

A
Mini Project
On
BLOCKCHAIN BASED PORTAL FOR FARMERS

(Submitted in partial fulfillment of the requirements for the award of Degree)

BACHELOR OF TECHNOLOGY

In
COMPUTER SCIENCE AND ENGINEERING

By
PADIGELA SAHITHI REDDY (207R1A0547)
KIRTHI SANTOSHI LAXMI (207R1A0527)

Under the Guidance of
DR. K. SRUJAN RAJU
(Professor)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CMR TECHNICAL CAMPUS
UGC AUTONOMOUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New
Delhi) Recognized Under Section 2(f) & 12(B) of the UGC Act. 1956, Kandlakoya (V),
Medchal Road, Hyderabad-501401.

2020-2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the project entitled “**BLOCK CHAIN BASED PORTAL FOR FARMERS** ” being submitted by **PADIGELA SAHITHI REDDY (207R1A0547)** **KIRTHI SANTOSHI LAXMI (207R1A0527)** in partial fulfillment of the requirements for the award of the degree of B. Tech in Computer Science and Engineering to the CMR Technical Campus, is a record of bonafide work carried out by them under our guidance and supervision during the year 2023-2024.

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Dr. K. SRUJAN RAJU
(Professor)
INTERNAL GUIDE

Dr. A. Raji Reddy
DIRECTOR

Dr. K. Srujan Raju
HOD

EXTERNAL EXAMINER

Submitted for viva voice Examination held on _____

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of any project depends largely on the encouragement and guidelines of many others. I take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We take this opportunity to express my profound gratitude and deep regard to my guide **Dr. K. Srujan Raju**, Professor for his exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given by him shall carry us a long way in the journey of life on which I am about to embark.

We also take this opportunity to express a deep sense of gratitude to the Project Review Committee (PRC) **G. Vinesh Shanker, Dr. J. Narasimharao, Ms. Shilpa, & Dr. K. Maheshwari** for their cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to **Dr. K. Srujan Raju**, Head, Department of Computer Science and Engineering for providing encouragement and support for completing this project successfully.

We are obliged to **Dr. A. Raji Reddy**, Director for being cooperative throughout the course of this project. We also express our sincere gratitude to Sri. **Ch. Gopal Reddy**, Chairman for providing excellent infrastructure and a nice atmosphere throughout the course of this project.

The guidance and support received from all the members of **CMR Technical Campus** who contributed to the completion of the project. We are grateful for their constant support and help.

Finally, We would like to take this opportunity to thank our family for their constant encouragement, without which this assignment would not be completed. We sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of this project.

P SAHITHI REDDY (207R1A0547)

K SANTOSHI LAXMI (207R1A0527)

ABSTRACT

Farmers, as well as agriculture, are the foundation of life. Numerous work has been done towards the enhancement of agriculture by developing technologies that support directly and indirectly to agriculture. The Government of India has also taken many initiatives for the same. Few examples of such portals are Krishijagran.com, farmer.gov.in, agricrop.nic.in and agriwatch.com etc. A range of research shows that with the various enhancements in the field of ICT(Information and Communications Technology) , the farmers are unable to take its advantage and fail to get the proper sale value for their crops. So, Considering the features of blockchain such as immutability, decentralization and maintaining the footage of transaction details, this project highlights the usage of blockchain technology with farmer's portal that keep the track of selling and buying information of crops. This blockchain system will benefit the farmers or vendors and individuals by preserving the contract of trade. An interface for the farmers is designed using a python programming language in addition with blockchain technology, which is used to store the information related to seller, buyer, selling and buying an item and total value transacted.

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
Figure 3.1	Architecture of Blockchain Based Portal for Farmers	7
Figure 3.3	Use Case Diagram of Block chain Based Portal for Farmers	10
Figure 3.4	Class Diagram of Blockchain Based Portal for Farmers	11
Figure 3.5	Sequence diagram of Block chain Based Portal for Farmers	12
Figure 3.6	Activity diagram of Block chain Based Portal for Farmers	13

LIST OF SCREENSHOTS

SCREENSHOT NO.	SCREENSHOT NAME	PAGE NO.
Screenshot 5.1	Home Page of Farmer's Portal	33
Screenshot 5.2	Seller Register Form	33
Screenshot 5.3	Buyer Register Form	34
Screenshot 5.4	Seller Add Crop Form	34
Screenshot 5.5	Seller Update Crop Page	35
Screenshot 5.6	Buyer Search Results Page	35
Screenshot 5.7	Buyer Cart Page	36
Screenshot 5.8	Buyer Transaction Details View Page	36
Screenshot 5.9	Buyer Purchased Crops view	37
Screenshot 5.10	Admin View of Transactions	37

TABLE OF CONTENTS

ABSTRACT	i
LIST OF FIGURES	ii
LIST OF SCREENSHOTS	iii
1.INTRODUCTION	1
1.1 PROJECT SCOPE	1
1.2 PROJECT PURPOSE	1
1.3 PROJECT FEATURES	1
2.SYSTEM ANALYSIS	2
2.1 PROBLEM DEFINITION	2
2.2 EXISTING SYSTEM	2
2.2.1 LIMITATIONS OF THE EXISTING SYSTEM	3
2.3 PROPOSED SYSTEM	3
2.3.1 ADVANTAGES OF PROPOSED SYSTEM	3
2.4 FEASIBILITY STUDY	4
2.4.1 ECONOMIC FEASIBILITY	4
2.4.2 TECHNICAL FEASIBILITY	5
2.4.3 SOCIAL FEASIBILITY	5
2.5 HARDWARE & SOFTWARE REQUIREMENTS	5
2.5.1 HARDWARE REQUIREMENTS	5
2.5.2 SOFTWARE REQUIREMENTS	6
3.ARCHITECTURE	7
3.1 PROJECT ARCHITECTURE	7
3.2 DESCRIPTION	8
3.3 USE CASE DIAGRAM	10
3.4 CLASS DIAGRAM	11
3.5 SEQUENCE DIAGRAM	12
3.6 ACTIVITY DIAGRAM	13
4.IMPLEMENTATION	14
4.1 SAMPLE CODE	14
5.RESULT	33

6.TESTING	38
6.1 INTRODUCTION TO TESTING	38
6.2 TYPES OF TESTING	38
6.2.1 UNIT TESTING	38
6.2.2 INTEGRATION TESTING	39
6.2.3 FUNCTIONAL TESTING	39
6.3 TEST CASES	40
6.3.1 CLASSIFICATION	40
7.CONCLUSION & FUTURE SCOPE	42
7.1 PROJECT CONCLUSION	42
7.2 FUTURE SCOPE	42
8.BIBLIOGRAPHY	43
8.1 REFERENCES	43
8.2 GITHUB LINK	43

1. INTRODUCTION

1. INTRODUCTION

1.1 PROJECT SCOPE

This project is titled “BLOCKCHAIN BASED PORTAL FOR FARMERS”. It is an online portal for farmers, a single gateway through which the e-commerce activity of crops can be performed. This portal helps farmers to sell their crops and track payments securely. The use of Blockchain technology ensures that the data is tamper-proof and whole process of supply chain is transparent and immutable.

1.2 PROJECT PURPOSE

This blockchain based portal is proposed to deal with the issue of demand and sale price of crops which in result ensure crop security to farmers as well as to get fair price of the crop. It is like a digital helper for farmers, making it simpler for them to sell their products, get fair price, and keep track of everything. Leading the generations towards a decentralized network and entrusting them with the services provided, is the **is** the main objective of our study and this project.

1.3 PROJECT FEATURES

The users experience of the portal can be tailored according to the individual needs. The buyer can buy, search and add the product into the cart. The seller can add a new item, update the existing items, allot and update the price of the item. It implements a peer-to-peer transaction, coordination and collaborations based on distributed system databases by means of data encryption and time stamping. The blockchain distributed ledger guarantees the fairness and authenticity of transaction and avoids possibility of transaction being tampered.

2. SYSTEM ANALYSIS

2. SYSTEM ANALYSIS

SYSTEM ANALYSIS

System Analysis is the important phase in the system development process. The System is studied to the minute details and analyzed. The system analyst plays an important role of an interrogator and dwells deep into the working of the present system. In analysis, a detailed study of these operations performed by the system and their relationships within and outside the system is done. A key question considered here is, “what must be done to solve the problem?” The system is viewed as a whole and the inputs to the system are identified. Once analysis is completed the analyst has a firm understanding of what is to be done.

2.1 PROBLEM DEFINITION

The general statement of BLOCKCHAIN BASED PORTAL FOR FARMERS is to introduce an online portal for farmers to deal with the issue of demand and sale price of crops which in result ensure crop security to farmers as well as to get fair price of the crop.

2.2 EXISTING SYSTEM

Various enhancements were made in the field of agriculture, but the farmers fail to get the proper sale value for their crops. However, the system was good in providing the instant update to the farmer but the version was only in the English language, which was the limitation of the system. Involvement of middleman also leads to several problems. The incorporation of blockchain technology in agriculture has improved the efficiency of the agriculture supply chain by reducing the need for verification of data. Blockchain technology has proposed an agricultural tracing system that is decentralized, maintained collectively and provides trust and reliability in case of supply chain management.

2.2.1 LIMITATIONS OF EXISTING SYSTEM

Following are the disadvantages of existing system:

- Transaction depends on third party
- Intermediate disputes due to middle men
- Data stored in local servers it means data is not secure
- Tampered Transactions

2.3 PROPOSED SYSTEM

The Proposed Farmer's portal is a single gateway through which the e-commerce activity of crops can be performed. The portal can be tailored according to the individual need. It is a single access point i.e., everything is in a single place, the only thing needed is single login to approved users. User: A user can be a buyer or a seller. The seller may be either a farmer or representative of him. Interface: To access the portal, the user needs to register using a sign up. The registered user logs in using the correct credentials. Once the user signs in successfully. The user will have access to the portal/ interface. All these transactions are visible to everyone in the network. The blockchain is a peer-to-peer transaction based on distributed node systems by means of data encryption, time stamping and consensus. It makes the portal more secure at the data as it is immutable, transparent, and accessible to all.

2.3.1 ADVANTAGES OF THE PROPOSED SYSTEM

- The speed of the transaction is improved
- Immutable records
- Transparency is maintained in supply chain
- Decentralized system
- Improved security

2.4 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. Three key considerations involved in the feasibility analysis:

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

2.4.1 ECONOMIC FEASIBILITY

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on a project, which will give best, return at the earliest. One of the factors, which affect the development of a new system, is the cost it would require.

The following are some of the important financial questions asked during preliminary investigation :

- The costs conduct a full system investigation.
- The cost of the hardware and software.
- The benefits in the form of reduced costs or fewer costly errors.

Since the system is developed as part of project work, there is no manual cost to spend for the proposed system. Also all the resources are already available, it give an indication that the system is economically possible for development.

2.4.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.4.3 SOCIAL FEASIBILITY

This includes the following questions:

- Is there sufficient support for the users?
- Will the proposed system cause harm?

The project would be beneficial because it satisfies the objectives when developed and installed. All behavioral aspects are considered carefully and conclude that the project is behaviorally feasible.

2.5 HARDWARE & SOFTWARE REQUIREMENTS

2.5.1 HARDWARE REQUIREMENTS:

Hardware interfaces specify the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

- Processor : Intel core I3 or above.
- Hard disk : 516GB SSD or above.
- Memory : 8GB and above.

2.5.2 SOFTWARE REQUIREMENTS:

Software Requirements specifies the logical characteristics of each interface and software components of the system.

The following are some software requirements.

- Operating system : Windows 8 or above.
- Front-end : HTML, CSS and JavaScript
- Code language : Python
- Frameworks : Django
- Database : SQLite
- Security and Integration tools

3. ARCHITECTURE

3.ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project architecture shows the procedure how the transactions are maintained in the portal

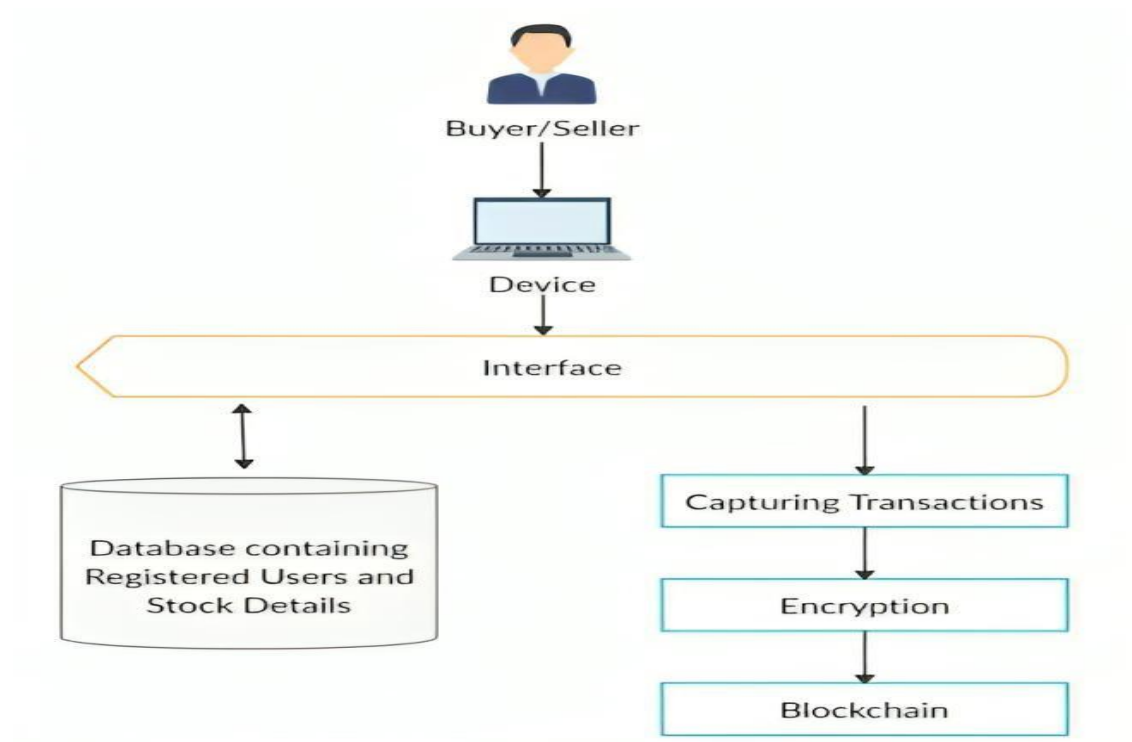


Figure 3.1: Architecture of Farmers Portal

3.2 DESCRIPTION

In the era of information and communication technology, a farmer's portal has always been helpful for farmers in many ways, providing ease of use and convenience of information to the farmers. Using blockchain technology in the field can make available decentralized computation and information sharing platform that enables multiple authoritative domains, which do not trust each other, to cooperate, coordinate and collaborate in a rational decision making process, a reliable information recording system can be made that can contribute for the development in the agriculture sector. Since blockchain works like a public ledger, so it can be utilized to ensure many different aspects:

- **Protocols for Commitment:** Ensure that every valid transaction from the clients are committed and included in the blockchain within a finite time.
- **Consensus:** Ensure that the local copies are consistent and updated.
- **Security:** The data needs to be tamper -proof. Note that the client may act maliciously or can be compromised.
- **Privacy and Authenticity:** The data or transactions belong to various clients; privacy and authenticity need to be ensured.

Cryptography is a crucial part of how blockchain technology works. Public key encryption is like the foundation of blockchain wallets and transactions. Cryptographic hash functions give the important quality of immutability, ensuring that data can't be changed. Merkle trees help organize transactions efficiently and make blockchain work better. All these things combined make blockchain secure and efficient.

Ensuring the above aspects numerous work has been carried out in the field of blockchain. The presented portal is a contribution over them. It can help to maintain a secure platform for farmers, where they can trade with the customers electronically. The main objective of this study is to record the secure transactions between a seller and a buyer that ensures a contract between the two. This can help farmers to get a legitimate price for their commodity. The system also facilitates a single place to record the whole trade transaction.

3.3 USE CASE DIAGRAM

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

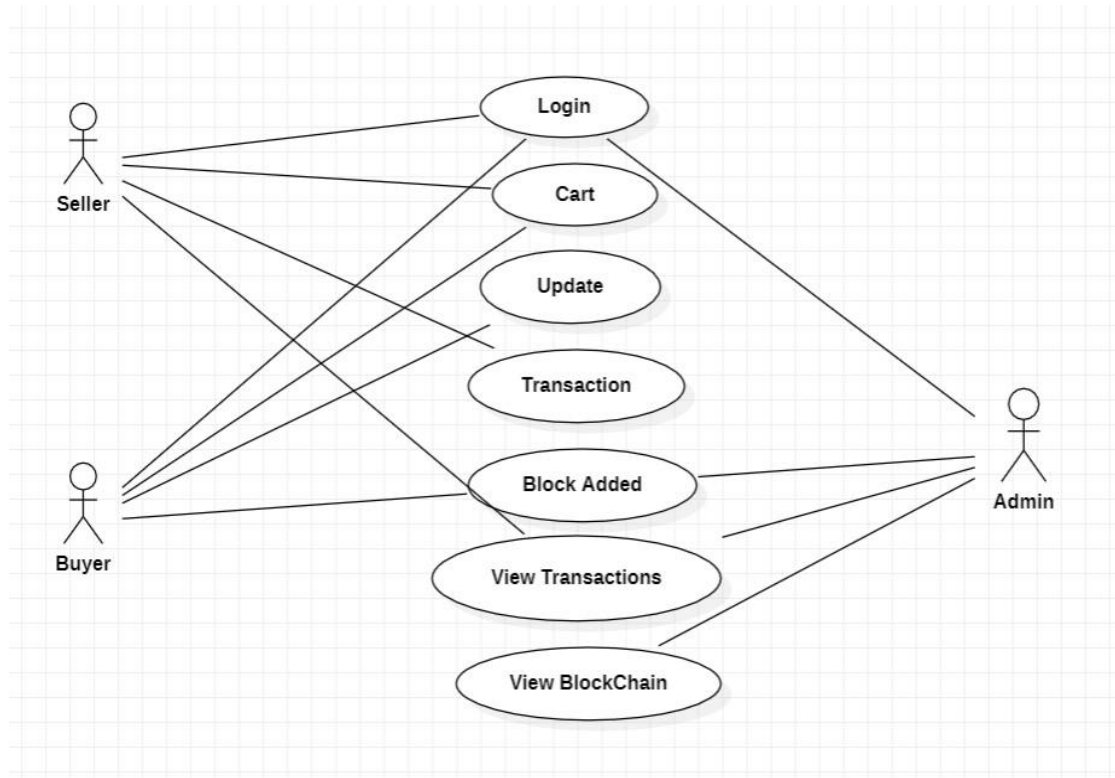


Figure 3.2: Use Case Diagram of Blockchain Based Portal For Farmers

3.4 CLASS DIAGRAM

Class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

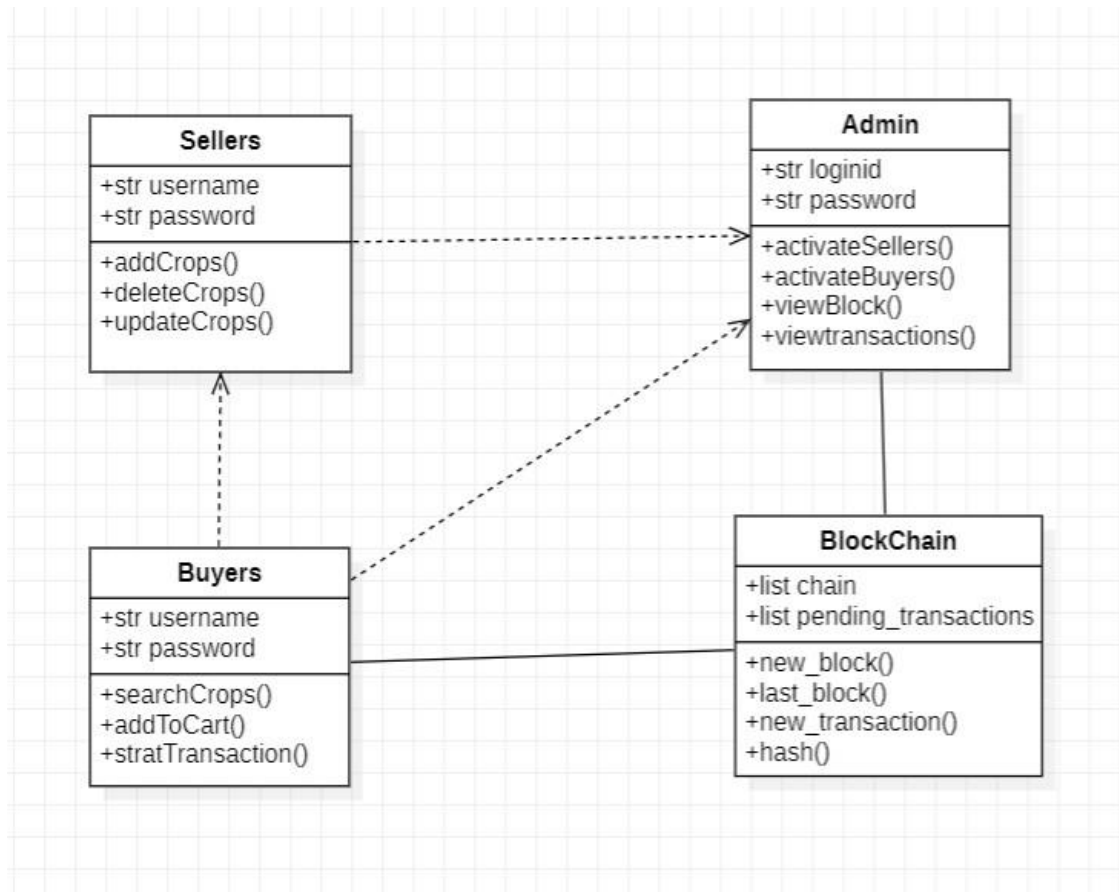


Figure 3.3: Class Diagram of Blockchain Based Portal For Farmers

3.5 SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development.

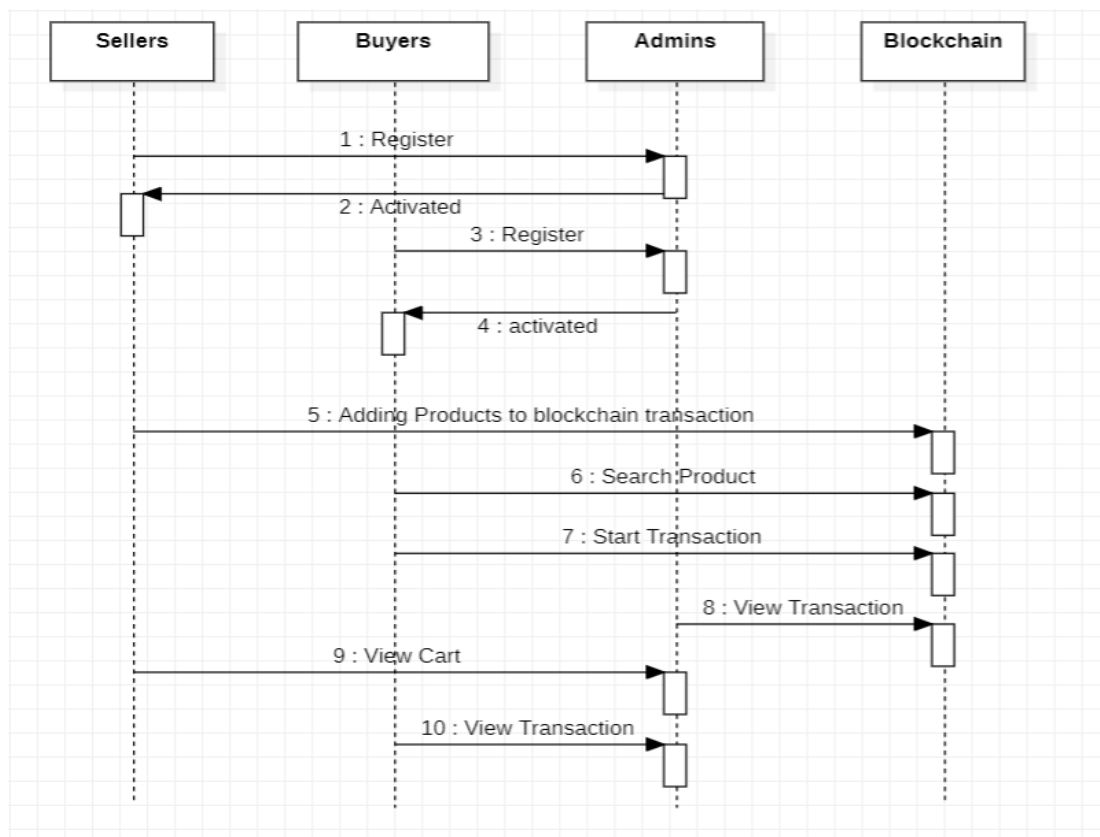


Figure 3.4: Sequence Diagram of Blockchain Based Portal For Farmers

3.6 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. They can also include elements showing the flow of data between activities through one or more data stores.

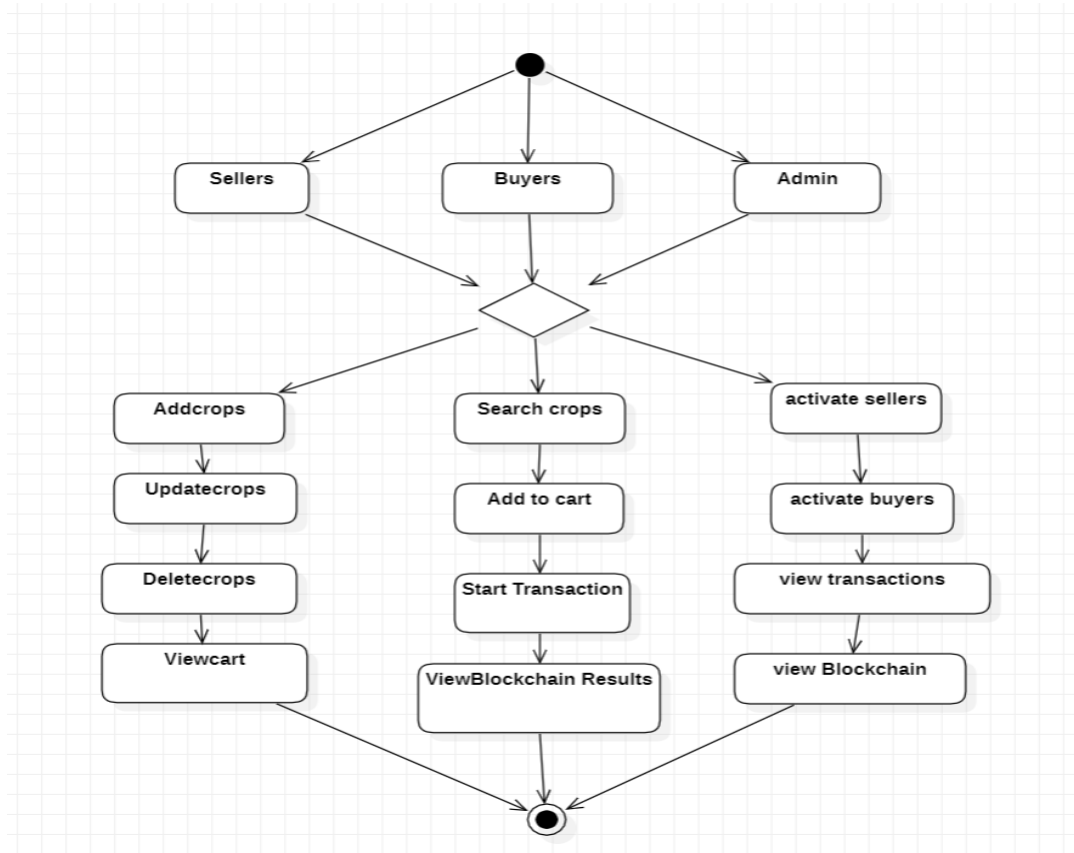


Figure 3.5: Activity Diagram of Blockchain Based Portal For Farmers

4.IMPLEMENTATION

SAMPLE CODE

BuyerUserSide views.py:

```

from django.shortcuts import render,HttpResponse, redirect
from django.contrib import messages
from .forms import BuyerUserRegistrationForm
from .models import BuyerUserRegistrationModel,
BuyerCropCartModels,BuyerTransactionModels,BlockChainTransactionModel
from sellers.models import FarmersCropsModels
from .utility.BlockChainImpl import Blockchain
from django.db.models import Sum
import random

blockchain = Blockchain()
# Create your views here.
def BuyerUserRegisterActions(request):
    if request.method == 'POST':
        form = BuyerUserRegistrationForm(request.POST)
        if form.is_valid():
            print('Data is Valid')
            form.save()
            messages.success(request, 'You have been successfully registered')
            form = BuyerUserRegistrationForm()
            return render(request, 'BuyerUserRegistrations.html', {'form': form})
        else:
            messages.success(request, 'Email or Mobile Already Existed')
            print("Invalid form")
    else:
        form = BuyerUserRegistrationForm()
        return render(request, 'BuyerUserRegistrations.html', {'form': form})
def BuyerUserLoginCheck(request):
    if request.method == "POST":
        loginid = request.POST.get('loginname')
        pswd = request.POST.get('pswd')
        print("Login ID = ", loginid, ' Password = ', pswd)
        try:
            check = BuyerUserRegistrationModel.objects.get(loginid=loginid,
password=pswd)
            status = check.status
            print('Status is = ', status)
            if status == "activated":
                request.session['id'] = check.id

```

```

        request.session['loggeduser'] = check.name
        request.session['loginid'] = loginid
        request.session['email'] = check.email
        print("User id At", check.id, status)
        cartin = checkCartCount(loginid)
        return render(request, 'buyers/BuyerUserHome.html', {'count':cartin})
    else:
        messages.success(request, 'Your Account Not at activated')
        return render(request, 'BuyerLogin.html')
except Exception as e:
    print('Exception is ', str(e))
    pass
    messages.success(request, 'Invalid Login id and password')
return render(request, 'BuyerLogin.html', {})

def BuyerUserHome(request):
    loginid = request.session['loginid']
    cartin = checkCartCount(loginid)
    return render(request, 'buyers/BuyerUserHome.html', {'count':cartin})

def BuyerSearchProductsForm(request):
    loginid = request.session['loginid']
    cartin = checkCartCount(loginid)
    return render(request, "buyers/BuyerSearchProducts.html", {'count':cartin})

def BuyerSearchCropsAction(request):
    if request.method=='POST':
        crpname = request.POST.get('cropname')
        search_data = FarmersCropsModels.objects.filter(cropname__icontains=crpname)
        loginid = request.session['loginid']
        cartin = checkCartCount(loginid)
        return render(request,
'buyers/BuyerSearchResults.html',{'data':search_data,'count':cartin})

def BuyerAddCropsToCart(request):
    crop_id = request.GET.get('cropid')
    crop = FarmersCropsModels.objects.get(id=crop_id)
    sellername = crop.sellername
    cropname = crop.cropname
    price = crop.price
    description = crop.description
    file = crop.file
    buyerUser = request.session['loginid']
    buyeremail = request.session['email']
    cartStatus = 'waiting'

```

```

BuyerCropCartModels.objects.create(buyerusername=buyerUser,buyeruseremail=buyere
mail,sellername=sellername,cropname=cropname, description=description, price=price,
file=file,status=cartStatus)

```

```

    print("Seller name ",sellername)
    search_data = FarmersCropsModels.objects.filter(cropname__icontains=cropname)
    cartin = checkCartCount(buyerUser)
    print("Cart Count = ",cartin)
    loginid = request.session['loginid']
    cartin = checkCartCount(loginid)
    return render(request, 'buyers/BuyerSearchResults.html', {'data':
search_data,'count':cartin})

```

```

def checkCartCount(buyername):
    cartin =
BuyerCropCartModels.objects.filter(buyerusername=buyername,status='waiting').count()
    return cartin

```

```

def BuyerCheckCartData(request):
    buyerName =request.GET.get('buyerUser')
    data = BuyerCropCartModels.objects.filter(buyerusername=buyerName,
status='waiting')
    return render(request,"buyers/BuyerCheckInCart.html",{ 'data':data})

```

```

def BuyerDeleteanItemfromCart(request):
    cropid = request.GET.get('cropid')
    BuyerCropCartModels.objects.filter(id=cropid).delete()
    buyerName = request.session['loginid']
    cartin = checkCartCount(buyerName)
    data = BuyerCropCartModels.objects.filter(buyerusername=buyerName,
status='waiting')
    return render(request, "buyers/BuyerCheckInCart.html", { 'data': data,'count':cartin})

```

```

def startBlockchainProcess(request):
    blockchain = Blockchain()
    t1 = blockchain.new_transaction("Satoshi", "Mike", '5 BTC')
    blockchain.new_block(12346)
    t2 = blockchain.new_transaction("Mike", "Satoshi", '1 BTC')
    t3 = blockchain.new_transaction("Satoshi", "Hal Finney", '5 BTC')
    blockchain.new_block(12345)
    print("Genesis block: ", blockchain.chain)
    return HttpResponse("Block Chain Started")

```

```

def BuyerTotalAmountCheckOut(request):
    buyerName = request.GET.get('buyername')

```

```

    cartstatuc = 'waiting'

```

```

total_price = BuyerCropCartModels.objects.filter(buyerusername=buyerName,
status='waiting').aggregate(Sum('price'))
total_price = total_price['price__sum']
print("Total Price ",total_price)
bank = ('SBI Bank','Union Bank','ICICI Bank','Axis Bank','Canara Bank','HDFC
Bank','FDI Bank','Chase Bank')
recipient = random.choice(bank)
return render(request,
'buyers/BuyerInitiateTransactionForm.html',{ 'buyername':buyerName,'totalPrice':total_price,
'bank':recipient})

```

```

def StartBlockchainTransaction(request):
    if request.method=='POST':
        ## Block Chain Data
        buyername = request.POST.get('buyername')
        totalamount = request.POST.get('totalamount')
        recipientnmae = request.POST.get('recipientnmae')

        #Transaction Data
        cardnumber = request.POST.get('cardnumber')
        nameoncard = request.POST.get('nameoncard')
        cvv = request.POST.get('cvv')
        cardexpiry = request.POST.get('cardexpiry')

        t1 = blockchain.new_transaction(buyername, recipientnmae, totalamount)
        proofId = ".join([str(random.randint(0, 999)).zfill(3) for _ in range(2)])
        blockchain.new_block(int(proofId))
        print("Genesis block: ", blockchain.chain)
        print("T1 is ",t1)
        currentTrnx = blockchain.chain[-1]
        previousTranx = blockchain.chain[-2]
        ### Current Tranasction Details
        c_transactions = currentTrnx.get('transactions')
        c_tnx_Dict = c_transactions[0]

        c_index = currentTrnx.get('index')
        c_timestamp = currentTrnx.get('timestamp')
        c_sender = c_tnx_Dict.get('sender')
        c_recipient = c_tnx_Dict.get('recipient')
        c_amount = c_tnx_Dict.get('amount')
        c_proof = currentTrnx.get('proof')
        c_previous_hash = currentTrnx.get('previous_hash')

        c_dict_rslt =
        {'c_index':c_index,'c_timestamp':c_timestamp,'c_sender':c_sender,'c_recipient':c_recipient,'c_amount':c_amount,'c_proof':c_proof,'c_previous_hash':c_previous_hash}

```

```

sh}

# previous Transaction
p_dict_rslt = { }
p_transactions = previousTranx.get('transactions')
if(len(p_transactions)!=0):
    p_tnx_Dict = p_transactions[0]

    p_index = previousTranx.get('index')
    p_timestamp = previousTranx.get('timestamp')
    p_sender = p_tnx_Dict.get('sender')
    p_recipient = p_tnx_Dict.get('recipient')
    p_amount = p_tnx_Dict.get('amount')
    p_proof = previousTranx.get('proof')
    p_previous_hash = previousTranx.get('previous_hash')

    BuyerTransactionModels.objects.create(buyername=buyername,
totalamount=totalamount,recipientname=recipientname,cradnumber=cardnumber,
nameoncard=nameoncard, cvv=cvv, cardexpiry=cardexpiry)
    p_dict_rslt = {'p_index': p_index, 'p_timestamp': p_timestamp, 'p_sender':
p_sender, 'p_recipient': p_recipient, 'p_amount': p_amount, 'p_proof': p_proof,
'p_previous_hash': p_previous_hash}

BlockChainTransactionModel.objects.create(c_index=c_index,c_timestamp=c_t
imestamp,c_sender=c_sender,c_recipient=c_recipient,
c_amount=c_amount,c_proof=c_proof,c_previous_hash=c_previous_hash,p_inde
x=p_index,
p_timestamp=p_timestamp,p_sender=p_sender,p_recipient=p_recipient,p_amount
=p_amount,p_proof=p_proof,p_previous_hash=p_previous_hash)
    buyer_name = request.session['loginid']
    print('buyername =',buyer_name)
    qs =
    BuyerCropCartModels.objects.filter(buyerusername=buyer_name).update(status='
purchased')
else:
    BuyerTransactionModels.objects.create(buyername=buyername,
totalamount=totalamount,
recipientname=recipientname, cradnumber=cardnumber,
nameoncard=nameoncard, cvv=cvv, cardexpiry=cardexpiry)

BlockChainTransactionModel.objects.create(c_index=c_index,
c_timestamp=c_timestamp, c_sender=c_sender,c_recipient=c_recipient,

c_amount=c_amount, c_proof=c_proof, c_previous_hash=c_previous_hash,
p_index='p_index',p_timestamp='p_timestamp', p_sender='p_sender',
p_recipient="p_recipient", p_amount="p_amount",

```

```

        p_proof="p_proof",p_previous_hash="p_previous_hash")
        buyer_name = request.session['loginid']
        print('buyername =', buyer_name)
        qs=BuyerCropCartModels.objects.filter(buyerusername=buyer_name).update(status='purchased')
    return
    render(request,'buyers/TransactionResults.html',{'c_dict_rslt':c_dict_rslt,'p_dict_rslt':p_dict_rslt})

def BuyerViewPurchasedDetails(request):
    buyer_name = request.session['loginid']
    cartin = checkCartCount(buyer_name)
    data = BuyerCropCartModels.objects.filter(buyerusername=buyer_name, status='purchased')
    return render(request,'buyers/BuyersViewPurchasedData.html',{'data':data,'count':cartin})

def BuyerViewTransactinDetails(request):
    bd_name = request.session['loginid']
    print('buyer_name',bd_name)
    data = BuyerTransactionModels.objects.filter(buyername = '+'bd_name)
    cartin = checkCartCount(bd_name)
    return
    render(request,'buyers/BuyersViewTransactionDetails.html',{'data':data,'count':cartin})

```

Buyer Models:

```
from django.db import models
```

```
# Create your models here.
```

```
class
```

```
    BuyerUserRegistrationModel(mode
ls.Model):name =
models.CharField(max_length=100)
loginid = models.CharField(unique=True,
max_length=100)password =
models.CharField(max_length=100)
mobile = models.CharField(unique=True,
max_length=100)
email = models.CharField(unique=True,
max_length=100)
locality = models.CharField(max_length=100)
addressmodels.CharField(max_length=1000)
city=models.CharField(max_length=100)
state = models.CharField(max_length=100)
status = models.CharField(max_length=100)
```

```
def str_self():
```

```
    return seller.loginid
```

```
    class Meta:
```

```
        db_table = 'BuyersRegistrations'
```

```
class BuyerCropCartModels(models.Model):
```

```
    buyerusername =
models.CharField(max_length=100)
buyeruseremail =
models.CharField(max_length=100)
sellername =
models.CharField(max_length=100)
cropname =
models.CharField(max_length=100)
description =
models.CharField(max_length=100000)
price = models.FloatField()
```

```
file = models.FileField(upload_to='files/')
```

```
cdate =
```

```
models.DateTimeField(auto_now_add=True)
```



```

status = models.CharField(max_length=50)
    def __str__(self):
        return self.buyerusername

    class Meta:
        db_table = "BuyerCartTable"
class
BuyerTransactionModels(models.Model):
    buyername=models.CharField(max_
    length=100)
    totalamount = models.FloatField()
    recipientname =
    models.CharField(max_length=100)
    cradnumber = models.IntegerField()
    nameoncard=models.CharField(max_l
    length=100)
    cvv = models.IntegerField()
    cardexpiry =
    models.CharField(max_length=200) trnx_date
    = models.DateTimeField(auto_now_add=True)

```

```

def class Meta:
    db_table = "BuyerTransactionTable"
class BlockchainTransactionModel(models.Model):
    c_index = models.CharField(max_length=100)
    c_timestamp =
    models.CharField(max_length=100)
    c_sender = models.CharField(max_length=100)
    c_recipient = models.CharField(max_length=100)
    c_amount = models.CharField(max_length=100)
    c_proof = models.CharField(max_length=100)
    c_previous_hash =
    models.CharField(max_length=100)
    p_index = models.CharField(max_length=100)
    p_timestamp =
    models.CharField(max_length=100)
    p_sender = models.CharField(max_length=100)
    p_recipient = models.CharField(max_length=100)
    p_amount = models.CharField(max_length=100)
    p_proof = models.CharField(max_length=100)
    p_previous_hash =
    models.CharField(max_length=100)

```

```

def str_self(id):
    return self.id

```

Sellers side views.py

```

from django.shortcuts import render,HttpResponse
from django.contrib import messages
from .forms import SellerUserRegistrationForm
from .models import SellerUserRegistrationModel, FarmersCropsModels
from django.core.files.storage import FileSystemStorage
from buyers.models import BuyerCropCartModels

# Create your views here.

def SellerUserRegisterActions(request):
    if request.method == 'POST':
        form = SellerUserRegistrationForm(request.POST)
        if
            form.is_vali
            d():
            print('Data
is Valid')
            form.save()
            messages.success(request, 'You have been successfully
            registered')form = SellerUserRegistrationForm()
            return render(request, 'SellerUserRegistrations.html', {'form': form})
        else:
            messages.success(request, 'Email or Mobile Already
            Existed')print("Invalid form")
    else:
        form = SellerUserRegistrationForm()
        return render(request, 'SellerUserRegistrations.html', {'form': form})
def SellerUserLoginCheck(request):
    if request.method == "POST":
        loginid =
        request.POST.get('loginname')
        pswd =
        request.POST.get('pswd')
        print("Login ID = ", loginid, ' Password = ', pswd)
        try:
            check =
            SellerUserRegistrationModel.objects.get(loginid=loginid,
            password=pswd)
            status = check.status
            print('Status is = ',
            status)if status ==
            "activated":

            request.session['id'] = check.id

```

```

        request.session['loggeduser'] =
        check.name
        request.session['loginid'] = loginid
        request.session['email'] =
        check.email print("User id At",
        check.id, status)
        return render(request, 'sellers/SellerUserHome.html', {})
    else:
        messages.success(request, 'Your Account Not at
        activated')return render(request,
        'SellerLogin.html')
    except Exception as e:
        print('Exception is ',
        str(e))pass
        messages.success(request, 'Invalid Login id and
        password')return render(request, 'SellerLogin.html',
        {})
def SellerUserHome(request):
    return render(request, 'sellers/SellerUserHome.html', {})

def SellerAddItemsForm(request):
    return render(request, 'sellers/SellerAddItems.html',{ })

def SellerAddItems ():

    cropname = request.POST.get('cropname')
    price = request.POST.get('price')
    description = request.POST.get('description')
    # let's check if it is a csv file
    if not image_file.name.endswith('.jpg'):
        messages.error(request, 'THIS IS NOT A JPG
        FILE')
        fs = FileSystemStorage()filename = fs.save(image_file.name, image_file)
        detect_filename = fs.save(image_file.name, image_file)uploaded_file_url =
        fs.url(filename)
        loginid=request.session['logini
        d']
        email =request.session['email']
        FarmersCropsModels.objects.create(sellername=loginid,
        selleremail=email,cropname=cropname,price=price,
        description=description,file=uploaded_file_url)
        messages.success(request, 'Crop Data Added
        Success') return render(request,

        'sellers/SellerAddItems.html', {})

```

```

def SellerCommodities(self):
    loginid =
    request.session[loginid]
    data = FarmersCropsModels.objects.filter(sellername=loginid)
    return render(request, 'sellers/SellersCommoditiesData.html', {'data':data})

def SellerUpdateProducts(re):
    cropid =
    request.GET.get('cropid')
    data = FarmersCropsModels.objects.get(id=cropid)
    return render(re, 'sellers/CropsUpdatesbySeller.html', {'data': data})
    return HttpResponse("Update products Working Success")

def SellerDeleteProducts(req):
    cropid =
    req.GET.get('cropid')
    FarmersCropsModels.objects.filter(id=cropid).
    delete()loginid = req.session['loginid']
    data = FarmersCropsModels.objects.filter(sellername=loginid)
    return render(req, 'sellers/SellersCommoditiesData.html', {'data': data})

def SellerCropUpdateAction(request):
    #MyModel.objects.filter(pk=some_value).update(field1='some value')
    cropname = request.POST.get('cropname')
    price =
    request.POST.get('price')
    cropid =
    request.POST.get('cropid')
    description =
    request.POST.get('description')
    image_file = request.FILES['file']
    # let's check if it is a csv file
    if not image_file.name.endswith('jpg'):
        messages.error(request, 'THIS IS NOT A  
JPG FILE')
    fs = FileSystemStorage()
    filename = fs.save(image_file.name, image_file)
    detect_filename = fs.save(image_file.name,
    image_file)uploaded_file_url = fs.url(filename)
    FarmersCropsModels.objects.filter(id=cropid).update(cropname=cropname,
    price=price, description=description, file=uploaded_file_url)

```

```
loginid = request.session['loginid']
data = FarmersCropsModels.objects.filter(sellername=loginid)
return render(request, 'sellers/SellersCommoditiesData.html', {'data': data})
def SellerViewCarts(request):
    sellername = request.session['loginid']
    data = BuyerCropCartModels.objects.filter(sellername=sellername)
    return render(request, 'sellers/SellersViewCart.html', {'data': data})
```

Sellers Side models.py

```

from django.db import models
# Create your models here.
class
    SellerUserRegistrationModel(models
    .Model):name =
    models.CharField(max_length=100)
    loginid = models.CharField(unique=True,
    max_length=100)password =
    models.CharField(max_length=100)
    mobile = models.CharField(unique=True,
    max_length=100)email =
    models.CharField(unique=True,
    max_length=100) locality =
    models.CharField(max_length=100)
    address =
    models.CharField(max_length=100
    0)city =
    models.CharField(max_length=100)
    state =
    models.CharField(max_length=100)
    status =
    models.CharField(max_length=100)
    def __str__(self):
        return self.loginid

    class Meta:
        db_table = 'SellerRegistrations'

class
    FarmersCropDataModels(models.Mode
    l): sellername =
    models.CharField(max_length=100)
    selleremail =
    models.CharField(max_length=100)
    cropname =
    models.CharField(max_length=100)
    price = models.FloatField()
    description =
    models.CharField(max_length=100000)
    file =
    models.FileField(upload_to='files/') cdate
    =

```

```
models.DateTimeField(auto_now_add=True)
```

```
def __str__
    (self):
    return
    self.logi
    nid
```

```
class Meta:
    db_table = "Farmerscroptable"
```

```
class
FarmersCropsModels(models.Model):
    sellername =
models.CharField(max_length=100)
    selleremail =
models.CharField(max_length=100)
    cropname =
models.CharField(max_length=100)
    price = models.FloatField()
    description =
models.CharField(max_length=100000)
    file =
models.FileField(upload_to='files/') cdate
=
models.DateTimeField(auto_now_add=True)
ue)
```

```
def __str__
    (self):
    return
    self.logi
    nid
```

```
class Meta:
    db_table = "FarmersCrops"
```

Base.html

```

<!DOCTYPE html>
{%load static%}
<html lang="en">
  <head>
    <title>Study of Blockchain</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link
href="https://fonts.googleapis.com/css?family=Poppins:200,300,400,500,600,700,800&display=swap" rel="stylesheet">
    <link
href="https://fonts.googleapis.com/css?family=Lora:400,400i,700,700i&display=swap" rel="stylesheet">
    <link
href="https://fonts.googleapis.com/css?family=Amatic+SC:400,700&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="{ %static 'css/open-iconic-bootstrap.min.css'% }">
    <link rel="stylesheet" href="{ %static 'css/animate.css'% }">
    <link rel="stylesheet" href="{ %static 'css/owl.carousel.min.css'% }">
    <link rel="stylesheet" href="{ %static 'css/owl.theme.default.min.css'% }">
    <link rel="stylesheet" href="{ %static 'css/magnific-popup.css'% }">
    <link rel="stylesheet" href="{ %static 'css/aos.css'% }">
    <link rel="stylesheet" href="{ %static 'css/ionicons.min.css'% }">
    <link rel="stylesheet" href="{ %static 'css/bootstrap-datepicker.css'% }">
    <link rel="stylesheet" href="{ %static 'css/jquery.timepicker.css'% }">
    <link rel="stylesheet" href="{ %static 'css/flaticon.css'% }">
    <link rel="stylesheet" href="{ %static 'css/icomoon.css'% }">
    <link rel="stylesheet" href="{ %static 'css/style.css'% }">
  </head>
  <body class="goto-here">
    <nav class="navbar navbar-expand-lg navbar-dark ftco_navbar bg-dark ftco-navbar-light" id="ftco-navbar">
      <div class="container">
        <a class="navbar-brand" href="{ %url 'index'% }">Study of Blockchain Technology in Farmer's</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#ftco-nav" aria-controls="ftco-nav" aria-expanded="false" aria-label="Toggle navigation">
<span class="oi oi-menu"></span> Menu
        </button>

        <div class="collapse navbar-collapse" id="ftco-nav">
          <ul class="navbar-nav ml-auto">

```



```

<li class="nav-item active"><a href="{ %url 'index'% }" class="nav-link">Home</a></li>
<li class="nav-item"><a href="{ %url 'SellerLogin'% }" class="nav-link">Seller</a></li>
<li class="nav-item"><a href="{ %url 'BuyerLogin'% }" class="nav-link">Buyer</a></li>
<li class="nav-item"><a href="{ %url 'AdminLogin'% }" class="nav-link">Admin</a></li>
<li class="nav-item"><a href="{ %url 'SellerRegister'% }" class="nav-link">SellerRegister</a></li>
<li class="nav-item"><a href="{ %url 'BuyerRegister'% }" class="nav-link">BuyerRegister</a></li>

```

```

</ul>

```

```

</div>

```

```

</div>

```

```

</nav>

```

```

<!-- END nav -->

```

```

{ %block contents% }

```

```

{ %endblock% }

```

```

<hr>

```

```

<footer class="ftco-footer ftco-section">

```

```

  <div class="container">

```

```

    <div class="row">

```

```

      </div>

```

```

    <div class="row">

```

```

      <div class="col-md-12 text-center">

```

```

        <p>

```

```

        Copyright &copy;<script>document.write(new Date().getFullYear());</script> All rights
        reserved | This template is made with <i class="icon-heart color-danger" aria-
        hidden="true"></i> by <a href="#" target="_blank">Alex Corporations</a>

```

```

        </p>

```

```

      </div>

```

```

    </div>

```

```

  </div>

```

```

</footer>

```

```

<!-- loader -->

```

```

  <div id="ftco-loader" class="show fullscreen"><svg class="circular" width="48px"
  height="48px"><circle class="path-bg" cx="24" cy="24" r="22" fill="none" stroke-
  width="4" stroke="#eeeeee"/><circle class="path" cx="24" cy="24" r="22" fill="none"
  stroke-width="4" stroke-miterlimit="10" stroke="#F96D00"/></svg></div>

```

```

  <script src="{ %static 'js/jquery.min.js'% }"></script>

```

```

  <script src="{ %static 'js/jquery-migrate-3.0.1.min.js'% }"></script>

```

```

  <script src="{ %static 'js/popper.min.js'% }"></script>

```

```

  <script src="{ %static 'js/bootstrap.min.js'% }"></script>

```

```

  <script src="{ %static 'js/jquery.easing.1.3.js'% }"></script>

```

```

  <script src="{ %static 'js/jquery.waypoints.min.js'% }"></script>

```

```

  <script src="{ %static 'js/jquery.stellar.min.js'% }"></script>

```

```

  <script src="{ %static 'js/owl.carousel.min.js'% }"></script>

```

```
<script src="{%static 'js/jquery.magnific-popup.min.js'%}"></script>
<script src="{%static 'js/aos.js'%}"></script>
<script src="{%static 'js/jquery.animateNumber.min.js'%}"></script>
<script src="{%static 'js/bootstrap-datepicker.js'%}"></script>
<script src="{%static 'js/scrollax.min.js'%}"></script>
<script src="https://maps.googleapis.com/maps/api/js?key=
AlzaSyBVWaKrjvy3MaE7SQ74_uJiULgl1JY0H2s&sensor=false"></script>
<script src="{%static 'js/google-map.js'%}"></script>
<script src="{%static 'js/main.js'%}"></script>
</body>
</html>
```

BuyerUserRegistrations.html

```

{%extends 'base.html'%}
{%load static%}
{%block contents%}
<section class="ftco-section ftco-no-pt ftco-no-pb py-5 bg-light">
  <div class="container py-4">
    <div class="row d-flex justify-content-center py-5">
      <div class="col-md-6">
        <center>
          <h2 style="font-size: 22px;" class="mb-0">Buyer Register Form</h2>
          <span>&nbsp;  </span>
          <form action="{ %url 'BuyerUserRegisterActions'% }" method="POST"
class="text-primary"
          style="width:100%">
            {% csrf_token %}
            <table>
              <tr>
                <td></td>
                <td class="text-primary">Customer Name</td>
                <td>{{ form.name }}</td>
              </tr>
              <tr>
                <td></td>
                <td>Login ID</td>
                <td>{{ form.loginid }}</td>
              </tr>
              <tr>
                <td></td>
                <td>Password</td>
                <td>{{ form.password }}</td>
              </tr>
              <tr>
                <td></td>
                <td>Mobile</td>
                <td>{{ form.mobile }}</td>
              </tr>
              <tr>
                <td></td>
                <td>email</td>
                <td>{{ form.email }}</td>
              </tr>
              <tr>
                <td></td>
                <td>Locality</td>
                <td>{{ form.locality }}</td>
              </tr>
              <tr>
                <td></td>
                <td>Address</td>
                <td>{{ form.address }}</td>
              </tr>
            </table>
          </div>
        </div>
      </div>
    </div>
  </div>
</section>

```

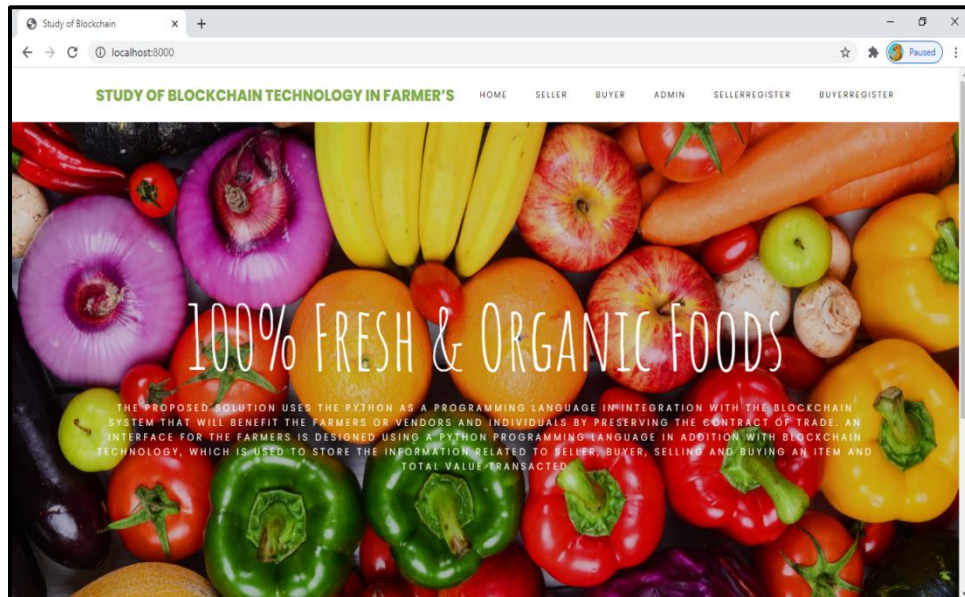
```

        <td></td>
        <td>City</td>
        <td>{{ form.city }}</td>
    </tr>
    <tr>
        <td></td>
        <td>State</td>
        <td>{{ form.state }}</td>
    </tr>
    <tr>
        <td></td>
        <td></td>
        <td>{{ form.status }}</td>
    </tr>
    <tr>
        <td></td>
        <td></td>
        <td>
            <button type="submit" value="Register" class="btn btn-
primary">Register</button>
        </td>
    </tr>
    <tr>
        <td>
            <div class="form-group mt-3">
                <span>&nbsp;</span>
            </div>
        </td>
    </tr>
    <tr>
        <td>
            {% if messages %}
            {% for message in messages %}
            <font color='GREEN'> {{ message }}</font>
            {% endfor %}
            {% endif %}
        </td>
    </tr>
</table>
</form>
</center>
</div>
</div>
</div>
</section>
{%endblock%}

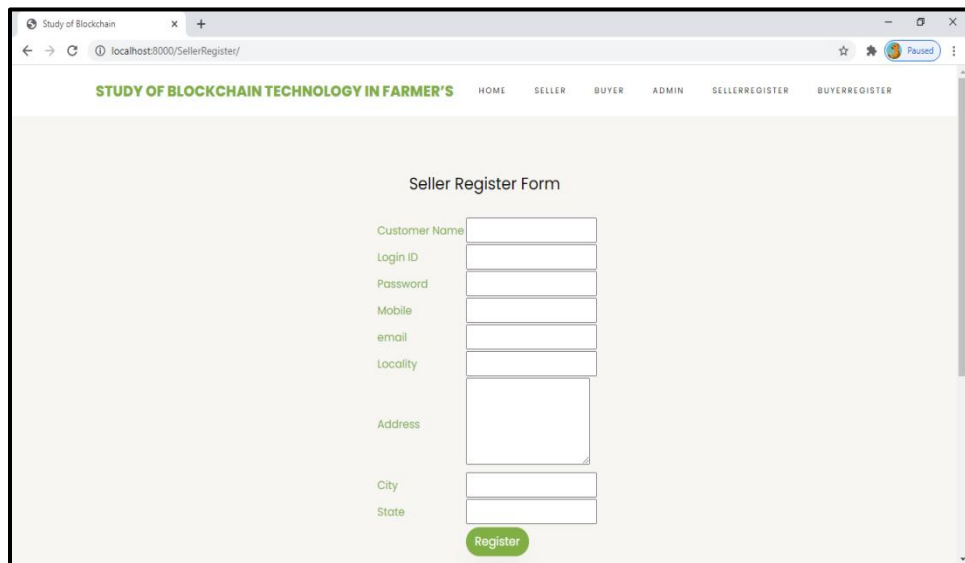
```

5.RESULTS

SCREENSHOTS



Screenshot 5.1: Home Page of Farmer's Portal

A screenshot of a web browser displaying the "Seller Register Form" page. The browser's address bar shows "localhost:8000/SellerRegister/". The page has a light beige background. At the top, the title "STUDY OF BLOCKCHAIN TECHNOLOGY IN FARMER'S" is displayed in green, followed by a navigation menu with links for HOME, SELLER, BUYER, ADMIN, SELLERREGISTER, and BUYERREGISTER. The main content area is titled "Seller Register Form" in a bold, black font. Below the title, there are several input fields for registration: "Customer Name", "Login ID", "Password", "Mobile", "email", "Locality", "Address" (a larger text area), "City", and "State". A green "Register" button is located at the bottom right of the form.

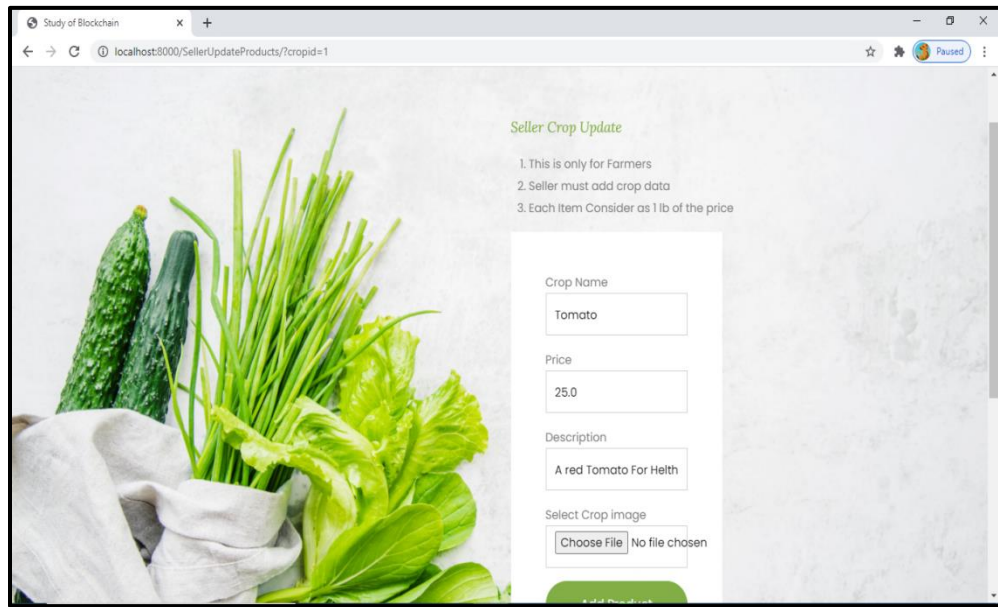
Screenshot 5.2: Seller Register Form

The screenshot shows a web browser window with the address bar displaying 'localhost:8000/BuyerRegister/'. The page title is 'STUDY OF BLOCKCHAIN TECHNOLOGY IN FARMER'S'. The navigation menu includes 'HOME', 'SELLER', 'BUYER', 'ADMIN', 'SELLERREGISTER', and 'BUYERREGISTER'. The main content area is titled 'Buyer Register Form' and contains the following fields: Customer Name, Login ID, Password, Mobile, email, Locality, Address, City, and State. A green 'Register' button is located at the bottom of the form. The Windows taskbar at the bottom shows the search bar and various application icons.

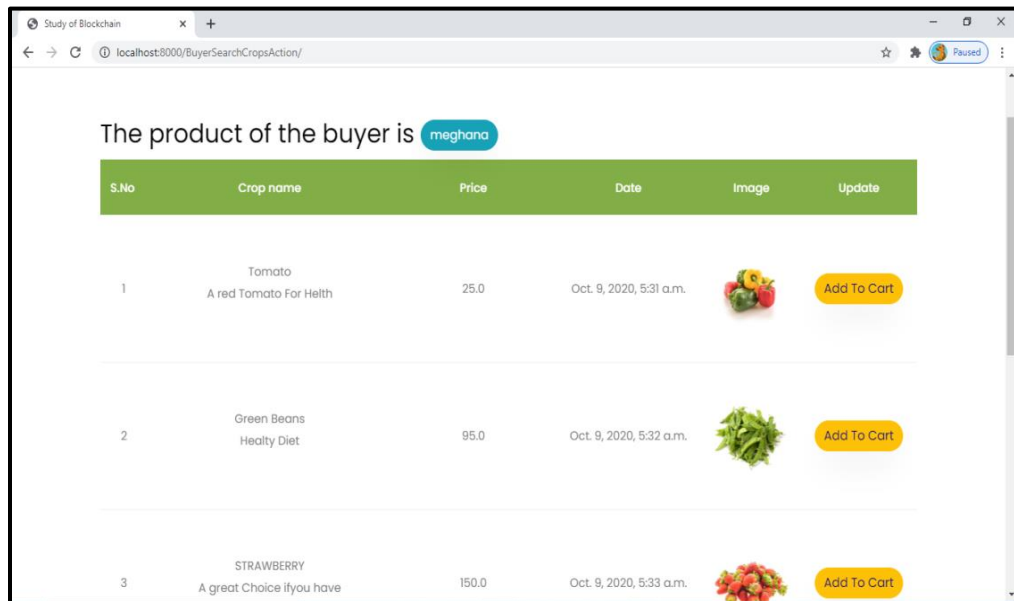
Screenshot 5.3: Buyer Register Form

The screenshot shows a web browser window with the address bar displaying 'localhost:8000/SellerAddItemsForm/'. The page title is 'STUDY OF BLOCKCHAIN TECHNOLOGY IN FARMER'S'. The navigation menu includes 'HOME', 'SELLER', 'BUYER', 'ADMIN', 'SELLERREGISTER', and 'BUYERREGISTER'. The main content area is titled 'Add Items Details' and includes a list of instructions: '1. This is only for Farmers', '2. Seller must add crop data', and '3. Each Item Consider as 1 KG of the price'. Below the instructions is a form with the following fields: Enter Crop name, Enter Price, Enter Descriptions, and a file upload section with a 'Choose File' button and 'No file chosen' text. A green 'Add Product' button is located at the bottom of the form. The background of the page features a photograph of fresh vegetables, including cucumbers, chives, and leafy greens.

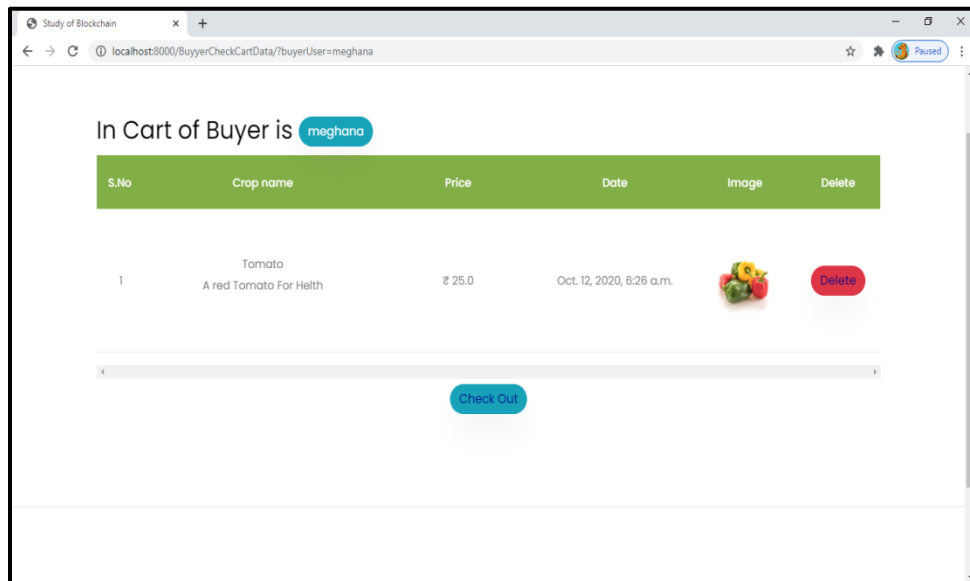
Screenshot 5.4: Seller Add Crop Form



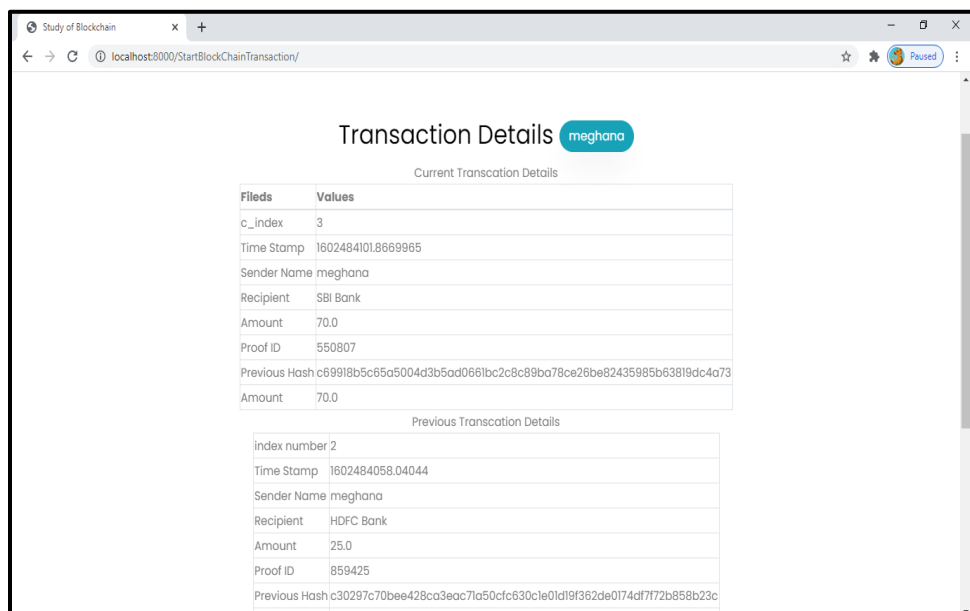
Screenshot 5.5 : Seller Update Crop Page



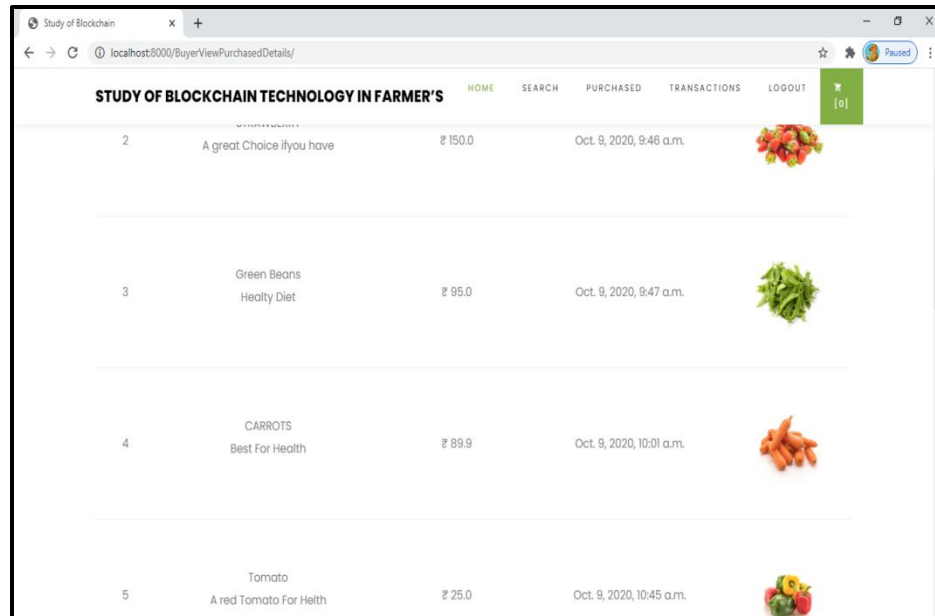
Screenshot 5.6: Buyer Search Results Page



Screenshot 5.7: Buyer Cart Page



Screenshot 5.8 Buyer Transaction Details View Page



Screenshot 5.9: Buyer Purchased Crops View

S.No	C_index	C_timestamp	C_Sender	C_Recipient	C_Amount	C_Proof	C_Currenthash
1	2	1602325602.7917922	harish	HDFC Bank	50.0	286923	38e7551d5e22862e689bcbcf688ba339cd5c84
2	3	1602325649.1655002	harish	HDFC Bank	300.0	486421	9e03e15112b4a62cbd43149078bccd80c9e93c
3	4	1602325701.1176872	ramesh	Canara Bank	495.0	14272	ae83dcd03eda83691aa254c9ce68231a29aaa2
4	2	1602326025.9633133	ramesh	Canara Bank	25.0	410301	b79d3f83920ab7c4f7db34e8d2b60dc358b8b8

Screenshot 5.10: Admin View of Transactions

6.TESTING

6.TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTING

6.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.2.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Integration tests demonstrate that although the components were individually satisfactory, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

6.2.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked. Organization and preparation of functional tests is focused on requirements, key functions, or special test cases

6.3 TEST CASES

6.3.1 CLASSIFICATION

Test case ID	Test case name	Expected Result	Result	Remarks(IF Fails)
1	Sellers Registration	If Seller User registration successfully	Pass	If already user email exist then it fails.
2	Seller User Login	If Username and password is correct then it will getting valid page.	Pass	Un Register Users will not logged in.
3	Buyers Registrations	If Buyers User registration successfully.	Pass	If already user email exist then it fails.
4	Buyer User Login	If User name and password is correct then it will getting valid page.	Pass	UnRegister Users will not logged in.
5	Adding crops for sell	Seller user will add the crops price and images	Pass	If model object not defined the failed
6	Buyer Search the crops	Based on seller crops the search nearly matching crops will be displayed	Pass	If model not properly trained the failed
7	Start the transaction	When Buyer user start the transaction then a block added to block chain	Pass	Block Chain object must be defined
10	View previous transactions	In the buyer transaction can view its previous transaction details	Pass	Blockchain object required

8	Admin login	Admin can login with his login credential. If success he get his home page	Pass	Invalid login details will not allowed here
9	Admin can activate the register users	Admin can activate the register user id	Pass	If user id not found then it won't login.

7.CONCLUSION

7.CONCLUSION & FUTURE SCOPE

7.1PROJECT CONCLUSION

Blockchain Technology in the field of agriculture can bring revolutionary enhancement in the area of maintaining farmers data securely, ensuring the security of transactions, and sale price of crops. In this project the block chain-based portal is involved in dealing with the issue of demand and sale price of crops which in result ensure crop security to farmers as well as to get fair price of the crop. A farmer can register and sell his crops, recording a transaction on a block chain at a point when buyers commit to buy a farmer's crop. This transaction is capable of recording crop details, the price at which it is committed to buying and quantity of crop purchased. This immutable nature of block chain technology will enable the farmers to get a legitimate price of crop and reduce the cost of operation for selling and buying crops when compared to traditional methods.

7.2 FUTURE SCOPE

Smart Contracts for Transactions : Implementing smart contracts for automated and secure transactions between farmers, suppliers, and buyers. This can streamline payment processes and reduce fraud.

Government Partnerships_: Collaborating with government agencies to support subsidy distribution, regulatory compliance, and sustainable agriculture initiatives.

Crop Insurance : Integrating crop insurance services within the portal to help farmers protect their crops and livelihoods against unforeseen events

8.BIBLIOGRAPHY

8. BIBLIOGRAPHY

8.1 REFERENCES

- [1] Hileman, Garrick, and Michel Rauchs. "2017 Global Blockchain Benchmarking Study." Available at SSRN 3040224 (2017).
- [2] Yadav, Vinay Surendra, and A. R. Singh. "A Systematic Literature Review of Blockchain Technology in Agriculture."
- [3] Ghosh, Soumalya, A. B. Garg, Sayan Sarcar, PSV S. Sridhar, Ojasvi Maleyvar, and Raveesh Kapoor. "Krishi-Bharati: An Interface for Indian Farmers". In Proceedings of the 2014 IEEE Students' Technology Symposium, pp. 259-263. IEEE, 2014.
- [4] Zhu, Xingxiong, and Dong Wang. "Research on Blockchain Application for E-Commerce, Finance and Energy." In IOP Conference Series: Earth and Environmental Science, vol. 252, no. 4, p. 042126. IOP Publishing, 2019.

8.2 GITHUB LINK

<https://github.com/sahithi-padigela/MINI-PROJECT>

