.setState({ name: event.target.value })
  }
  render() {
    let a = {
      a: "hello"
    }
    if (!this.state.web3) {
      return <div>Loading Web3, accounts, and
contract...</div>;
    }
    return (
      <div className="App jumbotron justify">
        <div ><span
className={this.state.isRegistered ? "badge bg-
success" : "badge bg-
danger"}>{this.state.isRegistered ? `registered User
${this.state.accounts[0]}` : "non registered user
```

```
please register"}</span></div>
        <h1>Welcome to Antiquetion</h1>
        <p>Your Truffle Box is installed and
ready.</p>
        <h2>Smart Contract Example</h2>
        <p>
          If your contracts compiled and migrated
successfully,
              </p>
        {/*              {!this.state.isRegistered &&  */}
        {/* <form onSubmit={this.registerNewUser}
className="container-sm bg-light bg-gradient
  w-25"> */}
        {/* <label>Enter name: </label><br /><input
value={this.state.name}
onChange={this.handleAuctionNameChange}
class="form-control"/><br /><br /> */}
        {/* <input type="submit" value="Register
User" className="btn btn-outline-primary"/> */}
        {/* </form> */}


        {/* } */}


        <div>
          <form onSubmit={this.submitNewAuction}
className="container-sm bg-light bg-gradient
```

```
  ">
          <h3>create Auction</h3>
          <label>Title: </label><br /><input
value={this.state.title}
onChange={this.handleAuctionTitleChange}
class="form-control" /><br /><br />
          <label>Description: </label><br /><textarea
rows="4" value={this.state.description}
onChange={this.handleAuctionDescriptionChange
} class="form-control" /><br /><br />
          <label>MSP: </label><br /><input
value={this.state.msp}
onChange={this.handleAuctionMspChange}
class="form-control" /><br /><br />
          <input type="submit" value="Create
Auction" className="btn btn-outline-primary" />
        </form>
      </div>
    </div>
   );
  }
}

export default Auction;
```

APPENDIX – II

REFERENCES

[1] Vikas Hassija; Gaurang Bansal; Vinay Chamola; Vikas Saxena; Biplab Sikdar- **BlockCom:** A Blockchain Based Commerce Model for Smart Communities using Auction Mechanism

[2] Steven Goldfeder, Joseph Bonneau, Rosario Gennaro & Arvind Narayanan : "Escrow Protocols for Cryptocurrencies: How to Buy Physical Goods Using Bitcoin"

[3] Jehangir Arshad; Muhammad Abu Bakar Siddique; Zainab Zulfiqar; Amna Khokhar; Saqib Salim; Talha Younas: "A Novel Remote User Authentication Scheme by using Private Blockchain-Based Secure Access Control"

[4] Xianyun Ge : "Smart Payment Contract Mechanism Based on Blockchain Smart Contract Mechanism.

[5] C. Esposito, A. De Santis, G. Tortora, H. Chang and K.-K. R. Choo, "Blockchain: A panacea for healthcare cloud-based data security and privacy?", *IEEE Cloud Computing*, vol. 5, no. 1, pp. 31-37, 2018.

[6] Hybrid Blockchain Architecture for Privacy-Enabled and Accountable Auctions- Harsh Desai; Murat Kantarcioglu; Lalana Kagal

[7] Optimizing Blockchain Based Smart Grid Auctions: A Green Revolution- Muneeb Ul Hassan;Mubashir Husain Rehmani;Jinjun Chen

[8] SECAUCTEE: Securing Auction Smart Contracts using Trusted Execution Environments- Harsh Desai;Murat Kantarcioglu

[9] BEADS: Blockchain-Empowered Auction in Decentralized Storage-Xinxuan Huang;Jigang Wu;Jiaxing Li;Chengpeng Xia