AK FRESH FRUITS WEBSITE

By

A.KANI ALAGAR

REGISTRATION NO: 952622622008

Of

S.VEERASAMY CHETTIAR COLLEGE OF ENGINEERING &

TECHNOLOGY, PULIANGUDI.

PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements

For the award of the degree

Of

MASTER OF COMPUTER APPLICATIONS

ANNA UNIVERSITY

CHENNAI – 600 025

AUGUST,2024

BONAFIDE CERTIFICATE

Certified that this Project Report titled **(AK Fresh Fruits Website)** is the Bonafide work of **Mr. A.KANI ALAGAR (952622622008)** who carried out the Project Report work under my supervision. Certified further, that to the best of our knowledge the work reported here in does not from part of any other Project Report on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Supervisor
Mrs. S.Karthika, MCA.,
Assistant Professor,
Department of Computer Applications
S.Veerasamy Chettiar College of Engg.Tech,
Puliangudi.

Head of the Department
Ms. SHIRIN AYESHA MARIYAM, ME,
Head of the Department,
Department of Computer Applications
S.Veerasamy Chettiar College of Engg.Tech,
Puliangudi.

Submitted for the viva-voce examination held at **S.Veerasamy Chettiar College of Engineering and Technology, Puliangudi** on _____

Internal Examiner **External Examiner**

DECLARATION

I hereby state that the thesis submitted for the degree of

MASTER OF COMPUTER APPLICATIONS

In

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

On

“AK Fresh Fruits Website"

Is my original work and that it has not previously formed the basis for the award of any

Degree, Diploma, Associate ship, Fellowship or any other similar title.

ACKNOWLEDGEMENT

First of all we thank God for his abundant grace and countless blessings in making this Project Report work successful.

We wish to express our sincere thanks to Principal **Dr. K. MUTHULAKSHMI, M.E., PH.D.,** S. Veerasamy Chettiar College of Engineering and Technology, Puliangudi for giving the opportunity to do my Project Report work in the concerned organization.

We express our heartfelt thanks and sincere gratitude to **Ms. SHIRIN AYESHA MARIYAM, ME,** Head of the Department, Department of Computer Applications, S.Veerasamy Chettiar College of Engineering and Technology, Puliangudi, who facilitated us to complete my Project Report successfully.

We express our deep sense of gratitude to **Mrs. S.Karthika, MCA.,** Assistant Professor, Department of Computer Applications, S.Veerasamy Chettiar College of Engineering and Technology, Puliangudi, for her guidance, valuable suggestions and support.

We also wish to record our heartfelt thanks to all the staff members of our MCA Department, parents, friends who have rendered direct or indirect help to complete the Project Report.

ABSTRACT

Nowadays farmers are unable to set their own prices for their Fruits because the farmers give their Fruits to contractors in the market. Those contractors buy Fruits from farmers at a lower price and sell them to customers at a higher price. Thus people do not offered Fruits at the right price. Moreover, the farmers lose their investment most of the time to sell lower price to the Broker.

This project address the issues of farmers to sell the nominal price to the customer. In this aspect "AK Fresh Fruits Website" created to connect farmers and customers using Web application. By using this Website farmer fix their vegetable selling price. Customers directly order Fruits through this Website. Web application developed in Python and Django.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 SCOPE OF THE PROJECT

Today mobile devices are used commonly by everyone, including the farmers and countryside people. Agriculture is the support of Indian economy so information sharing to the knowledge intensive agriculture area is upgraded by mobile-enabled information services and speedy growth of mobile telephony. Mobile application provides varied information services to farmers which are helpful for management, controlling and monitoring of the farm.

Mobile app is very helpful for farmers to increase their farming to yield more profit. This paper explores how Mobile Apps of agricultural services have impacted the farmers in their farming activities and which more innovative agriculture services will provide through Mobile App.

A contractor is someone who is hired to build something. Based on the contracting terms, generally the contractor is responsible for providing all the equipment needed to run the project.

Farmers are the backbone of our society. They are the ones who provide us all the food that we eat. As a result, the entire population of the country depends upon farmers. Be it the smallest or the largest country. Because of them only we are able to live on the planet.

Nowadays farmers are unable to set their own prices for their vegetables because the farmers give their vegetables to contractors in the market. Those contractors buy Vegetables from farmers at a lower price and sell them to customers at a higher price. Thus people do not offered vegetables at the right price.

Moreover, the farmers lose their investment most of the time to sell lower price to the Broker. This Project address the issues of farmers to sell the nominal price to the customer. In this aspect farmers market app created to connect

farmers and customers using mobile application. By using this app farmer fix their vegetable selling price. Customers directly order vegetable through this app. Mobile application developed in android studio and Backend connected with Firebase.

1.2 OBJECTIVES OF THE PROJECT

- Provide a user-friendly and intuitive interface for users to browse and purchasemobile devices and services.
- Ensure a responsive design that adapts to different screen sizes, making the websiteaccessible from various devices.
- Implement a secure and reliable payment system to facilitate online transactions.
- Offer efficient customer support channels for addressing user inquiries and service requests.
- Create a robust and scalable database structure to store and manage relevant data effectively

CHAPTER 2

SYSTEM STUDY

2.1 FEASIVILITY STUDY

The feasibility of the project is analyzed in this phase and 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. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

2.1.1 ECONOMICAL FEASIBILITY

2.1.2 TECHNICAL FEASIBILITY

2.1.3 SOCIAL FEASIBILITY

2.1.1 ECONOMICAL FEASIBILITY

An organization makes good investment on the system. So, they should be worth full for the amount they spend in the system. Always the financial benefit and equalsor less the cost of the system, but should not exceed the cost.

- The cost of investment is analyzed for the entire system
- The cost of Hardware and Software is also noted.
- Analyzing the way in which the cost can be reduced

Every organization wants to reduce their cost but at the same time quality of the Service should also be maintained. The system is developed according the estimation of the cost made by the concern. In this project, the proposed system will definitely reduce the cost and also the manual work is reduced and speed of work is also increased.

2.1.2 TECHNICAL FEASIVILITY

The Technical feasibility is the study of the software and how it is included in the study of our project. Regarding this there are some technical issues that should be noted they are as follows:

- Is the necessary technique available and how it is suggested and acquired?
- Does the proposed equipment have the technical capacity to hold the datarequired using the new system?
- Will the system provide adequate response that is made by the requester atan periodic time interval
- Can this system be expanded after this project development
- Is there a technique guarantees of accuracy, reliability in case of access of data and security

The technical issues are raised during the feasibility study of investigating our System. Thus, the technical consideration evaluates the hardware requirements, software etc. This system uses Android as front end and MySQL as back end. They also provide sufficient memory to hold and process the data. As the company is going to install all theprocess in the system it is the cheap and efficient technique.

This system technique accepts the entire request made by the user and the response is done without failure and delay. It is a study about the resources available and how they are achieved as an acceptable system. It is an essential process for analysis and definition of conducting a parallel assessment of technical feasibility.

Though storage and retrieval of information is enormous, it can be easily handledby MySQL. As the MySQL can be run in any system and the operation does not differ from one to another. So, this is effective.

2.1.3 SOCIAL FEASIVILITY

Proposed project will be beneficial only when they are turned into an information system and to meet the organization operating requirements. The following issues are considered for the operation:

- Does this system provide sufficient support for the user and the management?
- What is the method that should be used in this project?
- Have the users been involved in the planning and development of the projects?
- Will the proposed system cause any harm, bad result, loss of control and accessibility of the system will lost?

Issues that may be a minor problem will sometimes cause major problem in the operation. It is the measure of how people can able to work with the system. Finding out the minor issues that may be the initial problem of the system. It should be a user- friendly environment. All these aspect should be kept in mind and steps should be taken for developing the project carefully.

Regarding the project, the system is very much supported and friendly for the user. The methods are defined in an effective manner and proper conditions are given in other to avoid the harm or loss of data. It is designed in GUI interface, as working will beeasier and flexible for the user.

CHAPTER 3

SYSTEM ANALYSIS

The objective of this project is to service and maintain our mobiles and devices through online website. Our web site works is a more convenient method to customers to get the online services. This is the attraction of this project. It is more efficient to the customers.

3.1 EXISTING SYSTEM

1. User Registration/Login:

- Users register on the platform by providing basic details or log in if already registered.

2. Browse Vegetables:

- Users can browse through a variety of vegetables available on the platform.
- Vegetables are categorized based on type, price, seasonality, etc.

3. Product Details:

- Clicking on a vegetable displays detailed information including price, weight, origin, and any special offers.

4. Add to Cart:

- Users can add vegetables to their shopping cart for purchase.

5. Checkout:

- Users proceed to checkout where they confirm their order and provide delivery details.

6. Payment:

- Payment is processed securely through various methods like credit/debit cards, netbanking, or digital wallets.

7. Order Tracking:

- Users can track the status of their order from processing to delivery.

8. Delivery:

- Vegetables are delivered to the specified address within the promised timeframe.

9. Feedback and Reviews:

- Users can provide feedback and reviews on the quality of vegetables and the overallshopping experience.

3.2 PROPOSED SYSTEM

1. Improved User Experience:

- Enhance the user interface for easier navigation and faster loading times.
- Implement responsive design for seamless shopping across devices.

2. Personalization:

- Introduce personalized recommendations based on past purchases, preferences, and seasonal availability.

3. Expanded Product Range:

- Increase the variety of vegetables offered, including organic and exotic options.
- Partner with local farmers and vendors to source fresh produce.

4. Integration with Social Media:

- Allow users to share their purchases on social media platforms, facilitating word-of-mouth marketing.

5. Loyalty Program:

- Implement a loyalty program where users earn points for each purchase, redeemable for discounts or freebies.

6. Subscription Service:

- Offer subscription options for regular vegetable deliveries, with customizablefrequency and content.

7. Live Chat Support:

- Provide real-time assistance to users through a live chat feature for any queries or concerns.

8. Sustainability Initiatives:

- Promote eco-friendly practices such as biodegradable packaging and supportingsustainable farming methods.

CHAPTER 4

SYSTEM SPECIFICATION

4.1 HARDWARE REQUIREMENTS

Processor	:	Any Processor above 500 MHz
RAM	:	4.00 GB
Hard Disk	:	128 GB

4.2 SOFTWARE REQUIREMENTS

Operating system	:	Windows 10
Front-End	:	HTML, CSS
Back-End	:	Python
Frame Work	:	Django
Data Base	:	MY-SQL

4.3. ABOUT SOFTWARE

4.4.1 FRONT END: HTML

Hypertext Mark-up Language is the standard mark-up language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

HTML is used by the browser to manipulate text, images, and other content, to display it in the required format.HTML elements are the building blocks of HTML pages, With HTML constructs, images, and other objects such as interactive forms may be embedded into the rendered page.HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes, and other items.

HTML elements are delineated by tags, written using angle brackets. Tags such as surround and provide information about document text and may include other tags as sub-elements.

HTML page structure: The basic structure of an HTML page is laid out below, It contains the essential building-block elements (i.e., doc type declaration, html, head, title, and body elements) upon which all webpages are created.

HTML5 can be used to write web applications that still work w hen you're not connected to the net; to tell websites where you are physically located; to handle h igh-definition video; and to deli ver extraordinary graphics.

The core objectives of HTML5 are to offer increased m ult imedia support and make the coding m uch easier to read and u nderstand for both people and machi nes. HTML5 coding is clear,simple, anddescriptive.

HTML5 also makes placi ng audio and video content a breeze.

ADVANTAGES:

- ·HTML isused to build websites.
- Itis supported by all browsers.
- Itcan be integrated with other languages like CSS, JavaScript etc.

4.4.2 FRONT END: CSS

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.CSS handles the look and feel part of a web page. Usi ng CSS, you can control the color of the tex t, the style of fonts, the spacing between paragraphs, how colu mns are sized and laid out, w hat background images or colors are used, layout designs, and variations in display for differen t devices and screen sizes as well as a variety of other effects.

CSS is easy to learn and u nderstand bu t it provides powerful control over the presen tation of an HTML documen t. Most comn1on ly, CSS is combined wi th the mark-up languages HTML or XHTML.

Types of CSS: Cascad ing Style Sheet (CSS) is used to set the style in web pages that con tain HTML elemen ts. It sets the background color, font-size, fon t-famil y, color etc. Property of elements on a web page. There are three types of CSS which are gi ven below:

- Inline CSS
- Internal or Embedded CSS
- External CSS

Inline CSS: Inline CSS con tains the CSS property in the body section attached with element is known as inline CSS. This kind of style is specified within an HTML tag usi ng the style attribute.

Internal CSS: This can be used when a single HTML docu men t must be styled u niquely.The CSS rule set shoul d be within the HTML file in the head section i.e. the CSS is embedded within the HTML file.

External CSS: External CSS contai ns separate CSS file which contains only style property with the help of tag attributes (For example class, id, heading . . . etc.) CSS property written in a separate file wi th .CSS extension and should be linked to 25 HTML docu ment usi ng link tag . This means that for each elemen t, style can be set only once and that will be applied across web pages.

Below is the HTML file that is making u se of the created external style sheet.

ADVANTAGES OF CSS

- Create Stunning Web Site: CSS handles the look and feel part of a web page. Usi ng CSS, you can control the colour of the text, the style of fonts, the spacing between paragraphs, how colu mns are sized and laid out, w hat background images or colours are used, layou t designs, and variations in display for different devices and screen sizes as well as a variety of other effects.
- Become a web designer: If you wan t to start a career as a professional web designer,
- HTML and CSS designing is a must skill.
- Control web: CSS is easy to learn and u nderstan d bu t it provides powerful con trol over the presentation of an HTML document. Most commonl y, CSS is com bi ned with the mark up languages HTML or XHTML.
- Learn other languages: Onceyou u nderstan d thebasic of HTML and CSSthen other related technologies like JavaScript, React JS, or angular are become easier.

4.4.3 BACKEND: Python

Python is a widely used programming language that offers several unique features and advantages compared to languages like Java and C++. Our Python tutorial thoroughly explains Python basics and advanced concepts, starting with installation, conditional statements, loops, built-indatastructures, Object-Oriented
Programming, Generators, Exception Handling, Python RegEx, and many other concepts. This tutorial is designed for beginners and working professionals.

In the late 1980s, Guido van Rossum dreamed of developing Python. The first version of Python 0.9.0 was released in 1991. Since its release, Python started gaining popularity.

According to reports, Python is now the most popular programming language among developers because of its high demands in the tech realm.

- **FEATURES OF JAVAEasy to use and Read -** Python's syntax is clear and easy to read, making it an ideal language for both beginners and experienced programmers. This simplicity can lead to faster development and reduce the chances of errors.
- **Dynamically Typed** - The data types of variables are determined during run-time. We do not need to specify the data type of a variable during writing codes.
- **High-level** - High-level language means human readable code.
- **Compiled and Interpreted** - Python code first gets compiled into bytecode, and then interpreted line by line. When we download the Python in our system form org we download the default implement of Python known as CPython. CPython is considered to be Complied and Interpreted both.
- **Garbage Collected** - Memory allocation and de-allocation are automatically managed. Programmers do not specifically need to manage the memory.
- **Purely Object-Oriented** - It refers to everything as an object, including numbers and strings.
- **Cross-platform Compatibility** - Python can be easily installed on Windows, macOS, and various Linux distributions, allowing developers to create software that runs across different operating systems.
- **Rich Standard Library** - Python comes with several standard libraries that provide ready-to-use modules and functions for various tasks, ranging from **web development** and **data manipulation** to **machine learning** and **networking**.
- **Open Source** - Python is an open-source, cost-free programming language. It is utilized in several sectors and disciplines as a result.

Python has many *web-based assets*, *open-source projects*, and *a vibrant community*. Learning the language, working together on projects, and contributing to the Python ecosystem are all made very easy for developers.

Because of its straightforward language framework, Python is easier to understand and write code in. This makes it a fantastic programming language for novices. Additionally, it assists seasoned programmers in writing clear and error-free code.

Python has many third-party libraries that can be used to make its functionality easier. These libraries cover many domains, for example, web development, scientific computing, data analysis, and more.

4.4.4 DATABASE: MySQL

MySQL is an open-source relational database management system. As with other relational databases, MySQL stores data in tables made up of rows and columns. Users can define, manipulate, control, and query data using Structured Query Language, more commonly known as SQL. MySQL's name is a combination of "My," the name of MySQL creator Michael Wideness's daughter, and "SQL". The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

ADVANTAGES OF MYSQL:

Scalability: MySQL can handle high volumes of data and scale effectively to meet the needs of growing applications. It supports various replication and clustering techniques, allowing for easy scaling across multiple servers.

Performance: MySQL is designed to deliver excellent performance. It is optimized for quick response times, efficient data storage, and high-speed data processing. With proper configuration and indexing, MySQL can handle large datasets and complex queries efficiently.

Reliability and Stability: MySQL is known for its reliability and stability. It has a proven track record in production environments and is used by numerous organizations worldwide. It offers features like ACID (Atomicity, Consistency, Isolation and Durability) compliance and crash recovery mechanisms, ensuring data integrity and availability.

Security: MySQL provides various security features to protect data and ensure the privacy of sensitive information. It supports user authentication, access control, encrypted connections, and data encryption at rest to prevent unauthorized access and safeguard data.

Flexibility: MySQL supports various platforms, operating systems, and programming languages, making it highly flexible and compatible with different environments. It also integrates well with other tools and frameworks commonly used in web development.

Cost-effective: MySQL is an open-source database, which means it is free to use and distribute. It eliminates the need for expensive licensing fees, making it a cost-effective option for businesses and individuals.

Large Community and Support: MySQL has a vast and active community of developers and users who contribute to its development and provide support. The community-driven nature ensures regular updates, bug fixes, and a wealth of resources, including documentation, forums, and online tutorials.

4.4.5 FRAME WORK: DJANGO

Django is a web application framework written in Python programming language. It is based on MVT (Model View Template) design pattern. The Django is very demanding due to its rapid development feature. It takes less time to build application after collecting client requirement.

4.4.5.1 FEAURES OF DJANGO

- Rapid Development
- Secure
- Scalable
- Fully loaded
- Versatile
- Open Source
- Vast and Supported Community

CHAPTER 5

SYSTEM DESIGN

5.1 DATA FLOW DIAGRAM

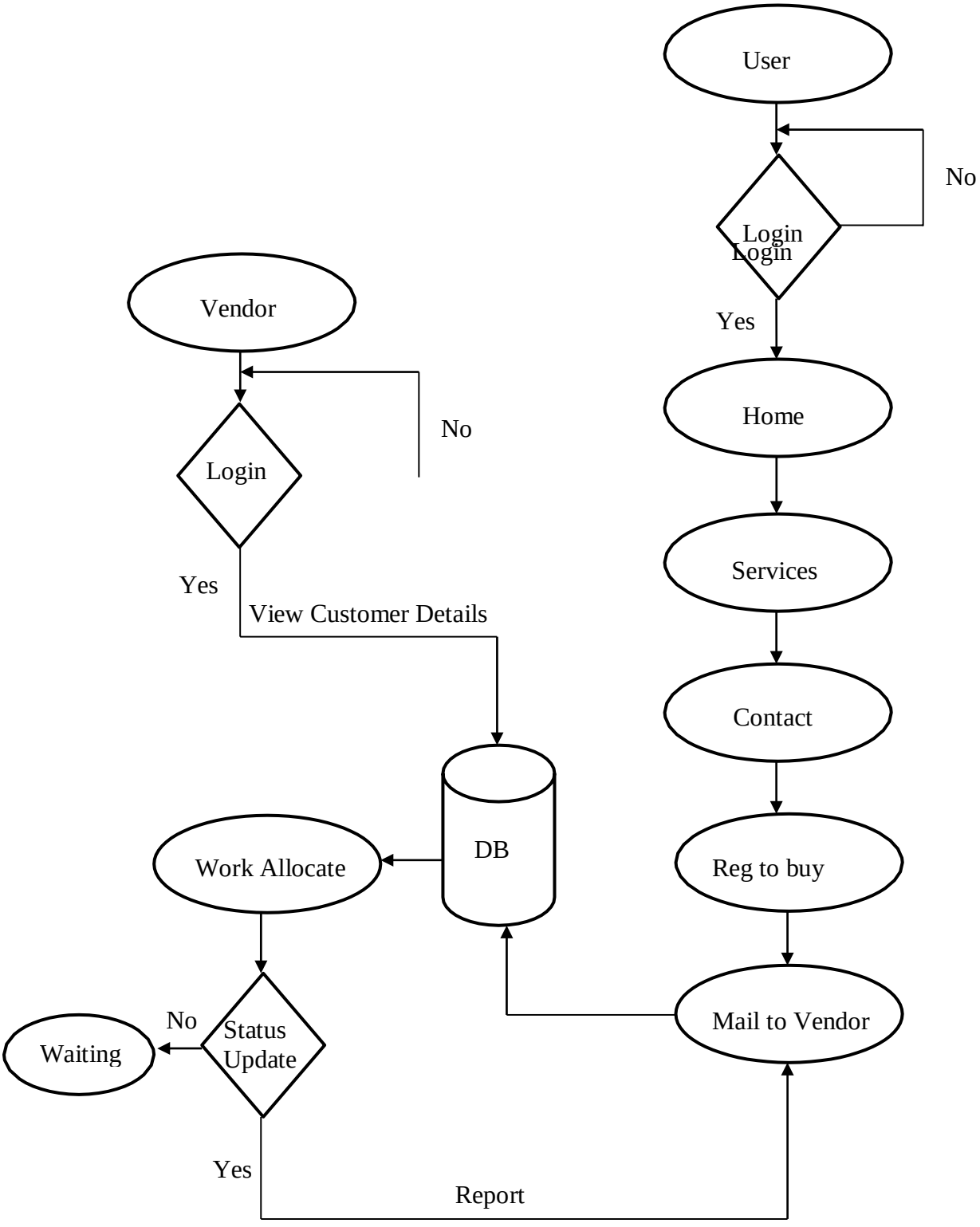

Fig 5.1.1 Data Flow Diagram

5.2 USE CASE DIAGRAM:

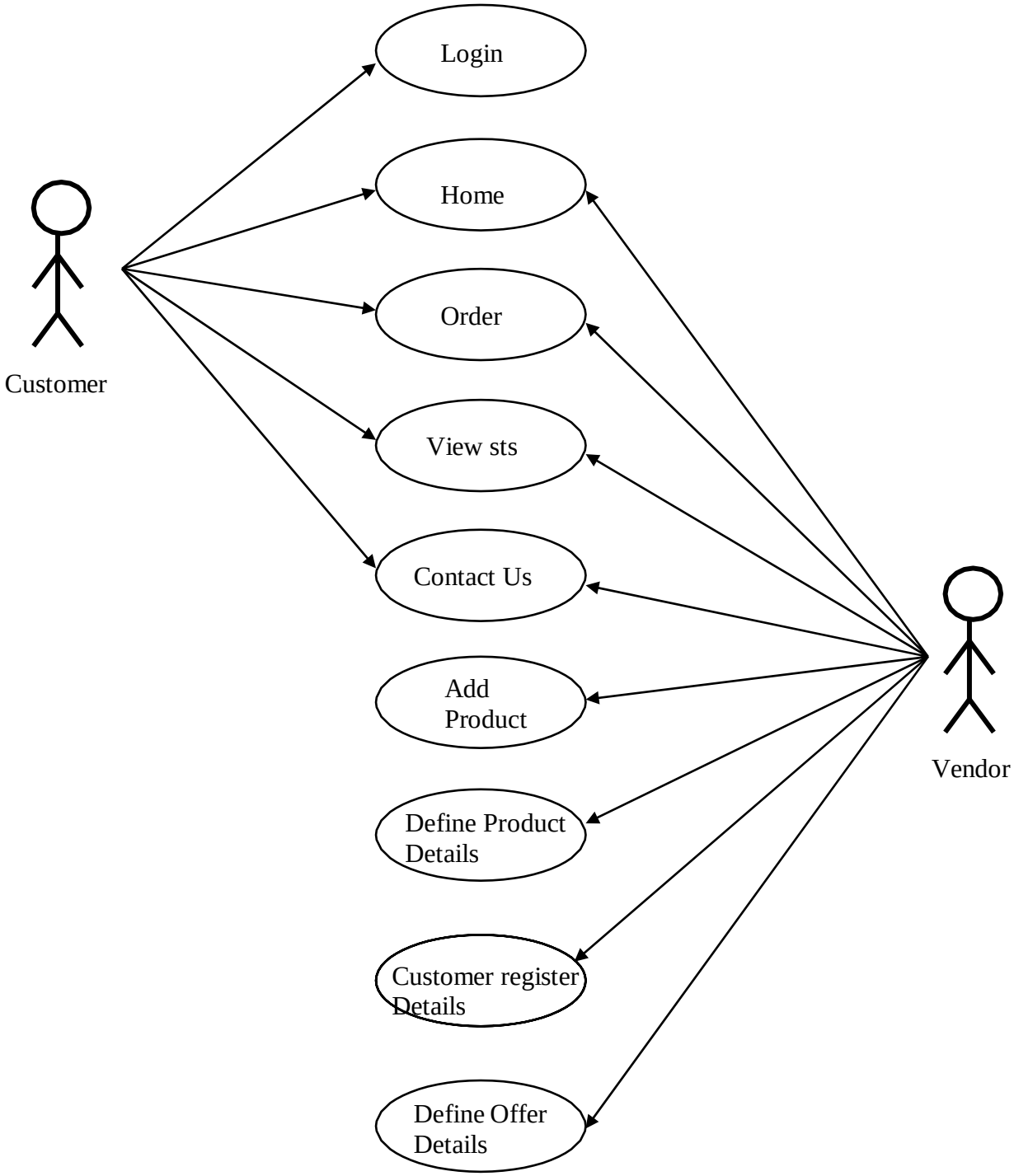

Fig 5.2.1 Use case diagram

5.3 CLASS DIAGRAM

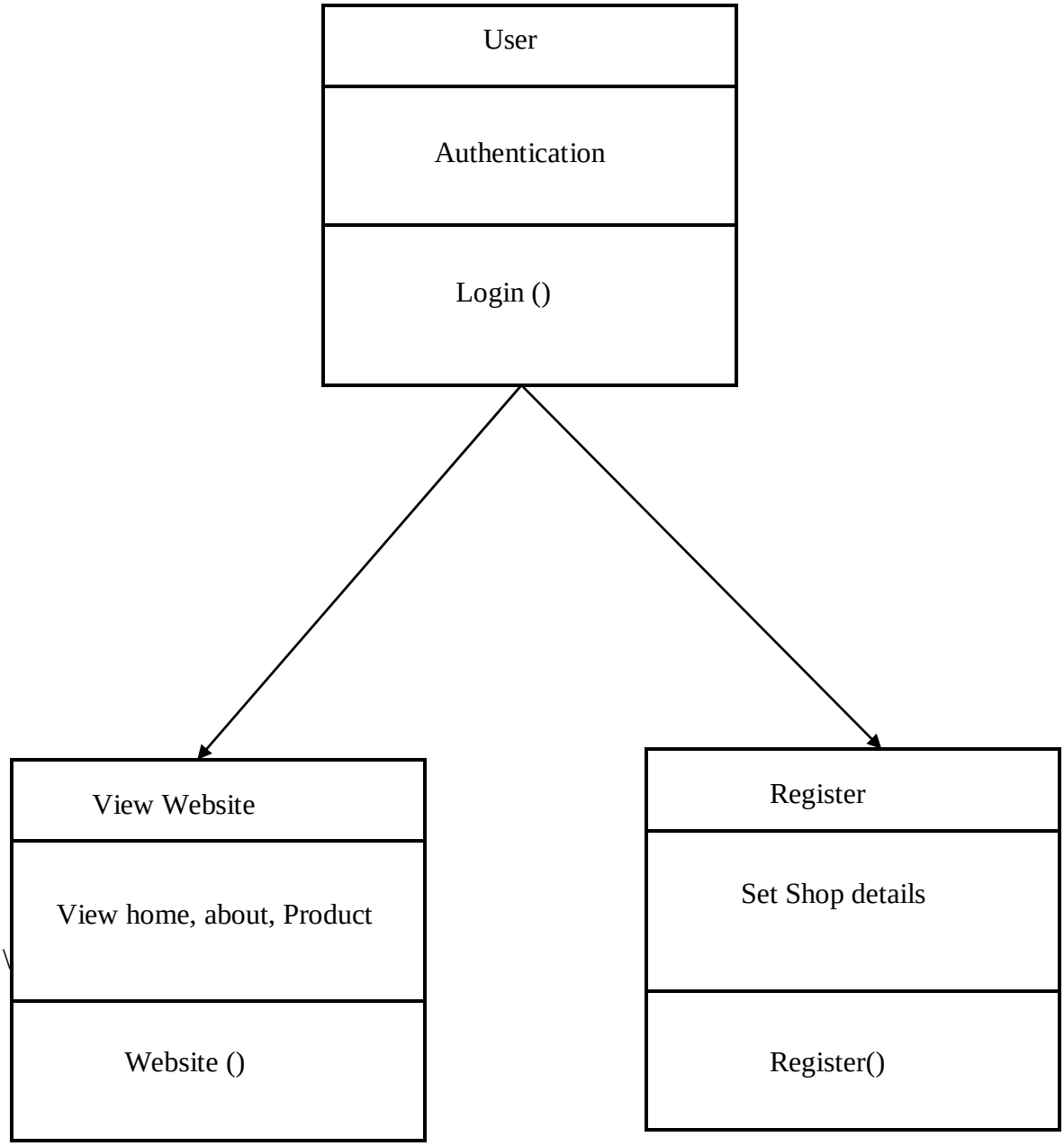

Fig 5.3.1 User class diagram

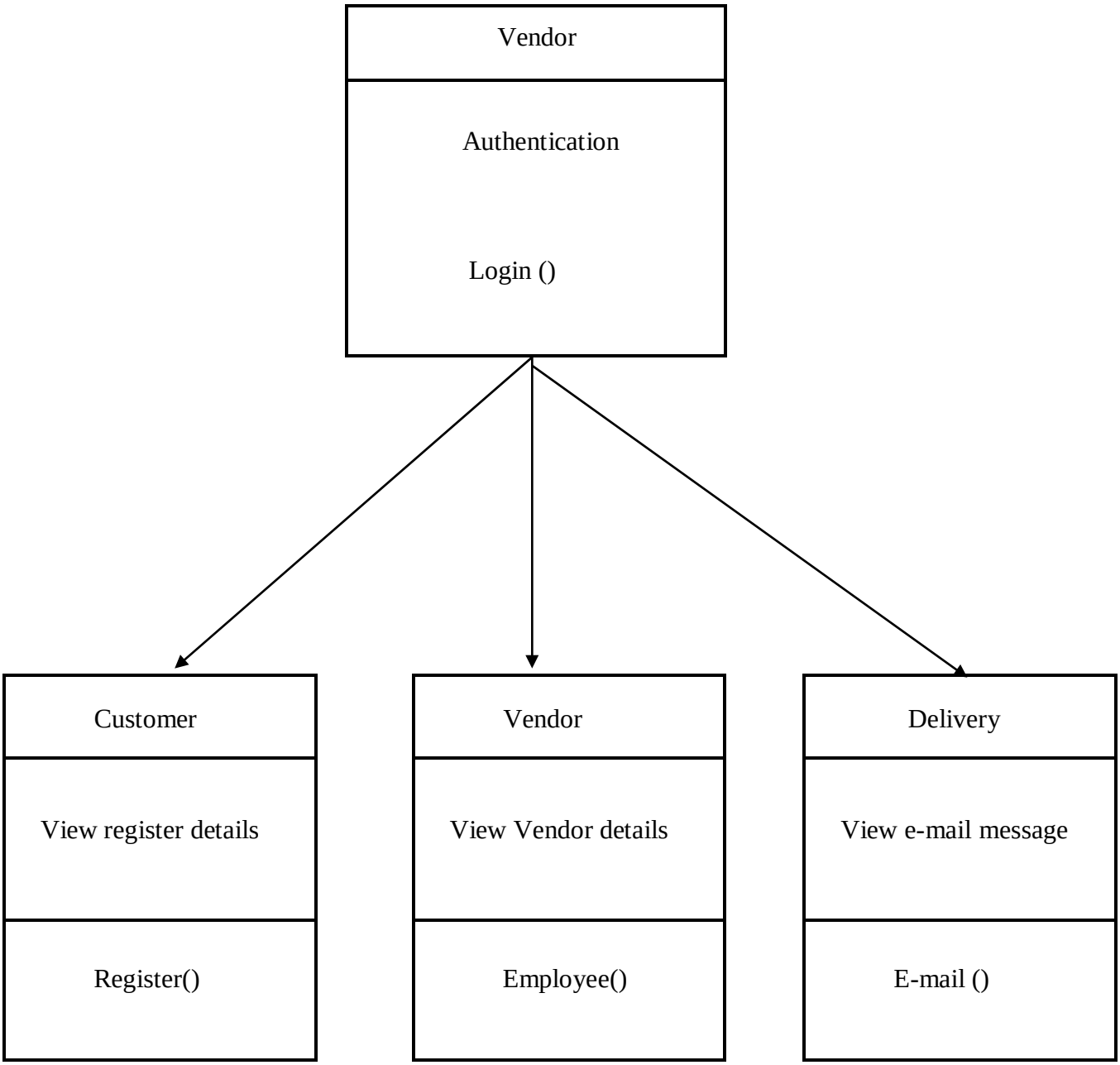

Fig 5.3.2 Admin class diagram

5.4 SEQUENCE DIAGRAM

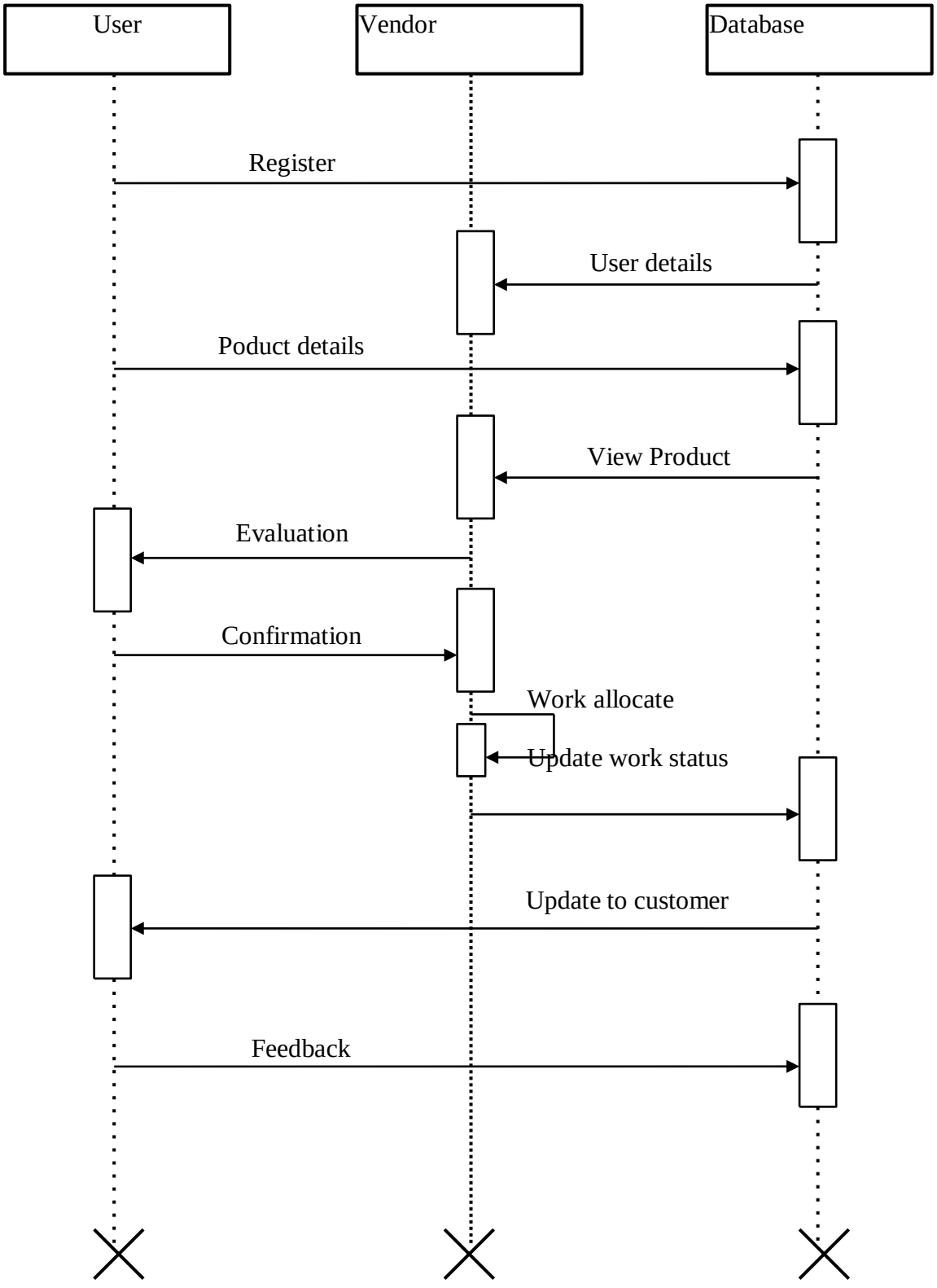

Fig 5.4.1 Sequence diagram

5.5 DATABASE TABLE

FIELD NAME	DATA TYPE
User	InnoDB
Employee	InnoDB
Service	InnoDB
Vendor	InnoDB

Table 5.5.1 Database

FIELD NAME	DATA TYPE
Username	Varchar(30)
Set Password	Varchar(20)
Confirm Password	Varchar(20)
Phone Number	Varchar(10)
Date Of Birth	Varchar(18)
Gender	Varchar(10)
Email Id	Varchar(35)

Table 5.5.2 User/Customer Details

FIELD NAME	DATA TYPE
Employee Id	Int(10)
Employee Name	Varchar(20)
Employee Department	Varchar(20)

Table 5.5.3 Employee details

FIELD NAME	DATA TYPE
Name	Varchar(20)
Phone Number	Varchar(10)
Mobile Name With Model	Varchar(30)
Location	Varchar(50)
Issue	Varchar(23)
Option	Varchar(23)

Table 5.5.4 Customer service details

FIELD NAME	DATA TYPE
Name	Varchar(20)
Password	Varchar(20)

Table 5.5.5 Vendor login details

CHAPTER 6

MODULE DESCRIPTION

A module description provides an overview and explanation of the functionalities and features of a specific module within a larger project. It outlines the purpose, scope, and key components of the module, helping stakeholders and developers understand its role and functionality. These module descriptions provide a general overview of the functionalities each module offers. Depending on the specific requirements of your project, you may need to include additional features or details

MODULES

The important modules in the proposed system and their interaction. It can be seen that the application consists of eight distinct modules namely,

Modules:

- Vendor Module
- Price Submission Module
- User verification Module
- Order list Module
- Order Submission Module

Vendor Module:

In the admin module, the admin can fill out the form of login and move to the next page. This page verifying the administrator data

Price Submission Module:

In the price submission module, the administrator maintains the Price list details. The administrator collects the Price list from the Farmers. The administrator gives the Price details directly to the customers.

User verification Module:

In the User verification module, get the phone number for verify purpose and this application sent OTP for this mobile number.

Order list Module:

In the order list module, customers they will select the required vegetables. The Customers gives the order request to the admin to collect the vegetable list.

CHAPTER 7

SYSTEM TESTING

7.1 UNIT TESTING

In the unit testing the analyst tests the program making up a system. The software units in a system are the modules and routines that are assembled and integrated to perform a specific function. In a large system, many modules on different levels are needed.

Unit testing can be performed from the bottom up starting with the smallest and lowest level modules and proceeding one at a time. For each module in a bottom-up testing, a short program executes the module and provides the needed data.

7.2 INTEGRATION TESTING

Integration testing is a systematic technique for constructing the program structure while conducting test to uncover errors associate with interfacing. Objectives are used to take unit test modules and built program structure that has been directed by design.

The integration testing is performed for this Project when all the modules where to make it a complete system. After integration the project works successfully.

7.3. VALIDATION TESTING

Validation testing can be defined in many ways, but a simple definition is that can be reasonably expected by the customer. After validation test has been conducted, one of two possible conditions exists.

The functions or performance characteristics confirm to specification and are accepted. A deviation from specification is uncovered and a deficiency list is created.

Proposed system under consideration has been tested by using validation testing and found to be working satisfactorily.

For example, in this project validation testing is performed against module. This module is tested with the following valid and invalid inputs for the field id.

7.4. WHITE BOX TESTING

White box testing, sometimes called glass-box testing is a test case design method that uses the control structure of the procedural design to derive test cases. Using white box testing methods, the software engineer can derive test cases that

- Guarantee that all independent paths with in a module have been exercised at least once.
- Exercise all logical decisions on their true and false sides.
- Execute all loops at their boundaries and within their operational bounds and
- Exercise internal data structure to assure their validity.

For example in this project white box testing is performed against patient module. Without entering text if we apply it displays the message “First add record then save it” else it should be saved.

7.5. BLACK BOX TESTING

This method treats the coded module as a black box. The module runs with inputs that are likely to cause errors. Then the output is checked to see if any error occurred. This method cannot be used to test all errors, because some errors may depend on the code or algorithm used to implement the module.

CHAPTER 8
CONCLUSION

The project **"FARMERS MARKET APP"** is a mobile application. The application is developed in **Django** and **Python** which provides a user- friendly application and an enhanced performance. So, it helps the administrator to go through the database anytime from anywhere. In this application farmers can change and update the price of their vegetables daily or weekly. This application helps to increase the economy of the farmers. This application is to reduce the middle man between the farmers and customers. This application helps to the customers also get vegetables at the right price.

CHAPTER 9
FUTURE ENHANCEMENT

In future application to add features to delivery purpose like tomato, swigy. This project will help many unemployed people by working as the delivery man and improve their life status. If there is any problem in the product the customers can directly compact to the farmers directly through this application.

CHAPTER 10

APPENDICES

10.1 SCREEN SHOTS

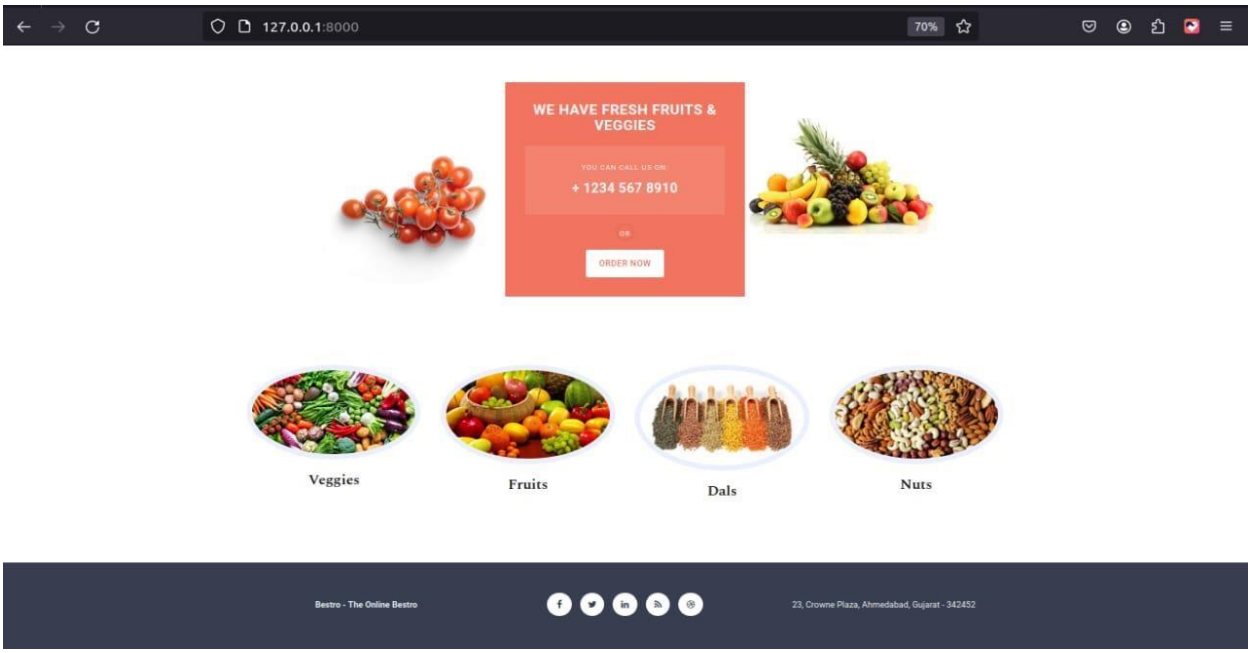

Fig 10.1.1 Home Page

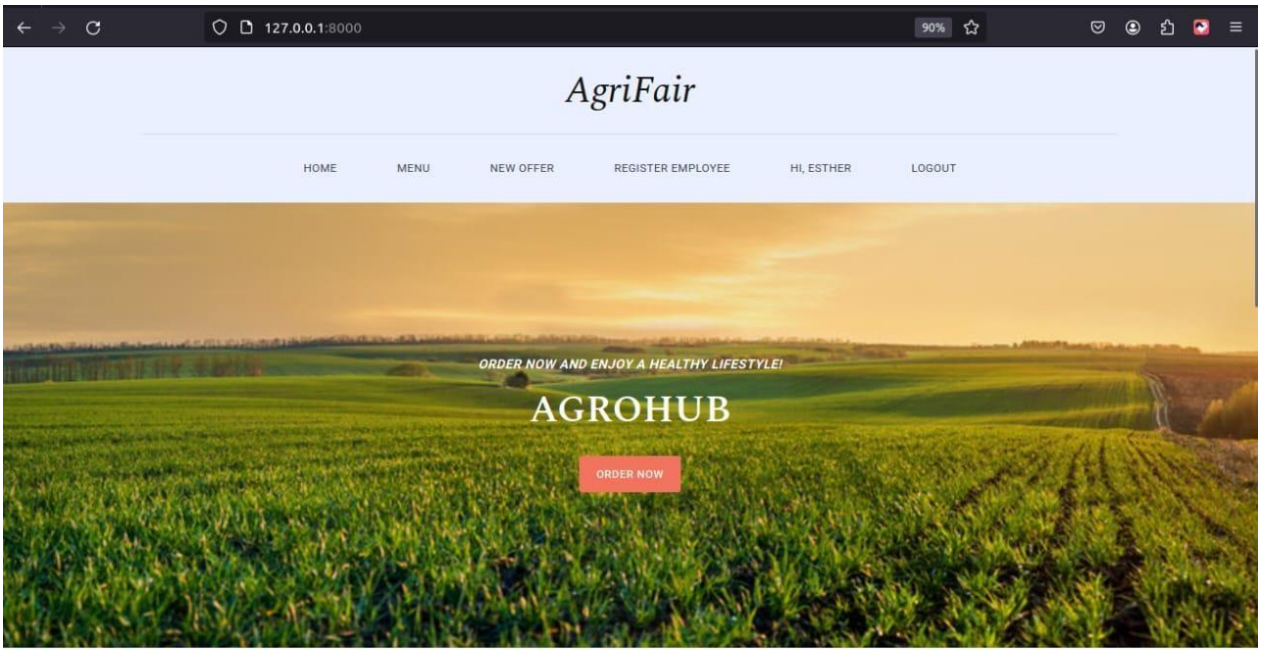

Fig 10.1.2 Home Page

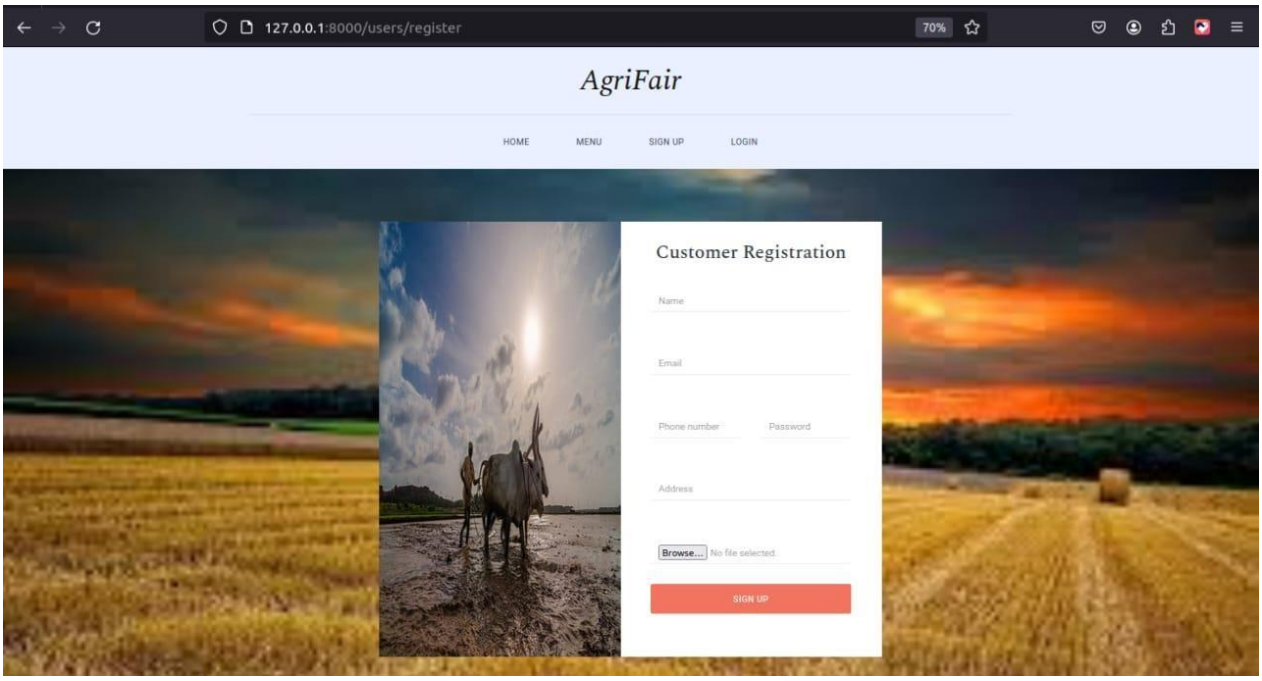

Fig 10.1.3 Register page

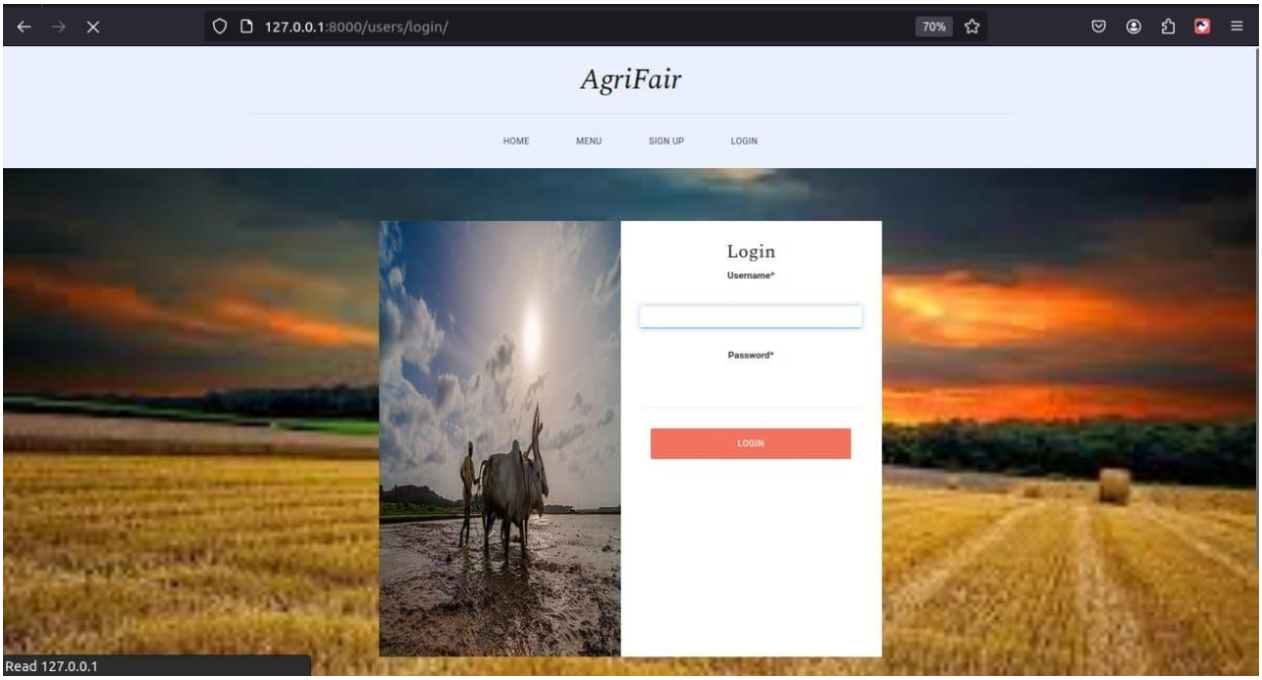

Fig 10.1.4 Login page

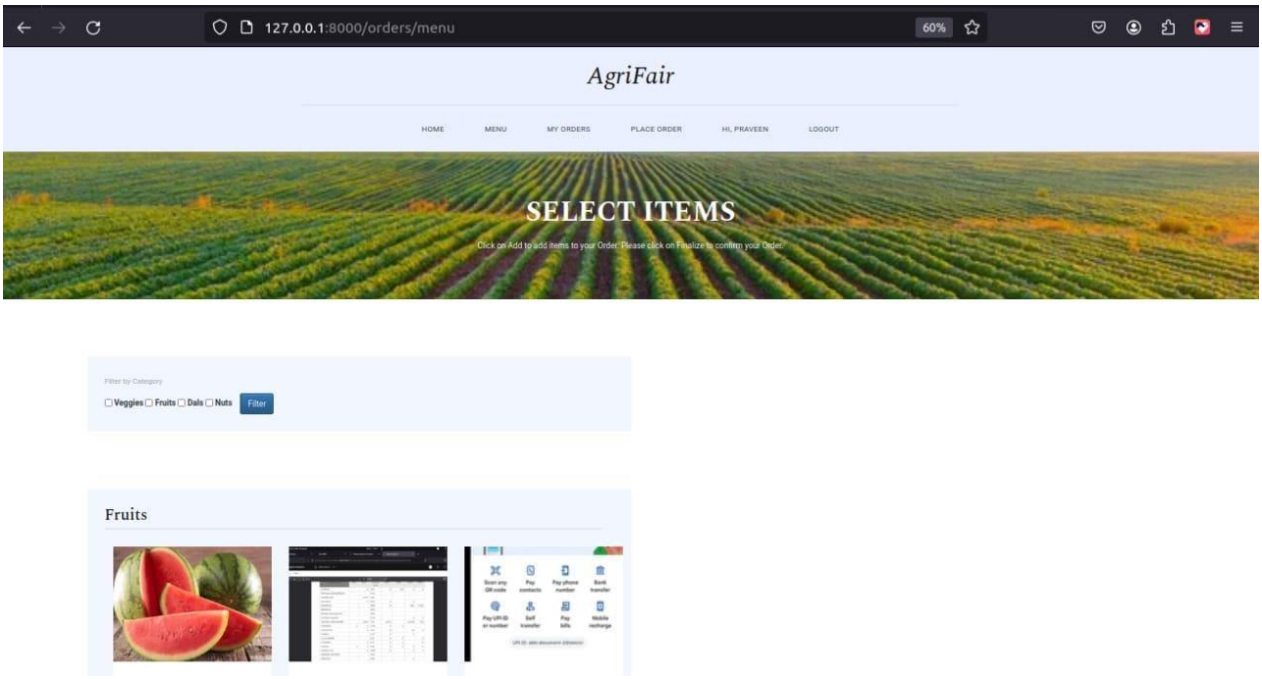

Fig 10.1.5 Product menu page

s

Figure 10.1.6 Order page

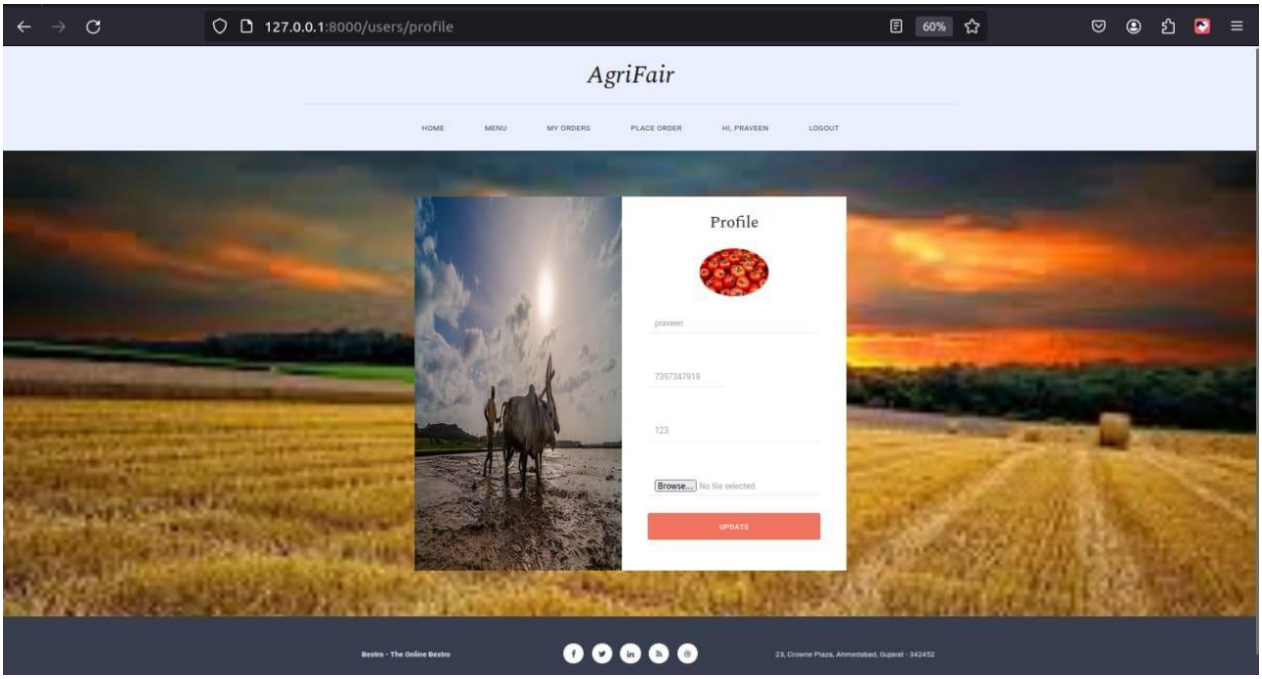

Fig 10.1.7 User profile page

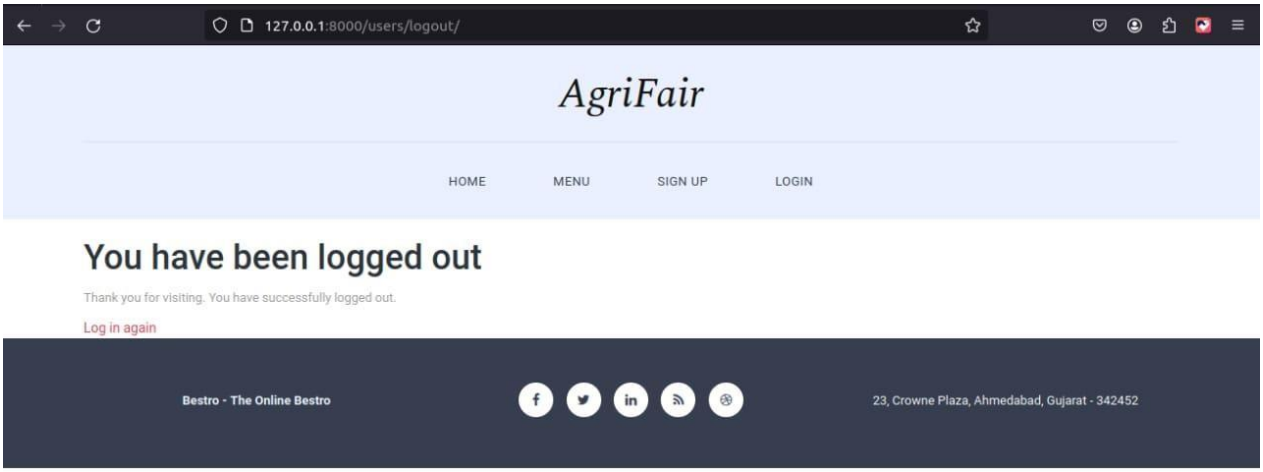

Fig 10.1.8 logged out page

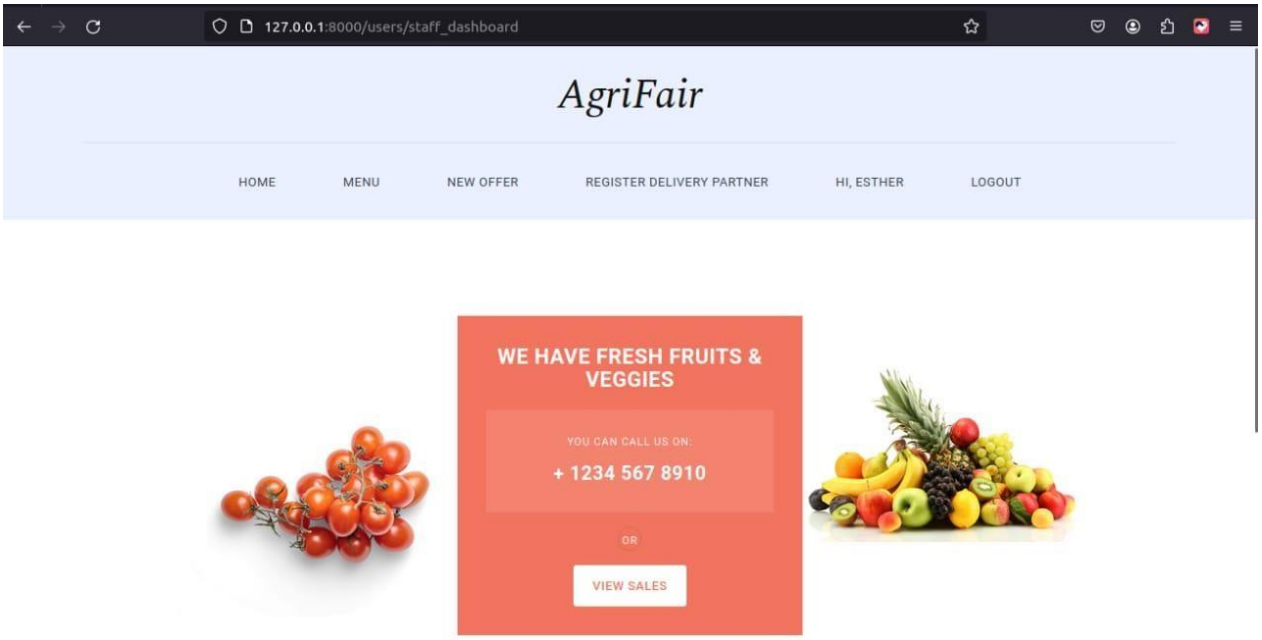

Fig 10.1.9 Vendor dashboard page

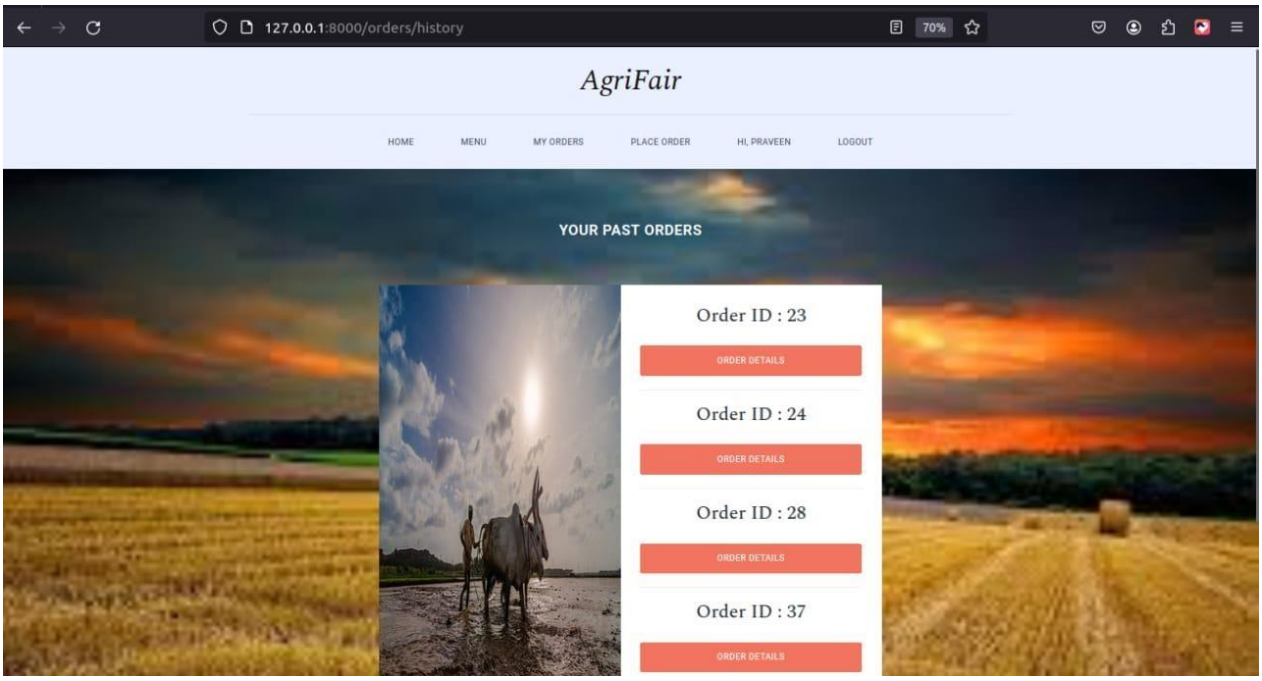

Fig 10.1.10 Order history page

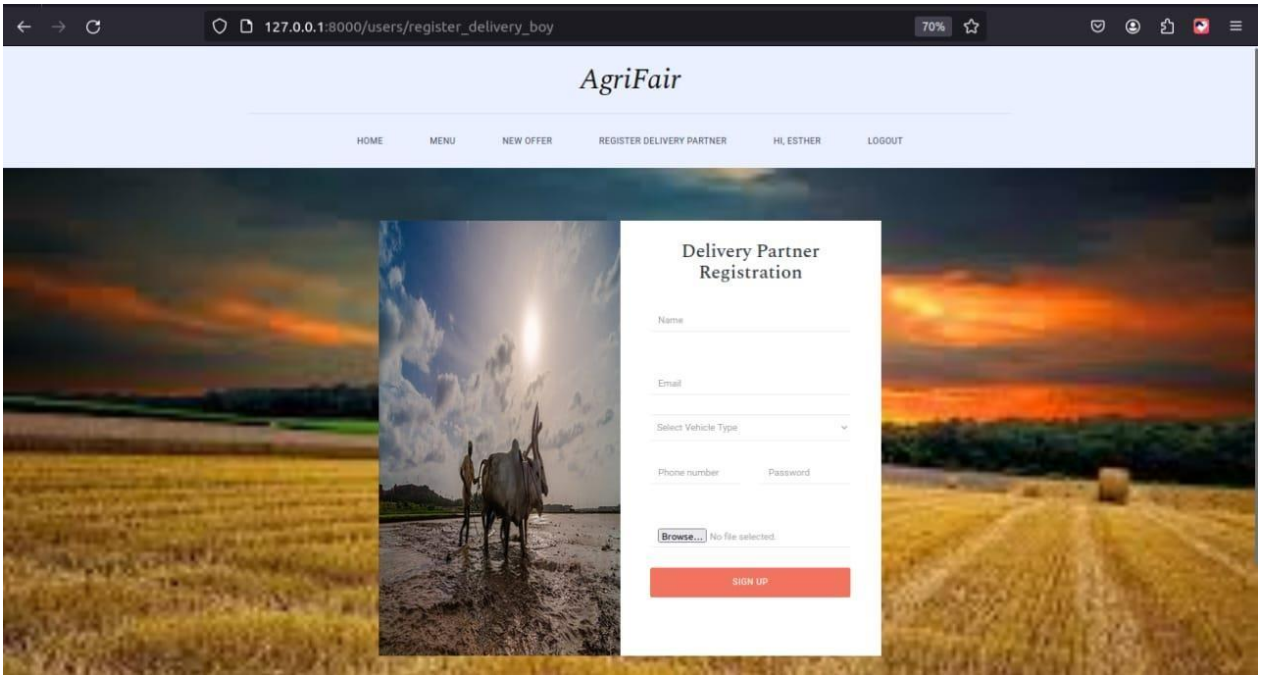

Fig 10.1.11 Delivery pattern add page

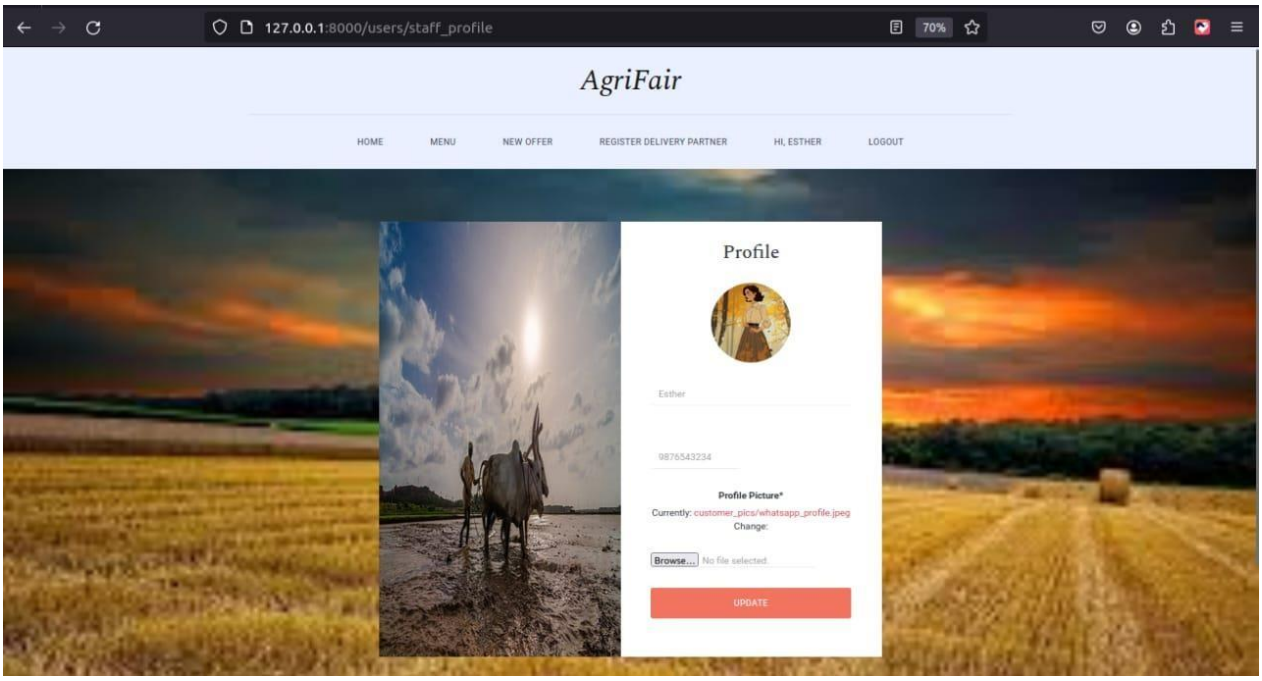

Fig 10.1.12 Vendor profile page

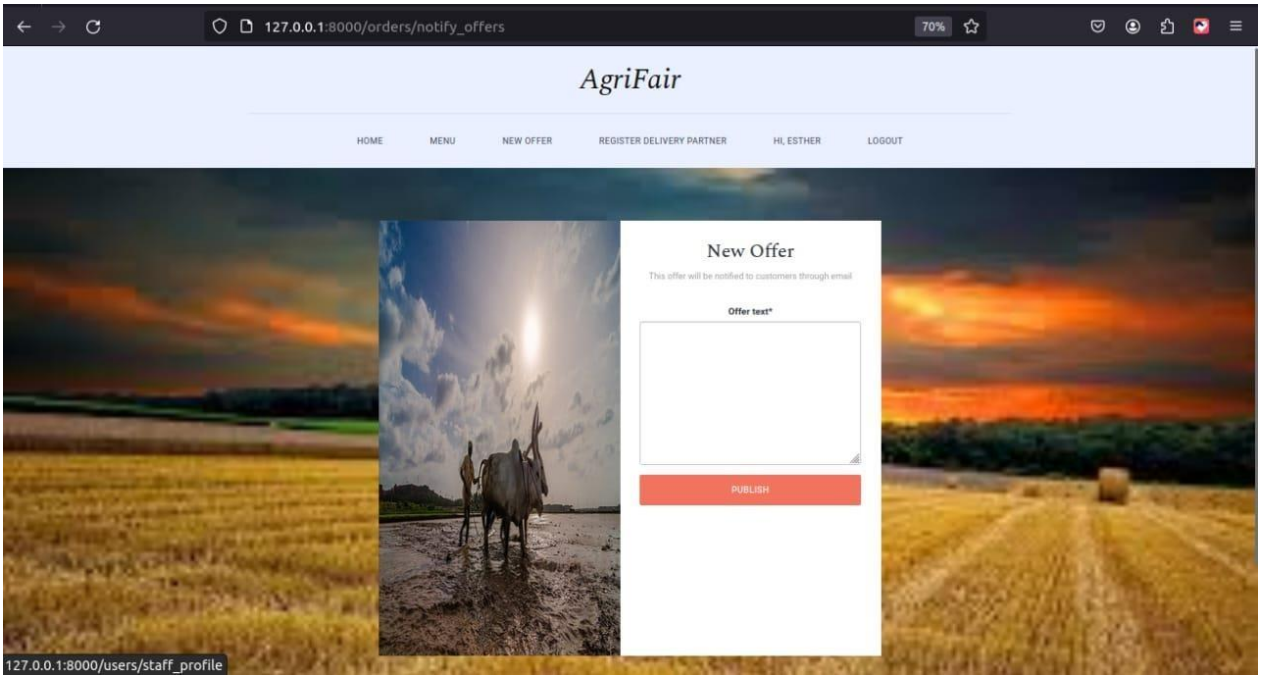

Fig 10.1.13 Offer published page

10.2 CODINGS

DJANGO CODING

#items_App

#admin_py

```
from django.contrib import admin
from items.models import Item, Combo
admin.site.register([Item, Combo])
# Register your models here.
```

#forms.py

```
from django import forms
from items.models import Item, Combo
class ItemForm(forms.ModelForm):
    class Meta:
        model = Item
        fields = ['name', 'description', 'category', 'rate', 'stock', 'is_non_veg', 'image']
class ComboForm(forms.ModelForm):
    class Meta:
        model = Combo
        fields = ['name', 'description', 'rate', 'item1', 'item2', 'item3', 'item4', 'item5', 'image']
```

#models.py

```
from django.db import models
from PIL import Image
from items.constants import CATEGORIES
class Item(models.Model):
    name = models.CharField(max_length=50)
    category = models.CharField(max_length=50, choices=CATEGORIES)
```

```
    description = models.TextField(max_length=500)
    rate = models.DecimalField(max_digits=5, decimal_places=2)
    is_non_veg = models.BooleanField(default=False)
    stock = models.IntegerField(default=10)
    rating = models.DecimalField(max_digits=2, decimal_places=1, default=5)
    image = models.ImageField(verbose_name="Feature Image", upload_to="item_pics",
default="media/default_item.png")
    #non_availablity_time = models.DateTimeField(null=True, blank=True)
    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        img = Image.open(self.image.path)
        if img.height>300 or img.width > 300:
            output_size = (300, 300)
            img.thumbnail(output_size)
            img.save(self.image.path)
class Combo(models.Model):
    name = models.CharField(max_length=50, default = "")
    description = models.TextField(max_length=500, default="")
    item1 = models.ForeignKey(Item, on_delete=models.CASCADE, related_name="item1")
    item2 = models.ForeignKey(Item, on_delete=models.CASCADE, related_name="item2")
    item3 = models.ForeignKey(Item, on_delete=models.CASCADE, related_name="item3")
    item4 = models.ForeignKey(Item, on_delete=models.CASCADE, related_name="item4")
    item5 = models.ForeignKey(Item, on_delete=models.CASCADE, related_name="item5")
    rate = models.DecimalField(max_digits=5, decimal_places=2)
    rating = models.DecimalField(max_digits=2, decimal_places=1, default=5)
    image = models.ImageField(verbose_name="Feature Image", upload_to="item_pics",
default="media/default_item.png")
```

```
non_availablity_time = models.DateTimeField(auto_now=False, null=True)
def save(self, *args, **kwargs):
    calculate_rate(self)
    super().save(*args, **kwargs)
    img = Image.open(self.image.path)
    if img.height>400 or img.width > 400:
        output_size = (400, 400)
        img.thumbnail(output_size)
        img.save(self.image.path)
def calculate_rate(self):
    self.rate = 0
    if self.item1:
        self.rate += item1.rate
    if self.item2:
        self.rate += item2.rate
    if self.item3:
        self.rate += item3.rate
    if self.item4:
        self.rate += item4.rate
    if self.item5:
        self.rate += item5.rate
@property
def is_available(self):
    if item1 is not None and item1.stock == 0:
        return False
    if item2 is not None and item2.stock == 0:
```

```
            return False
        if item3 is not None and item3.stock == 0:
            return False
        if item4 is not None and item4.stock == 0:
            return False
        if item5 is not None and item5.stock == 0:
            return False
        return True
# Create your models here.
```

#Urls.py

```
from django.urls import path
from items.views import *
from django.contrib.auth import views as auth_views
urlpatterns = [
    path('create_item', create_item, name='create_item'),

    path('<int:pk>/update_item', update_item, name='update_item'),
    #path('create_combo', create_combo, name='create_combo'),
    #path('<int:pk>/update_combo', update_combo, name='update_combo'),
]
```

#Views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.admin.views.decorators import staff_member_required
from django.contrib import messages
from items.models import Item, Combo
from items.forms import ItemForm, ComboForm
```

```
@staff_member_required
def create_item(request):
    if request.method == "POST":
        form = ItemForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            messages.success(request, f'Item Added to Menu!')
            return redirect('menu')
    else:
        form = ItemForm()
    context = {
        'form' : form
    }
    return render(request, 'items/create_item.htm', context)
@staff_member_required
def update_item(request, pk):
    item = get_object_or_404(Item, pk = pk)
    if request.method == "POST":
        form = ItemForm(request.POST, instance = item)
        if form.is_valid():
            form.save()
            messages.success(request, f'Item Updated!')
            return redirect('menu')
    else:
        form = ItemForm(instance = item)
    context = {
```

```
        'form' : form,
        'item' : item
    }
    return render(request, 'items/update_item.htm', context)
@staff_member_required
def create_combo(request):
    if request.method == "POST":
        form = ComboForm(request.POST)
        if form.is_valid():


            form.save()
            return redirect('create_combo')
    else:
        form = ComboForm()
    context = {
        'form' : form
    }
    return render(request, 'combos/create_combo.htm', context)
@staff_member_required
def update_combo(request, pk):
    combo = get_object_or_404(Combo, pk = pk)
    if request.method == "POST":
        form = ComboForm(request.POST, instance = combo)
        if form.is_valid():
            form.save()
            return redirect('create_combo')
```

```
    else:
        form = ComboForm(instance = combo)
    context = {
        'form' : form,
        'combo' : combo
    }
    return render(request, 'items/create_combo.htm', context)
# Create your views here.
```

#Orders_App

#admin.py

```
from django.contrib import admin
from orders.models import Order, OrderedItem
admin.site.register([Order, OrderedItem])
# Register your models here.
```

#forms.py

```
from django import forms
class OfferForm(forms.Form):
    offer_text = forms.CharField(widget=forms.Textarea, max_length=2000, required=True)
```

#model.py

```
from django.db import models
from django_mysql.models import Model
from django.utils import timezone
from datetime import datetime, date
from users.models import Customer
from items.models import Item, Combo
class Order(models.Model):
```

```
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    date_placed = models.DateTimeField(default=date.today())
    time_placed = models.TimeField(default=datetime.now())
    grand_total = models.DecimalField(max_digits=6, decimal_places=2, default=0)
    expected_time = models.IntegerField(default=15) # Expected Waiting time in minutes
    finalized = models.BooleanField(default=False)
    delivered = models.BooleanField(default=False)
    #json = JSONField(encoder="")
class OrderedItem(models.Model):
    item = models.ForeignKey(Item, on_delete=models.CASCADE, related_name="item")
    order = models.ForeignKey(Order, on_delete=models.CASCADE)
    quantity = models.IntegerField(default=0)
    price = models.DecimalField(max_digits=5, decimal_places=2, default=0)
    rating = models.DecimalField(max_digits=2, decimal_places=1, default=5)
    def calculate_price(self):
        self.price = self.item.rate * self.quantity
        return self.price
# Create your models here.
```

#urls.py

```
from django.urls import path
from orders.views import *
from django.contrib.auth import views as auth_views
urlpatterns = [
    path('new_order', create_order, name='create_order'),
    path('<int:pk>/menu', add_items, name='add_items'),
    path('<int:pk>/confirm', finalize_order, name='finalize_order')
```

```
    path('<int:pk>/delete', delete_order, name='delete_order'),
    path('<int:pk>/summary', order_summary, name='order_summary'),
    path('<int:pk>/confirm_delivery', accept_order, name='accept_order'),
    path('<int:pk>/feedback', feedback, name='feedback'),
    path('history', past_transactions, name='past_transactions'),
    path('generate_sales', generate_sales, name='generate_sales'),
    path('notify_offers', notify_offers, name='notify_offers'),
    path('menu', test_menu, name='menu'),
]
```

#views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
from django.contrib.admin.views.decorators import staff_member_required
from django.http import HttpResponse
import xlwt
from datetime import date, datetime
from django.contrib import messages
from users.models import Customer, Staff
from items.models import Item, Combo
from orders.models import Order, OrderedItem
from items.constants import CATEGORIES
from orders.forms import OfferForm
from orders.utils import *
def index(request):
    return render(request, 'orders/index.htm')
def test_menu(request):
```

```
    vg_items = []
    fr_items = []
    dl_items = []
    nt_items = []
    context = {
        'vg_items' : [],
        'fr_items' : [],
        'dl_items' : [],
        'nt_items' : []
    }
    for item in Item.objects.filter(stock_gte=1):
        category = (item.category).lower()
        context[category+"_items"].append(item)
    print(context, "><---------- ")
    return render(request, 'orders/menu-1.htm', context)
@login_required
def create_order(request):
    customer = get_object_or_404(Customer, user = request.user)
    order = Order.objects.create(
        customer = customer,
    )
    context = {
        'order' : order,
        'customer' : customer
    }
    return redirect('add_items', order.id)
```

```
@login_required
def add_items(request, pk):
    order = get_object_or_404(Order, pk = pk)
    ordered_items = list(OrderedItem.objects.filter(order = order))
    if request.method == "POST":
        data = dict(request.POST)
        items = Item.objects.all()
        for item in items:
            item_quantity = data[str(item.id)][0]
            if item_quantity is '':
                item_quantity = 0
            if int(item_quantity) >= 1:
                ordered_item = OrderedItem.objects.filter(
                    order = order,
                    item = item
                )
                if len(ordered_item)==0:
                    ordered_item = OrderedItem.objects.create(
                        order = order,
                        item = item
                    )
                    ordered_item.quantity += int(item_quantity)
                    ordered_item.save()
                    ordered_items.append(ordered_item)
                else:
                    ordered_item = ordered_item[0]
```

```
                ordered_item.quantity += int(item_quantity)
                ordered_item.save()
        # Calculate Price of Order
        order.save()
        messages.success(request, f'Items Added to Order!')
        return redirect('add_items', order.id)
    vg_items = []
    fr_items = []
    dl_items = []
    nt_items = []
    context = {
        'vg_items' : [],
        'fr_items' : [],
        'dl_items' : [],
        'nt_items' : [],
        'order' : order,
        'ordered_items' : ordered_items,
    }
    for item in Item.objects.filter(stock_gte=1):
        category = (item.category).lower()
        context[category+"_items"].append(item)
    print(context, "< ---------- ")
    return render(request, 'orders/menu.htm', context)
@login_required
def finalize_order(request, pk):
    order = get_object_or_404(Order, pk = pk)
```

```
    ordered_items = OrderedItem.objects.filter(order = order)
    print(ordered_items)
    if len(ordered_items):
        order.date_placed = date.today()
        order.time_placed = datetime.now()
        calculate_grand_total_and_update_stocks(order, ordered_items)
        calculate_expected_delivery_time(order)
        order.finalized = True
        order.save()
        messages.success(request, f'Order Placed!')
        return redirect('order_summary', order.id)
    else:
        return redirect('add_items', order.id)
@login_required
def order_summary(request, pk):
    order = get_object_or_404(Order, pk = pk)
    ordered_items = OrderedItem.objects.filter(order = order)
    context = {
        'order' : order,
        'items' : ordered_items
    }
    return render(request, 'orders/order_summary.htm', context)
@login_required
def accept_order(request, pk):
    order = get_object_or_404(Order, pk = pk)
    order.delivered = True
```

```
    order.save()
    return redirect('feedback', order.pk)
@login_required
def delete_order(request, pk):
    try:
        Order.objects.filter(id=pk).delete()
        messages.success(request, f'Order Cancelled')
        return redirect('index')
    except:
        return redirect('index')
@login_required
def feedback(request, pk):
    order = get_object_or_404(Order, pk = pk)
    ordered_items = OrderedItem.objects.filter(order = order)
    if request.method == "POST":
        data = dict(request.POST)
        print(data)
        for item in ordered_items:
            print(data.get(str(item.id))[0])
            item.rating = int(data.get(str(item.id))[0])
            item.save()
        calculate_item_ratings(ordered_items)
        messages.success(request, f'Thanks for your valuable feedback!')
        return redirect('customer_dashboard')
    context = {
        'order' : order,
```

```
        'items' : ordered_items
    }

    return render(request, 'orders/feedback.htm', context)
@login_required
def past_transactions(request):
    customer = get_object_or_404(Customer, user = request.user)
    orders = Order.objects.filter(customer = customer, finalized = True)
    context = {
        'customer' : customer,
        'orders' : orders
    }
    return render(request, 'orders/past_transactions.htm', context)
# Admin Controls
@login_required
@staff_member_required
def generate_sales(request):
    today = date.today().strftime("%d%m%Y")
    response = HttpResponse(content_type='application/ms-excel')
    response['Content-Disposition'] = "attachment; filename=Sales_"+str(today)+".xlsx"
    wb = xlwt.Workbook(encoding='utf-8')
    ws = wb.add_sheet("Sales_"+str(today))
    #Writing the headers
    row = 0
    font_style = xlwt.XFStyle()
    font_style.font.bold = True
```

```
columns = [
    'Category', 'Item ID', 'Item Name', 'Orders', 'Sales',
]
for col in range(len(columns)):
    ws.write(row, col, columns[col], font_style)
#Writing Orders data to the sheet
font_style = xlwt.XFStyle()
orders = Order.objects.filter(date_placed=date.today(), finalized = True)
items = Item.objects.all()
ordered_items = []
for order in orders:
    ordered_items += OrderedItem.objects.filter(order = order)
net_sales = 0
for c in range(len(CATEGORIES)):
    category = str(CATEGORIES[c][0])
    for i in items:
        if i.category == category:
            count = 0
            sales = 0
            for o_item in ordered_items:
                if o_item.item == i:
                    count += o_item.quantity
                    sales += o_item.price
            net_sales += sales
            row += 1
            ws.write(row, 0, i.category, font_style)
```

```
            ws.write(row, 1, i.id, font_style)
            ws.write(row, 2, i.name, font_style)
            ws.write(row, 3, count, font_style)
            ws.write(row, 4, sales, font_style)
    row += 1
    ws.write(row, 3, "TOTAL SALES", font_style)
    ws.write(row, 4, net_sales, font_style)
    wb.save(response)
    return response
@login_required
@staff_member_required
def notify_offers(request):
    customers = Customer.objects.all()
    if request.method=="POST":
        form = OfferForm(request.POST)
        if form.is_valid():
            offer_text = form.cleaned_data['offer_text']
            mail_customers(customers, offer_text)
            messages.success(request, f'Order Published!')
            return redirect('staff_dashboard')
    else:
        form = OfferForm()
    context = {
        'form' : form
    }
    return render(request, 'orders/notify_offers.htm', context)
```

```
# Create your views here.
```

#User_App

#admin.py

```
from django.contrib import admin
from users.models import Customer, Staff
admin.site.register([Customer, Staff])
# Register your models here.
```

#forms.py

```
from django import forms
from users.models import Customer, Staff
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
class UserRegistrationForm(UserCreationForm):
    class Meta(UserCreationForm):
        model = User
        fields = ['username', 'email', 'password1', 'password2']


class CustomerProfileForm(forms.ModelForm):
    class Meta:
        model = Customer
        fields = ['name', 'address', 'phone', 'image']
class StaffProfileForm(forms.ModelForm):
    class Meta:
        model = Staff
        fields = ['name', 'phone', 'image']
```

#models.py

```
from django.db import models
from django.contrib.auth.models import User
from users.constants import DESIGNATIONS
from PIL import Image
class Customer(models.Model):
    """
    Details of each customer registered on the website are stored as
    an instance of the Customer class. The relevant fields are enumerated below.
    """
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=100, default="", blank=False, null=False)
    address = models.TextField(max_length=1000, default="", blank=False, null=False)
    phone = models.CharField(verbose_name = "Mobile", max_length=14)
    email = models.EmailField(max_length=254)
    image = models.ImageField(verbose_name="Profile Picture", upload_to="customer_pics",
default="media/default_user.png")
    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        img = Image.open(self.image.path)
        if img.height>400 or img.width > 400:
            output_size = (400, 400)
            img.thumbnail(output_size)
            img.save(self.image.path)
class Staff(models.Model):
    """
    Details of the each staff of the website are stored as
```

```
    an instance of the Staff class. The relevant fields are enumerated below.
    """
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    emp_id = models.CharField(verbose_name="Employee ID", max_length=50)
    name = models.CharField(max_length=100, default="", blank=False, null=False)
    #designation = models.CharField(choices=DESIGNATIONS, max_length=50)
    phone = models.CharField(verbose_name = "Mobile", max_length=14)
    email = models.EmailField(max_length=254)

    image = models.ImageField(verbose_name="Profile Picture", upload_to="customer_pics",
default="media/default_user.png")

    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        img = Image.open(self.image.path)
        if img.height>400 or img.width > 400:
            output_size = (400, 400)
            img.thumbnail(output_size)
            img.save(self.image.path)
# Create your models here.
```

#urls.py

```
from django.urls import path
from users.views import *
from django.contrib.auth import views as auth_views
urlpatterns = [
    path('register', register_customer, name='register_customer'),
```

```
    path('register_staff', register_staff, name='register_staff'),
    path('dashboard', customer_dashboard, name='customer_dashboard'),
    path('staff_dashboard', staff_dashboard, name='staff_dashboard'),
    path('profile', customer_profile, name='customer_profile'),
    path('staff_profile', staff_profile, name='staff_profile'),
    path('redirecting', post_login, name='post_login'),
    path('login/', auth_views.LoginView.as_view(template_name='users/login.htm'),
name='login'),


    # path('logout/', auth_views.LogoutView.as_view(template_name='users/logout.htm'),
name='logoutt'),
    path('logout/', logout_view, name='logout'),
]
```

#views.py

```
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib.auth import login, authenticate
from allauth.socialaccount.models import SocialAccount
from django.contrib.auth.decorators import login_required
from django.contrib.admin.views.decorators import staff_member_required
from django.contrib import messages
from django.contrib.auth import logout
from users.models import Customer, Staff
from users.forms import UserRegistrationForm, CustomerProfileForm, StaffProfileForm
def post_login(request):
    if request.user.is_superuser:
        return redirect('staff_dashboard')
    else:
```

```
        return redirect('customer_dashboard')
def register_customer(request):
    if request.method == "POST":
@staff_member_required
def create_item(request):
    if request.method == "POST":
        form = ItemForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            messages.success(request, f'Item Added to Menu!')
            return redirect('menu')
    else:
        form = ItemForm()
    context = {
        'form' : form
    }
    return render(request, 'items/create_item.htm', context)
@staff_member_required
def update_item(request, pk):
    item = get_object_or_404(Item, pk = pk)
    if request.method == "POST":
        form = ItemForm(request.POST, instance = item)
        if form.is_valid():
            form.save()
            messages.success(request, f'Item Updated!')
            return redirect('menu')
```

```
    else:
        form = ItemForm(instance = item)
    context = {
        'form' : form,
        'item' : item
    }
    return render(request, 'items/update_item.htm', context)
@staff_member_required
def create_combo(request):
    if request.method == "POST":
        form = ComboForm(request.POST)
        if form.is_valid():

            form.save()
            return redirect('create_combo')
    else:
        form = ComboForm()
    context = {
        'form' : form
    }
    return render(request, 'combos/create_combo.htm', context)
@staff_member_required
def update_combo(request, pk):
    combo = get_object_or_404(Combo, pk = pk)
    if request.method == "POST":
        form = ComboForm(request.POST, instance = combo)
```

```
        if form.is_valid():
            form.save()
            return redirect('create_combo')
    else:
        form = ComboForm(instance = combo)
    context = {
        'form' : form,
        'combo' : combo
    }
    return render(request, 'items/create_combo.htm', context)
# Create your views here.
```

#Orders_App

#admin.py

```
from django.contrib import admin
from orders.models import Order, OrderedItem
admin.site.register([Order, OrderedItem])
# Register your models here.
```

#forms.py

```
from django import forms
class OfferForm(forms.Form):
    offer_text = forms.CharField(widget=forms.Textarea, max_length=2000, required=True)
```

#model.py

```
from django.db import models
from django_mysql.models import Model
from django.utils import timezone
from datetime import datetime, date
```

```
from users.models import Customer
from items.models import Item, Combo
class Order(models.Model):
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    date_placed = models.DateTimeField(default=date.today())
    time_placed = models.TimeField(default=datetime.now())
    grand_total = models.DecimalField(max_digits=6, decimal_places=2, default=0)
    expected_time = models.IntegerField(default=15) # Expected Waiting time in minutes
    finalized = models.BooleanField(default=False)
    delivered = models.BooleanField(default=False)
    #json = JSONField(encoder="")


        user_form = UserRegistrationForm(request.POST)
        customer_form = CustomerProfileForm(request.POST, request.FILES)


        if user_form.is_valid() and customer_form.is_valid():
            user_form.save()
            username = user_form.cleaned_data["username"]
            pwd = user_form.cleaned_data["password1"]
            auth_user = authenticate(username=username, password=pwd)
            login(request, auth_user)
            customer = customer_form.instance
            customer.user = auth_user
            customer.email = auth_user.email
            customer.save()
            messages.success(request, f'Welcome to Bistro! Happy Fooding!')
```

```
            return redirect('customer_dashboard')
    else:
        user_form = UserRegistrationForm()
        customer_form = CustomerProfileForm()
    context = {
        'uform' : user_form,
        'cform' : customer_form
    }
    return render(request, 'users/register_customer.htm', context)
@login_required
def customer_dashboard(request):
    user = request.user
    if not user.is_superuser:
        if not Customer.objects.filter(user = user).exists():
            customer = Customer.objects.create(
                user = user,
            )
            customer.save()
            messages.success(request, f'Please Update Your Profile to Help us serve You Better!')
            return redirect('customer_profile')
        else:
            customer = get_object_or_404(Customer, user = user)
        if customer.name == "" or customer.address == "" or customer.phone == "":
            messages.success(request, f'Please Update Your Profile to Help us serve You Better!')
            return redirect('customer_profile')
        context = {
```

```
            'customer' : customer
        }
        return render(request, 'users/customer_dashboard.htm', context)
@login_required
def customer_profile(request):
    customer = get_object_or_404(Customer, user = request.user)
    if request.method == "POST":
        form = CustomerProfileForm(request.POST, request.FILES, instance = customer)
        if form.is_valid():
            form.save()
            customer.email = request.user.email
            customer.save()
            messages.success(request, f'Profile Updated!')
            return redirect('customer_dashboard')
    else:
        form = CustomerProfileForm(instance = customer)
    context = {
        'customer' : customer,
        'form' : form
    }
    return render(request, 'users/customer_profile.htm', context)
# Staff Methods
@staff_member_required
def register_staff(request):
    if request.method == "POST":
        user_form = UserRegistrationForm(request.POST)
```

```
        staff_form = StaffProfileForm(request.POST, request.FILES)
        if user_form.is_valid() and staff_form.is_valid():
            user_form.save()
            user = user_form.instance
            user.is_superuser = True
            user.is_staff = True
            user.save()
            #auth_user = authenticate(username=username, password=pwd)
            #login(request, auth_user)
            staff = staff_form.instance
            staff.user = user
            staff.email = user.email
            staff.save()
            return redirect('staff_dashboard')
    else:
        user_form = UserRegistrationForm()
        staff_form = StaffProfileForm()
    context = {
        'uform' : user_form,
        'cform' : staff_form
    }
    return render(request, 'users/register_staff.htm', context)
@staff_member_required
def staff_dashboard(request):
    user = request.user
    if user.is_superuser:
```

```
        if not Staff.objects.filter(user = user).exists():
            staff = Staff.objects.create(
                user = user
            )
            staff.save()
            return redirect('staff_profile')
        else:
            staff = get_object_or_404(Staff, user = user)
        context = {
            'staff' : staff
        }
        return render(request, 'users/staff_dashboard.htm', context)
@staff_member_required
def staff_profile(request):
    staff = get_object_or_404(Staff, user = request.user)
    if request.method == "POST":
        form = StaffProfileForm(request.POST, request.FILES, instance = staff)
        if form.is_valid():
            form.save()
            staff.email = request.user.email
            staff.save()
            return redirect('staff_dashboard')
    else:
        form = StaffProfileForm(instance = staff)
    context = {
        'staff' : staff,
```

```
        'form' : form
    }
    return render(request, 'users/staff_profile.htm', context)
# Create your views here.
def logout_view(request):
    logout(request)
    return render(request, 'users/logout.htm')
    # return redirect('login')
```

HTML & CSS CODING

#Items

#Create Items

```
{% extends "orders/base.htm" %}
{% load crispy_forms_tags %}
{% load static %}
{% block content %}
    <!--- ====  AUTOFILL SCRIPTS ==== -->
    <script>
        function fillUsername() {
            document.getElementById('username').value =
document.getElementById('email').value;
        }
        function fillPassword() {
            document.getElementById('password').value =
document.getElementById('password1').value;
        }
    </script>
    <section id="book-table">
```

```
<div class="container">
    <div class="row">
        <div class="col-md-4 col-md-offset-2 col-sm-12">
            <div class="left-image">
                <img src="{% static 'orders/img/book_left_image.jpg' %}"" alt="">
            </div>
        </div>
        <div class="col-md-4 col-sm-12">
            <div class="right-info">
                <h4>Create New Item</h4>
                <form id="form-submit" action="" method="POST" enctype="multipart/form-data">
                    {% csrf_token %}
                    <div class="col-md-12">
                        <fieldset>
                            <input name="name" type="text" class="form-control" id="name" placeholder="Item Name" required oninput="fillUsername()">
                        </fieldset>
                    </div>

                    <div class="col-md-12">
                        <fieldset>
                            <input name="description" type="text" class="form-control" id="description" placeholder="Description" required hidden>
                        </fieldset>
                    </div>
                    <div class="col-md-12">
```

```
                {{ form.category | as_crispy_field }}
            </div>

            <div class="col-md-4">
                <fieldset>
                    <input name="rate" type="number" class="form-control" id="rate"
placeholder="Rate" required>
                </fieldset>
            </div>
            <div class="col-md-4">
                <fieldset>
                    <input name="stock" type="number" class="form-control" id="stock"
placeholder="Stock" required>
                </fieldset>
            </div>
            <div class="col-md-4">
                {{ form.is_non_veg | as_crispy_field }}
            </div>

            <div class="col-md-12">
                {{ form.image | as_crispy_field }}
            </div>
            <div class="col-md-12">
                <fieldset>
                    <button type="submit" id="form-submit" class="btn">SAVE</button>
                </fieldset>
            </div>
```

```
                    </div>
                </form>
            </div>
        </div>
    </div>
</section>
{% endblock content %}
```

#Update Items

```
{% extends "orders/base.htm" %}
{% load crispy_forms_tags %}
{% load static %}
{% block content %}


<!--- ====  AUTOFILL SCRIPTS ==== -->
<script>
    function fillUsername() {
        document.getElementById('username').value =
document.getElementById('email').value;
    }
    function fillPassword() {
        document.getElementById('password').value =
document.getElementById('password1').value;
    }
</script>
<section id="book-table">
    <div class="container">
```

```
<div class="row">
    <div class="col-md-4 col-md-offset-2 col-sm-12">
        <div class="left-image">
            <img src="{% static 'orders/img/book_left_image.jpg' %}"" alt="">
        </div>
    </div>
    <div class="col-md-4 col-sm-12">
        <div class="right-info">
            <h4>Create New Item</h4>
            <form id="form-submit" action="" method="POST" enctype="multipart/form-
data">
                {% csrf_token %}
                <div class="col-md-12">
                    <fieldset>
                        <input name="name" type="text" class="form-control" id="name"
placeholder="Item Name" value="{{ item.name }}" required oninput="fillUsername()">
                    </fieldset>
                </div>
                <div class="col-md-12">
                    <fieldset>
                        <input name="description" type="text" class="form-control"
id="description" placeholder="Description" value="{{ item.description }}" required hidden>
                    </fieldset>
                </div>

                <div class="col-md-12">
                    {{ form.category | as_crispy_field }}
```

```
                </div>

                <div class="col-md-4">
                    <fieldset>
                        <input name="rate" type="number" class="form-control" id="rate" placeholder="Rate" value="{{ item.rate }}" required>
                    </fieldset>
                </div>

                <div class="col-md-4">
                    <fieldset>
                        <input name="stock" type="number" class="form-control" id="stock" placeholder="Stock" value="{{ item.stock }}" required>
                    </fieldset>
                </div>

                <div class="col-md-4">
                    {{ form.is_non_veg | as_crispy_field }}
                </div>

                <div class="col-md-12">
                    {{ form.image | as_crispy_field }}
                </div>

                <div class="col-md-12">
                    <fieldset>
                        <button type="submit" id="form-submit" class="btn">SAVE</button>
```

```
                        </fieldset>
                    </div>


                </div>
            </form>
        </div>
    </div>
</div>
</section>


{% endblock content %}
```

#Base.html

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

        <title>AgroHub - The Online AgroHub</title>
        <meta name="description" content="">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="apple-touch-icon" href="apple-touch-icon.png">
```

```
    <link rel="stylesheet" href="{% static 'orders/css/bootstrap.min.css' %}">
    <link rel="stylesheet" href="{% static 'orders/css/bootstrap-theme.min.css' %}">
    <link rel="stylesheet" href="{% static 'orders/css/fontAwesome.css' %}">
    <link rel="stylesheet" href="{% static 'orders/css/hero-slider.css' %}">
    <link rel="stylesheet" href="{% static 'orders/css/owl-carousel.css' %}">
    <link rel="stylesheet" href="{% static 'orders/css/templatemo-style.css' %}">

    <link
href="https://fonts.googleapis.com/css?family=Spectral:200,200i,300,300i,400,400i,500,500i,60
0,600i,700,700i,800,800i" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Roboto:100,300,400,500,700,900"
rel="stylesheet">

    <script src="{% static 'orders/js/vendor/modernizr-2.8.3-respond-1.4.2.min.js'
%}"></script>
  </head>

<body>
  <div class="header">
    <div class="container">
      <a href="#" class="navbar-brand scroll-top">AgriFair</a>
      <nav class="navbar navbar-inverse" role="navigation">
        <div class="navbar-header">
          <button type="button" id="nav-toggle" class="navbar-toggle" data-
toggle="collapse" data-target="#main-nav">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span
```

REFERENCE

[1] Understanding the Link between Farmers' Market Size and Management Organization book by Garry Stephenson, Larry Lev, Linda Brewer.

[2] Agricultural Marketing In India book by S.S. Acharya, N.L. Agarwal.

[3] From the farm to your table book by James Thompsors & Adel Kader.

[4] www.digitalgreen.org/

[5] www.mobindustry.net/

[6] https://smallbiztrends.com/2018/09/agricultural-apps.html

[7] www.accessagriculture.org/

[8] https://agritech.tnau.ac.in/agricultural_marketing/agrimark_Farmers%20market.html

[9] https://www.treehugger.com/why-are-farmers-markets-so-popular-1203977

[10] https://www.nal.usda.gov/legacy/afsic/farmers-markets/

[11] https://www.dailyguides.com/webhtml