BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY),

COLLEGE OF ENGINEERING,

PUNE-411043

Department of Information Technology

A

Project Based Learning

On

" Farm Management System "

Submitted By

Name	PRN No.	Roll No.
Sarthak Srivastava	2314110314	37
Shashank Kumar	2314110330	50
Rushikesh Girpunje	2314110678	56
Rahul Kumar	2414111034	66

Under the Guidance of

Prof. Milind D. Gayakwad

DEPARTMENT OF INFORMATION TECHNOLOGY

Certificate

This is to certify that the work under Project Based Learning (PBL) for the topic **“Farm Management System ”** is carried out by Rahul Kumar, Rushikesh Girpunje and Sarthak Srivastava under the guidance of **Prof. Milind D. Gayakwad** in partial fulfillment of the requirement for the degree of **Bachelor of Technology in Information Technology Semester-III** of **Bharati Vidyapeeth (Deemed to be University), Pune** during the academic year **2024-2025**.

Date:- 15|10|2024

Prof. Milind D. Gayakwad
GUIDE

ACKNOWLEDGEMENT

We would like to express deepest appreciation towards Dr Vidula Sohoni, Principal, Bharati Vidyapeeth (Deemed to Be University) College of Engineering, Pune, India and Dr.Sandeep Vanjale Head of the Department (IT), who invaluable supported us in completing this project.

We are profoundly grateful to Prof. Milind D Gayakwad, Project guide for her expert guidance and continuous encouragement throughout to see that this project rights, its target since its commencement to its completion.

At last we must express our sincere heartfelt gratitude to all the staff members of IT Department who helped us directly or indirectly during this course of work.

INDEX

INTRODUCTION

The Farm Management System is designed to create an online platform where farmers can list their agricultural products for sale, and buyers can easily browse and purchase those products. The primary aim of the system is to streamline the process of buying and selling agricultural goods, reducing intermediaries and ensuring that both buyers and farmers benefit from direct communication. One of the system's unique features is its focus on quality assurance, as buyers are able to send purchase requests and verify the quality of the agricultural products through email correspondence. This system not only facilitates easy transactions but also creates transparency between buyers and sellers, fostering trust and enabling smooth operations within the agricultural marketplace.

Project Objective and Key Features

The primary objective of the Farm Management System is to provide an intuitive and secure online marketplace where farmers can sell and purchase agricultural products without hassle. This system enables farmers to list a wide range of agricultural goods, while buyers can browse these offerings and place purchase requests. By integrating email-based communication for quality checks, the system ensures that buyers can verify the quality of the products before finalizing a purchase. Additionally, the system provides administrative tools for managing users and products, thereby ensuring that both farmers and buyers enjoy a smooth experience.

1. **User Authentication**: Secure signup and login for farmers and buyers, with password encryption.
2. **Product Management**: Farmers can add, edit, and manage agricultural products, including descriptions and prices.
3. **Purchase Requests**: Buyers can browse products and place purchase requests, facilitating direct communication with farmers.
4. **Email Notifications**: Automated email alerts for quality checks and updates on purchase requests for buyers.
5. **Feedback System**: Buyers can leave ratings and reviews, promoting product quality and transparency.

Proposed System

The proposed Farm Management System has several important features aimed at simplifying the buying and selling of agricultural products. Farmers can easily register and list their products on the platform, specifying details such as the type of product, quantity, price, and quality standards. Buyers, in turn, can browse these products, compare prices, and request additional information regarding the quality via email before making a purchase decision.

Buyers can send a request to check the quality of products they are interested in, and this is facilitated through an integrated email system that allows for seamless communication between the parties. Both farmers and buyers have separate, secure login areas, with access privileges managed by the administrator. The admin has control over managing user accounts and ensuring the integrity of the system.

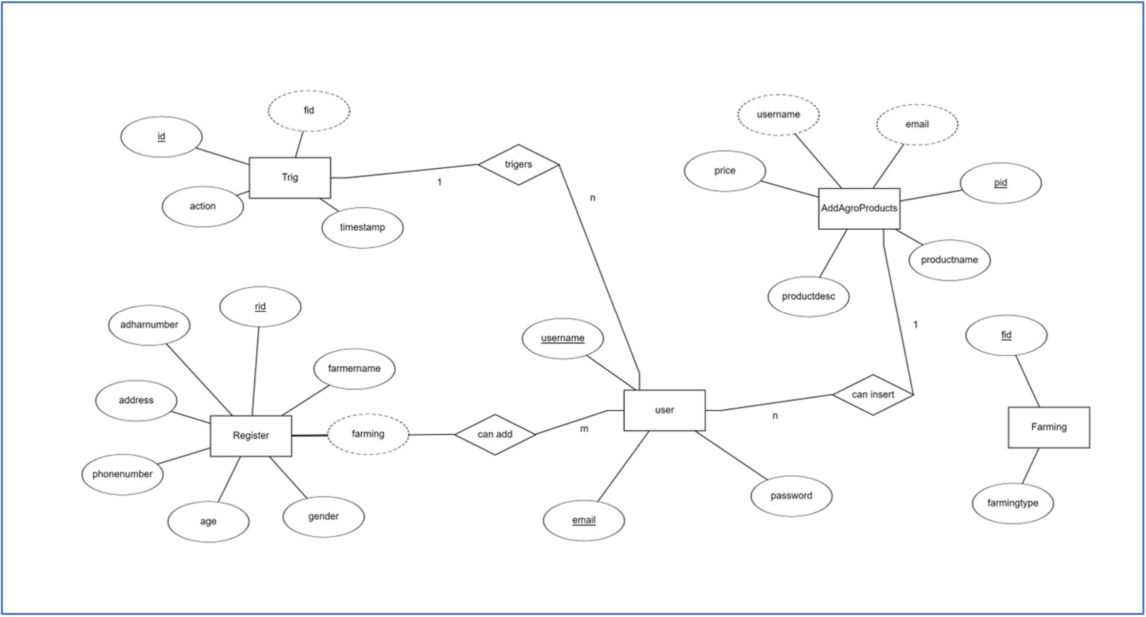
Technologically, the system is built using the Flask web framework, with a backend supported by SQL databases like MySQL. The frontend interface is developed using HTML, CSS, and JavaScript, ensuring a clean and responsive user experience. Furthermore, email services are integrated using an SMTP service to enable quality assurance communications between buyers and sellers.

Database Design

The Farm Management System's database is designed to handle a range of entities, ensuring that the system maintains data integrity and efficiently manages the relationships between users, products, orders, and messages. The database is structured to achieve third normal form (3NF), which reduces redundancy and ensures optimal data management.

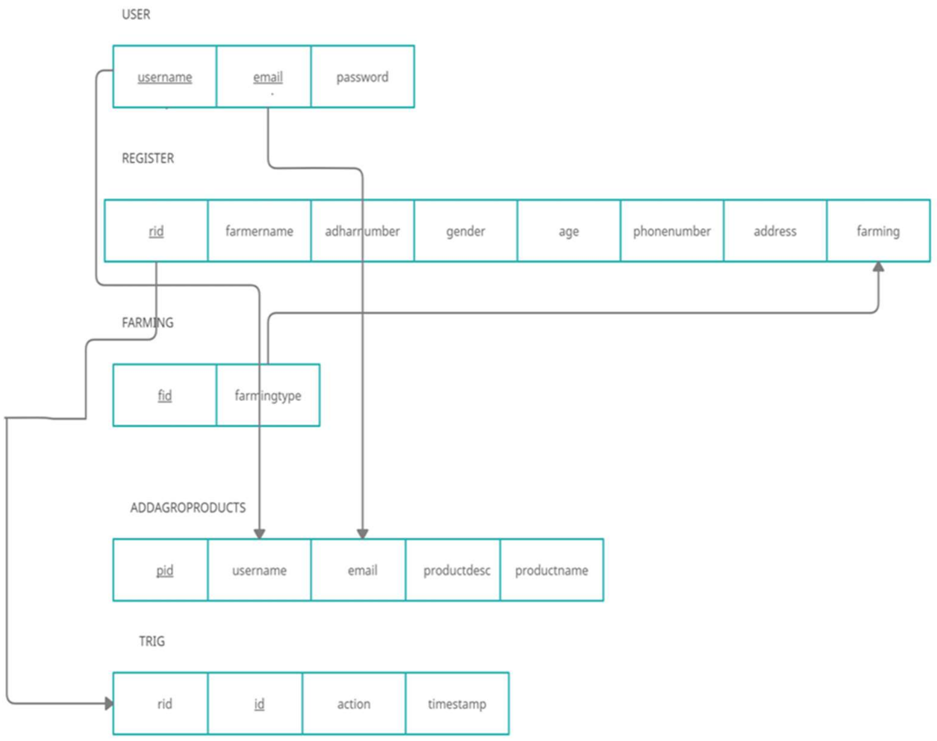
E-R DIAGRAM

The Entity-Relationship (ER) Diagram represents the logical structure of the system.



SCHEMA DIAGRAM

The relational database includes the following primary tables



Implementation

The implementation of the Farm Management System relies heavily on the Python programming language, specifically utilizing the Flask web framework as the primary backend framework. This framework provides the necessary support for developing web applications that execute Python code effectively. The most recognized implementation of Python ,which serves as the basis for this project. While there are several distributions and variants of Python available, this project uses Python to ensure a consistent environment for executing Python programs.

Backend (MySQL)

Database

A Database Management System (DBMS) is essential for managing the structured data within the Farm Management System. In this project, we use MySQL as our DBMS, which is a powerful and widely-used relational database system. A DBMS is a complex set of software programs designed to control the organization, storage, management, and retrieval of data in a database. MySQL provides various capabilities that facilitate effective data management, including:

1. **Modeling Language**: MySQL utilizes SQL (Structured Query Language) as its primary language for defining database schemas. SQL allows for the creation, alteration, and management of database objects like tables and indexes.
2. **Data Structures**: The database contains optimized data structures, including fields, records, files, and objects, designed to handle large amounts of data efficiently. MySQL ensures that data is stored on permanent storage devices while maintaining access speed suitable for user queries.
3. **Data Security**: MySQL implements security measures to prevent unauthorized access to the database. It uses passwords to control access, ensuring that users can only view or update data according to their privileges. This allows for the creation of sub-schemas, enabling specific user groups to access only the information relevant to their roles.
4. **Transaction Management:** MySQL supports ACID (Atomicity, Consistency, Isolation, Durability) properties to guarantee data integrity, even with concurrent user access. This

feature is crucial for ensuring that data remains consistent and reliable throughout the transaction process.

5. **Concurrency Control:** The DBMS prevents multiple users from updating the same record simultaneously, which helps maintain data integrity. Additionally, unique index constraints prevent the entry of duplicate records, ensuring that each entry is distinct.

By using MySQL, the Farm Management System can efficiently manage and retrieve data while maintaining security and integrity. The database is designed to accommodate the system's various operations, allowing for easy modifications as the organization's information requirements evolve.

SQL

SQL is the language used to manipulate relational databases, closely tied to the relational model. The Farm Management System uses SQL to perform various operations, including:

Data Definition: SQL is employed for defining the schema of the database. This includes creating tables, altering their structures, and managing the overall database architecture through Data Definition Language (DDL).

Example SQL Statement

```
CREATE TABLE register (
   rid INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL,
   role ENUM('farmer', 'buyer') NOT NULL,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Stored Procedure

Stored procedures are precompiled collections of SQL statements that can be executed as needed. They improve performance by reducing the amount of code sent over the network and allow for reusable operations within the database.

Routine Name: proc

Type: Procedure

Definition:

```
CREATE PROCEDURE proc AS
BEGIN
    SELECT * FROM register;
END;
```

Triggers

Triggers are special types of stored procedures that automatically execute in response to specific events occurring within the database. In the Farm Management System, we have implemented several triggers to monitor actions on the register table.

Trigger Name: on insert

Table: register

Time: AFTER

Event: INSERT

Definition:

```
CREATE TRIGGER on_insert AFTER INSERT ON register
FOR EACH ROW
BEGIN
    INSERT INTO trig VALUES (NULL, NEW.rid, 'Farmer Inserted', NOW());
END;
```

Trigger Name: on delete

Table: register

Time: AFTER

Event: DELETE

Definition:

```
CREATE TRIGGER on_delete AFTER DELETE ON register
FOR EACH ROW
BEGIN
    INSERT INTO trig VALUES (NULL, OLD.rid, 'FARMER DELETED', NOW());
END;
```

Trigger Name: on update

Table: register

Time: AFTER

Event: UPDATE

Definition:

```
CREATE TRIGGER on_update AFTER UPDATE ON register
FOR EACH ROW
BEGIN
    INSERT INTO trig VALUES (NULL, NEW.rid, 'FARMER UPDATED', NOW());
END;
```

Backend Python Code with MySQL

The backend of the Farm Management System is implemented using Python with the Flask framework, allowing for the creation of a dynamic web application. Below is an example of how to connect to the MySQL database and perform basic CRUD operations.

Code (app.py)

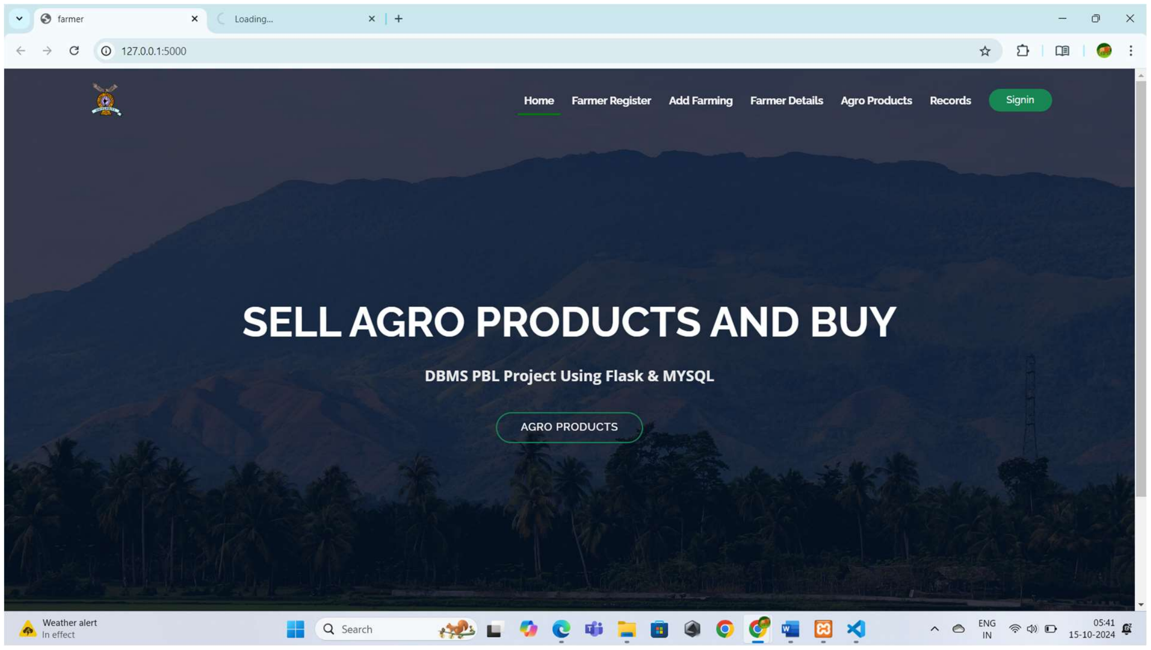
```
from flask import Flask,render_template,request,session,redirect,url_for,flash
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from werkzeug.security import generate_password_hash,check_password_hash
from flask_login import login_user,logout_user,login_manager,LoginManager
from flask_login import login_required,current_user

# MY db connection
local_server= True
app = Flask(__name__)
app.secret_key='rahulkumar'

# this is for getting unique user access
login_manager=LoginManager(app)
login_manager.login_view='login'

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

#app.config['SQLALCHEMY_DATABASE_URL']='mysql://username:password@localhost/
#databas__name'
app.config['SQLALCHEMY_DATABASE_URI']='mysql://root:@localhost/farmers'
db=SQLAlchemy(app)

# here we will create db models that is tables
class Test(db.Model):
    id=db.Column(db.Integer,primary_key=True)
    name=db.Column(db.String(100))

class Farming(db.Model):
    fid=db.Column(db.Integer,primary_key=True)
    farmingtype=db.Column(db.String(100))

class Addagroproducts(db.Model):
    username=db.Column(db.String(50))
    email=db.Column(db.String(50))
    pid=db.Column(db.Integer,primary_key=True)
    productname=db.Column(db.String(100))
    productdesc=db.Column(db.String(300))
    price=db.Column(db.Integer)


class Trig(db.Model):
    id=db.Column(db.Integer,primary_key=True)
    fid=db.Column(db.String(100))
    action=db.Column(db.String(100))
    timestamp=db.Column(db.String(100))

class User(UserMixin,db.Model):
    id=db.Column(db.Integer,primary_key=True)
    username=db.Column(db.String(50))
    email=db.Column(db.String(50),unique=True)
    password=db.Column(db.String(1000))

class Register(db.Model):
    rid=db.Column(db.Integer,primary_key=True)
    farmername=db.Column(db.String(50))
    adharnumber=db.Column(db.String(50))
    age=db.Column(db.Integer)
    gender=db.Column(db.String(50))
    phonenumber=db.Column(db.String(50))
    address=db.Column(db.String(50))
    farming=db.Column(db.String(50))



@app.route('/')
def index():
    return render_template('index.html')
```

```
@app.route('/farmerdetails')
@login_required
def farmerdetails():
    # query=db.engine.execute(f"SELECT * FROM `register`")
    query=Register.query.all()
    return render_template('farmerdetails.html',query=query)

@app.route('/agroproducts')
def agroproducts():
    # query=db.engine.execute(f"SELECT * FROM `addagroproducts`")
    query=Addagroproducts.query.all()
    return render_template('agroproducts.html',query=query)

@app.route('/addagroproduct',methods=['POST','GET'])
@login_required
def addagroproduct():
    if request.method=="POST":
        username=request.form.get('username')
        email=request.form.get('email')
        productname=request.form.get('productname')
        productdesc=request.form.get('productdesc')
        price=request.form.get('price')
         products=Addagroproducts(username=username,email=email,productname=productnam
e,productdesc=productdesc,price=price)
        db.session.add(products)
        db.session.commit()
        flash("Product Added","info")
        return redirect('/agroproducts')

    return render_template('addagroproducts.html')

@app.route('/triggers')
@login_required
def triggers():
    # query=db.engine.execute(f"SELECT * FROM `trig`")
    query=Trig.query.all()
    return render_template('triggers.html',query=query)
```

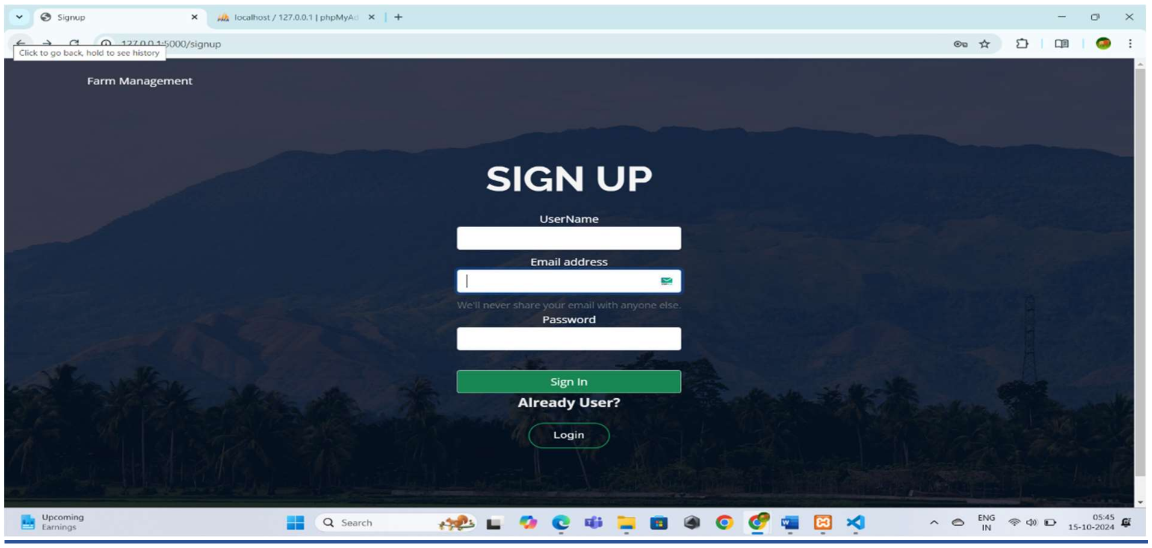
```
@app.route('/addfarming',methods=['POST','GET'])
@login_required
def addfarming():
    if request.method=="POST":
        farmingtype=request.form.get('farming')
        query=Farming.query.filter_by(farmingtype=farmingtype).first()
        if query:
            flash("Farming Type Already Exist","warning")
            return redirect('/addfarming')
        dep=Farming(farmingtype=farmingtype)
        db.session.add(dep)
        db.session.commit()
        flash("Farming Addes","success")
    return render_template('farming.html')



@app.route("/delete/<string:rid>",methods=['POST','GET'])
@login_required
def delete(rid):
    # db.engine.execute(f"DELETE FROM `register` WHERE `register`.`rid`={rid}")
    post=Register.query.filter_by(rid=rid).first()
    db.session.delete(post)
    db.session.commit()
    flash("Slot Deleted Successful","warning")
    return redirect('/farmerdetails')

@app.route("/edit/<string:rid>",methods=['POST','GET'])
@login_required
def edit(rid):
    # farming=db.engine.execute("SELECT * FROM `farming`")
    if request.method=="POST":
        farmername=request.form.get('farmername')
        adharnumber=request.form.get('adharnumber')
        age=request.form.get('age')
        gender=request.form.get('gender')
        phonenumber=request.form.get('phonenumber')
        address=request.form.get('address')
        farmingtype=request.form.get('farmingtype')
        # query=db.engine.execute(f"UPDATE `register` SET `farmername`='{farmername}',`adharnumber`='{adharnumber}',`age`='{age}',`gender`='{gender}',`phonenumber`='{phonenumber}',`address`='{address}',`farming`='{farmingtype}'")
        post=Register.query.filter_by(rid=rid).first()
        print(post.farmername)
        post.farmername=farmername
        post.adharnumber=adharnumber
        post.age=age
        post.gender=gender
        post.phonenumber=phonenumber
        post.address=address
        post.farming=farmingtype
        db.session.commit()
        flash("Slot is Updates","success")
        return redirect('/farmerdetails')
    posts=Register.query.filter_by(rid=rid).first()
    farming=Farming.query.all()
    return render_template('edit.html',posts=posts,farming=farming)

@app.route('/signup',methods=['POST','GET'])
def signup():
    if request.method == "POST":
        username=request.form.get('username')
        email=request.form.get('email')
        password=request.form.get('password')
        print(username,email,password)
        user=User.query.filter_by(email=email).first()
        if user:
            flash("Email Already Exist","warning")
            return render_template('/signup.html')
        # encpassword=generate_password_hash(password)

        # new_user=db.engine.execute(f"INSERT INTO `user` (`username`,`email`,`password`) VALUES ('{username}','{email}','{encpassword}')")

        # this is method 2 to save data in db
        newuser=User(username=username,email=email,password=password)
        db.session.add(newuser)
        db.session.commit()
        flash("Signup Succes Please Login","success")
        return render_template('login.html')



    return render_template('signup.html')

@app.route('/login',methods=['POST','GET'])
def login():
    if request.method == "POST":
        email=request.form.get('email')
        password=request.form.get('password')
        user=User.query.filter_by(email=email).first()

        if user and user.password == password:
            login_user(user)
            flash("Login Success","primary")
            return redirect(url_for('index'))
        else:
            flash("invalid credentials","warning")
            return render_template('login.html')

    return render_template('login.html')

@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash("Logout SuccessFul","warning")
    return redirect(url_for('login'))


@app.route('/register',methods=['POST','GET'])
@login_required
def register():
    farming=Farming.query.all()
    if request.method=="POST":
        farmername=request.form.get('farmername')
        adharnumber=request.form.get('adharnumber')
        age=request.form.get('age')
        gender=request.form.get('gender')
        phonenumber=request.form.get('phonenumber')
        address=request.form.get('address')
        farmingtype=request.form.get('farmingtype')
         query=Register(farmername=farmername,adharnumber=adharnumber,age=age,gender=g
ender,phonenumber=phonenumber,address=address,farming=farmingtype)
        db.session.add(query)
        db.session.commit()
                                #       query=db.engine.execute(f"INSERT       INTO       `register`
(`farmername`,`adharnumber`,`age`,`gender`,`phonenumber`,`address`,`farming`)     VALUES
('{farmername}','{adharnumber}','{age}','{gender}','{phonenumber}','{address}','{farmingtyp
e}')")
        # flash("Your Record Has Been Saved","success")
        return redirect('/farmerdetails')
    return render_template('farmer.html',farming=farming)

@app.route('/test')
def test():
    try:
        Test.query.all()
        return 'My database is Connected'
    except:
        return 'My db is not Connected'

app.run(debug=True)
```

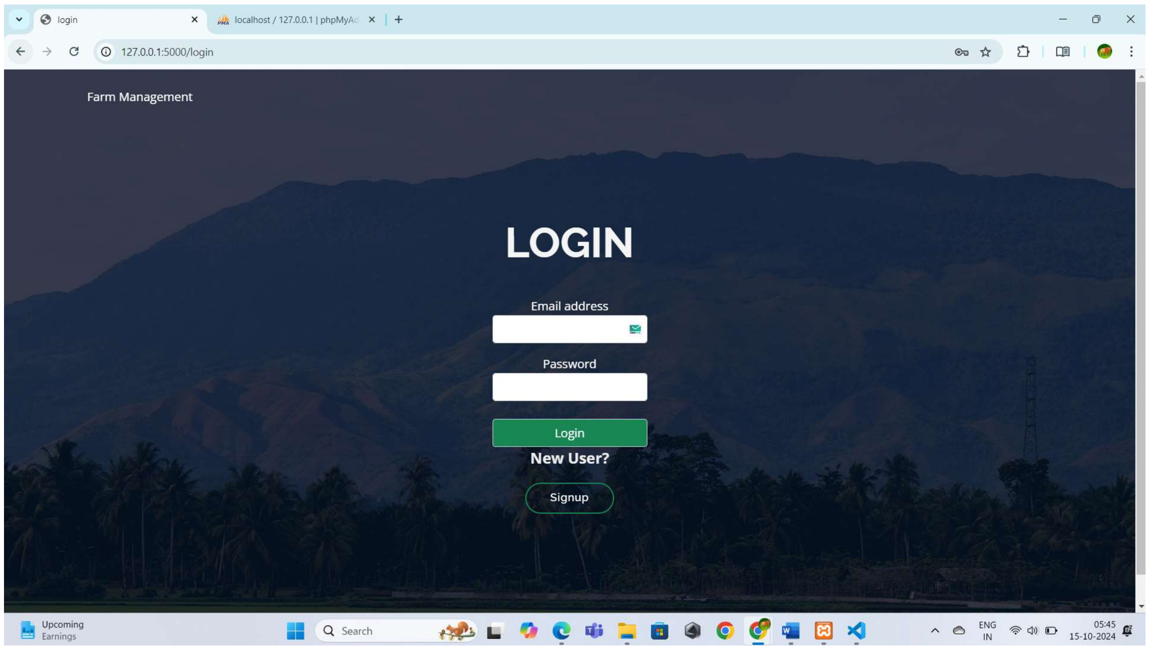
User interface

The user interface is designed to be simple and intuitive, with clear navigation paths for both farmers and buyers. The system ensures that both parties can perform their respective tasks efficiently.

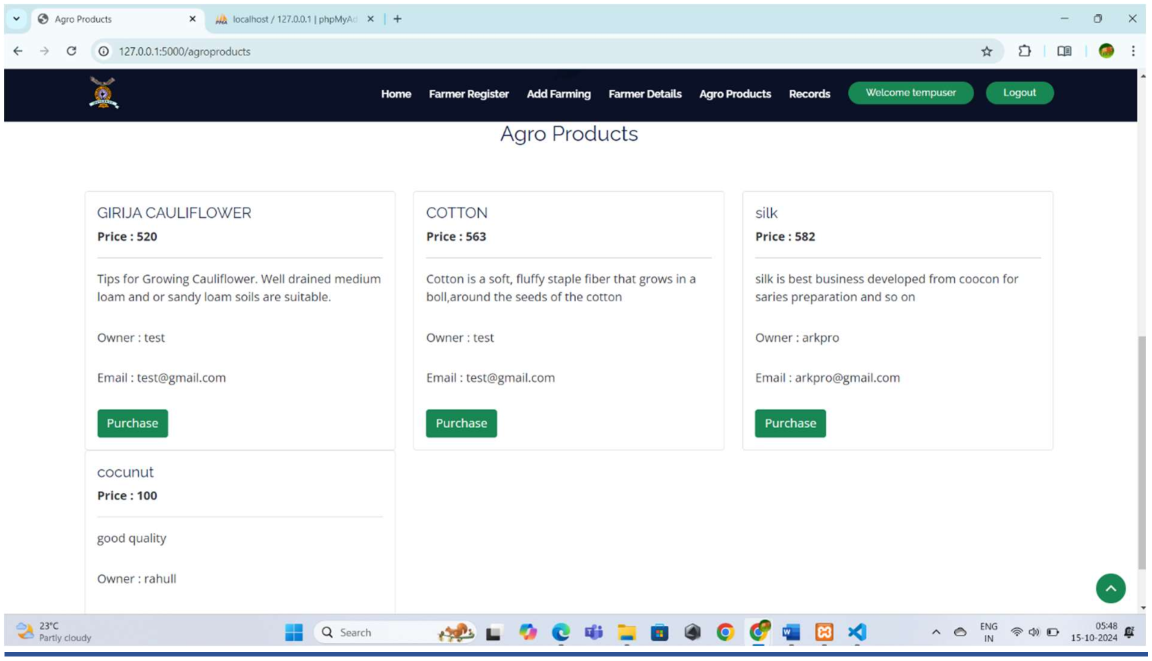


Sign In Page





Agro product



Agro Products

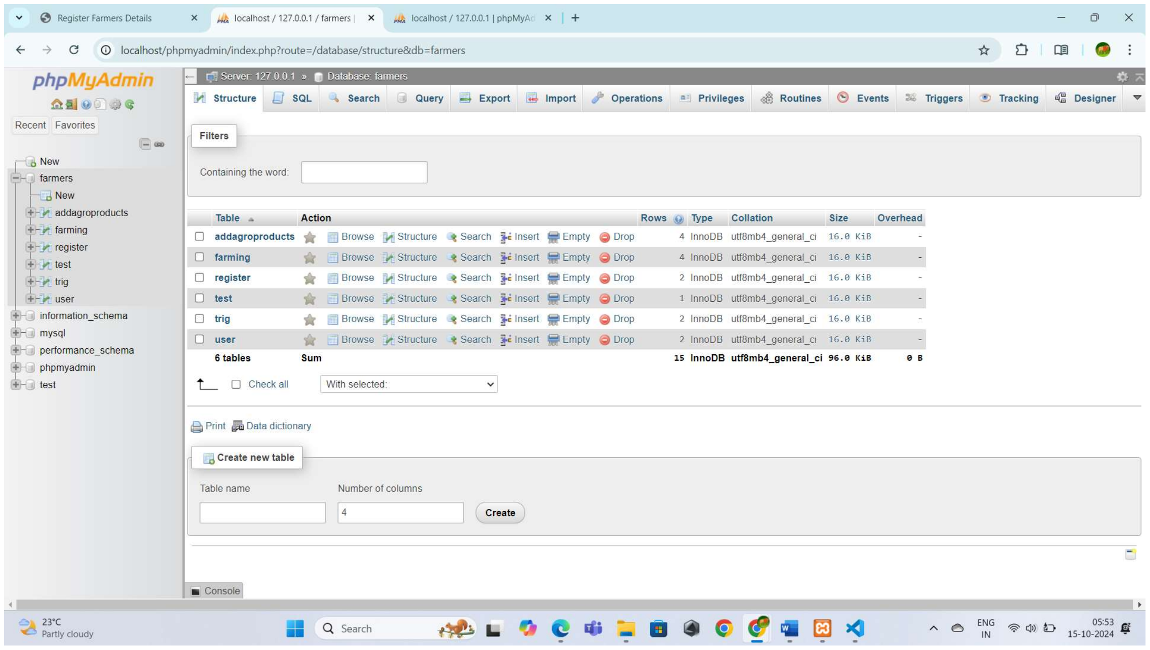
GIRIJA CAULIFLOWER
Price : 520
Tips for Growing Cauliflower. Well drained medium loam and or sandy loam soils are suitable.
Owner : test
Email : test@gmail.com
Purchase

COTTON
Price : 563
Cotton is a soft, fluffy staple fiber that grows in a boll,around the seeds of the cotton
Owner : test
Email : test@gmail.com
Purchase

silk
Price : 582
silk is best business developed from coocon for saries preparation and so on
Owner : arkpro
Email : arkpro@gmail.com
Purchase

cocunut
Price : 100
good quality
Owner : rahull

Database:



localhost/phpmyadmin/index.php?route=/database/structure&db=farmers

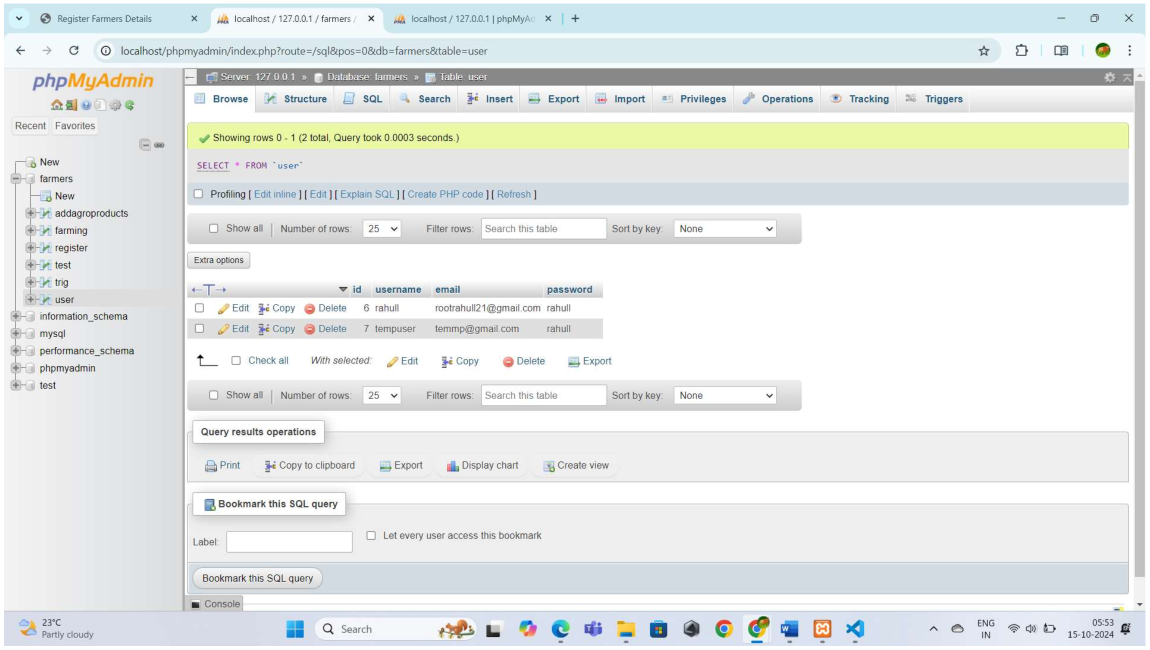
Server: 127.0.0.1 » Database: farmers

	Table		Action						Rows	Type	Collation	Size	Overhead
☐	addagroproducts	☆	Browse	Structure	Search	Insert	Empty	Drop	4	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	farming	☆	Browse	Structure	Search	Insert	Empty	Drop	4	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	register	☆	Browse	Structure	Search	Insert	Empty	Drop	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	test	☆	Browse	Structure	Search	Insert	Empty	Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	trig	☆	Browse	Structure	Search	Insert	Empty	Drop	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	user	☆	Browse	Structure	Search	Insert	Empty	Drop	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
	6 tables		Sum						15	InnoDB	utf8mb4_general_ci	96.0 KiB	0 B

Create new table

Table name:

Number of columns: 4



localhost/phpmyadmin/index.php?route=/sql&pos=0&db=farmers&table=user

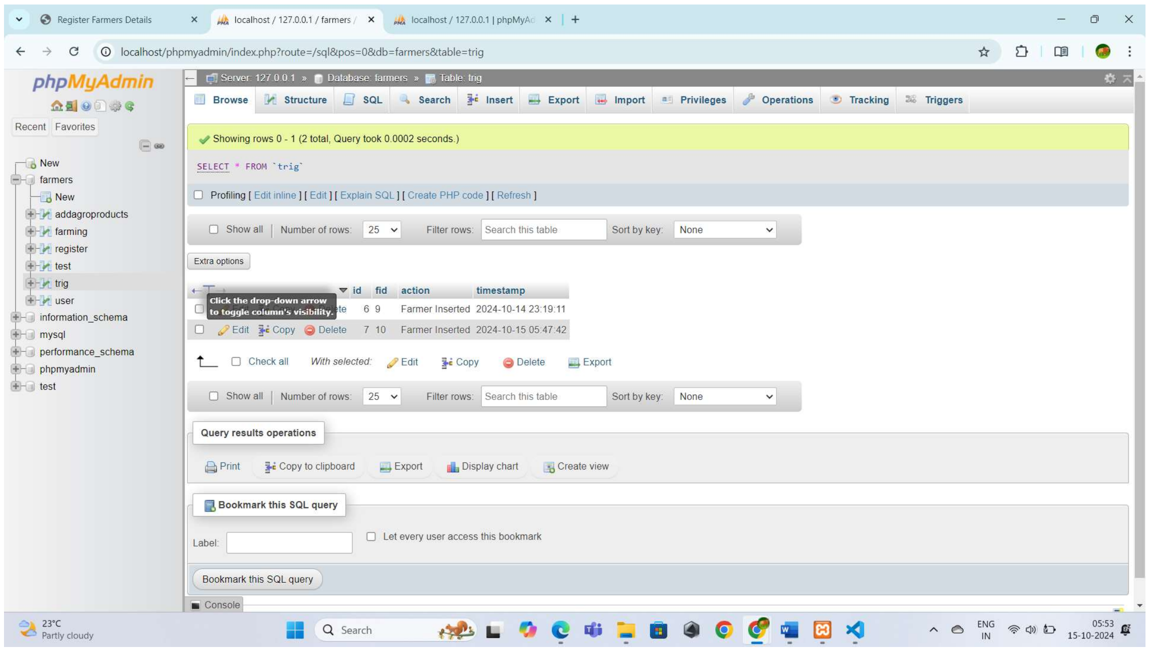
Server: 127.0.0.1 » Database: farmers » Table: user

Showing rows 0 - 1 (2 total, Query took 0.0003 seconds.)

```
SELECT * FROM `user`
```

Number of rows: 25 Filter rows: Search this table Sort by key: None

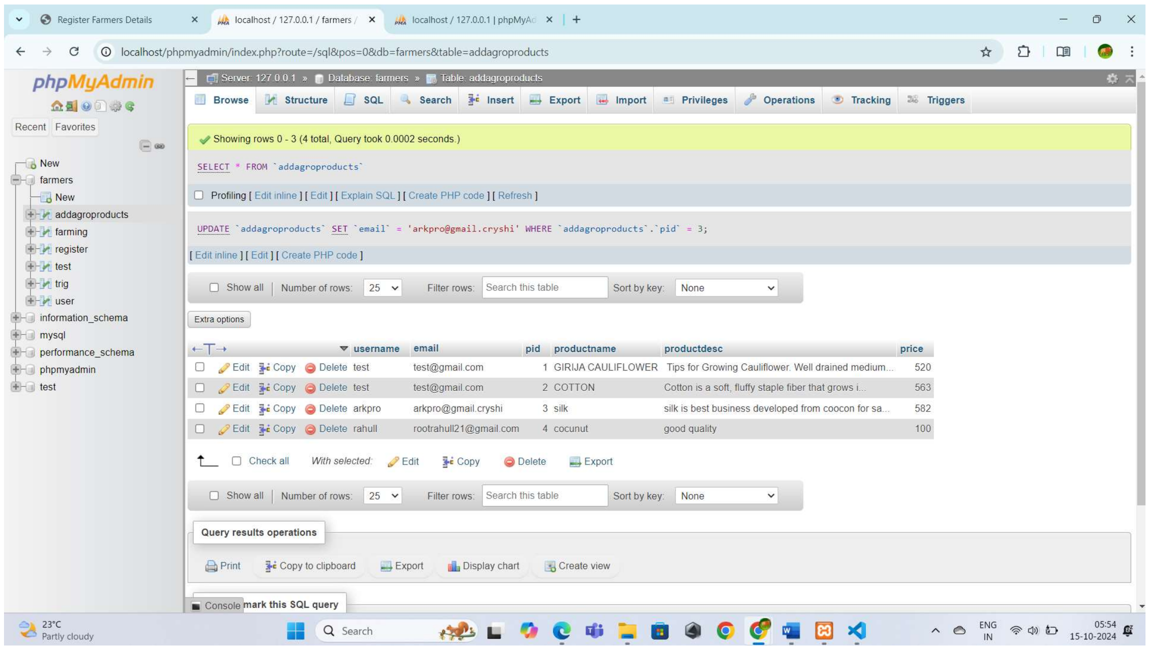
				id	username	email	password
☐	Edit	Copy	Delete	6	rahull	rootrahull21@gmail.com	rahull
☐	Edit	Copy	Delete	7	tempuser	temmp@gmail.com	rahull



Showing rows 0 - 1 (2 total, Query took 0.0002 seconds.)

```
SELECT * FROM `trig`
```

id	fid	action	timestamp
6	9	Farmer Inserted	2024-10-14 23:19:11
7	10	Farmer Inserted	2024-10-15 05:47:42



Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)

```
SELECT * FROM `addagroproducts`
```

```
UPDATE `addagroproducts` SET `email` = 'arkpro@gmail.cryshi' WHERE `addagroproducts`.`pid` = 3;
```

username	email	pid	productname	productdesc	price
test	test@gmail.com	1	GIRIJA CAULIFLOWER	Tips for Growing Cauliflower. Well drained medium...	520
test	test@gmail.com	2	COTTON	Cotton is a soft, fluffy staple fiber that grows i...	563
arkpro	arkpro@gmail.cryshi	3	silk	silk is best business developed from coocon for sa...	582
rahull	rootrahull21@gmail.com	4	cocunut	good quality	100

Conclusion:

The Farm Management System provides a user-friendly platform for farmers to sell and buyers to purchase agricultural products efficiently. The integration of email-based quality checks fosters transparency and builds trust between buyers and farmers, allowing informed purchasing decisions.

With a robust backend built on Flask and MySQL, the system ensures scalability and secure data management. Flask's lightweight framework enables quick deployment, while MySQL supports efficient data storage and retrieval, maintaining transaction integrity through triggers and stored procedures.

The user interface is designed for ease of use, facilitating seamless interactions for both farmers and buyers. Future enhancements could include a mobile application for greater accessibility and machine learning algorithms for personalized product recommendations.

References

1. https://github.com/manojs15/Farm-mini-project

Farm management system using Django, Python, HTML, CSS, and PostgreSQL. This project efficiently tracks farmer and crop data and has a user-friendly interface.

2. https://link.springer.com/article/10.1007/s11119-021-09847-3

This academic reference details the design of farm management information systems (FMIS) and includes feature modelling, software architecture design, and stakeholder concerns for farm management projects. It's a useful study for understanding the system architecture of farm management systems.

3. https://inria.hal.science/hal-01274751/document
   Conceptual Farm Management Information System: This paper presents a detailed model for farm management, focusing on decision support through data collection from various external services, including GIS data, remote sensors, and data loggers. It highlights how information about crops, land management, and field operations can be automated and digitized for better farm productivity. Read more about it here: Farm Management System Conceptual Model

4. https://www.mdpi.com/2311-7524/10/1/108
   Management Information Systems for Orchard Management: This article focuses on the application of MIS in fruit farming, covering management aspects such as pest control, irrigation, and harvest forecasting. It also discusses the technical features needed for effective adoption in large-scale agricultural setups. This review can provide useful insights into how a farm management system can be structured for broader agricultural applications. You can check it out

5. https://www.mdpi.com/2311-7524/10/1/108
   A Conceptual Model of Farm Management Information System for Decision Support: This source discusses a Farm Management Information System (FMIS) that helps farmers make decisions by integrating information from various services like geographic data, GPS for precision farming, and tools for analyzing soil and crop productivity. The system supports tasks such as creating spatial distribution maps, crop productivity assessments, and resource management. You can find more details here