YIELDSMART:A CROP YIELD PREDICTION SYSTEM

BCA PROJECT REPORT 2025

SUBMITTED BY

AHAMMAD BASITH
YOUSUF SAAD
AKSHAY KUMAR
MUHAMMED SABITH

HAJI C.H.M.M COLLEGE FOR ADVANCED STUDIES
CHAVARCODE,PALAYAMKUNNU P.O-695146
THIRUVANANTHAPURAM DISTRICT
KERALA

PROJECT REPORT
SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE AWARD OF
BCA DEGREE OF
UNIVERSITY OF KERALA
2025

HAJI C.H.M.M COLLEGE FOR ADVANCED STUDIES

CHAVARCODE,PALAYAMKUNNU P.O--695146

THIRUVANANTHAPURAM DISTRICT

KERALA

DEPARTMENT OF BCA

CERTIFICATE

Certified that this project report titled "**YIELDSMART : A Crop Yield Prediction System**" is a bonafide record of the project work done by **AHAMMAD BASITH (Reg.no:33222801005), AKSHAY KUMAR (Reg.no:33222801010), MUHAMMED SABITH (Reg no:33222801040), YOUSUF SAAD (Reg.no:33222801055)**,under our guidance and supervision towards partial fulfillment of the requirements for the award of BCA Degree of University of Kerala.

Internal Examiner **Head of the Institution**

External Examiner

1.

2.

ACKNOWLEDGEMENT

Our project work stands incomplete without dedicating my gratitude to those few who have contributed a lot towards the successful completion of this project.

We express our sincere gratitude towards our honorable Principal, **Prof.(Dr) L.Thulaseedharan** of **Haji C.H.Mohammed Koya Memorial College For Advanced Studies** for providing an opportunity for doing this project work.

We also extend our special thanks to **Mrs.Ajina M.Ameer, Assistant Professor and Head of the Department (BCA)** for her expert and valuable guidance, inspiration and fruitful discussions rendered throughout for successful completion of the project work.

We take this opportunity to express our sincere gratitude and indebtedness to our internal guide **Mrs. Amritha.A.L, Assistant Professor , Department of BCA** for providing all possible facilities to make this project a success.

We express our cordial and sincere thanks to our teachers **Mrs. Rajani.R , Mrs. Daniya Jijo and Mrs. Rahma**, Department of BCA , for valuable advice and great help in completing our project.

With great pleasure we may record our deep gratitude to all staff members of BCA department for immeasurable help rendered to us during the course of the project We express our friends and teachers of BCA Department for their encouragement, support and love.

With Gratitude

Ahammad Basith
Yousuf Saad
Akshay Kumar
Muhammed Sabith

YIELDSMART:A CROP YIELD PREDICTION SYSTEM

(USING MACHINE LEARNING)

SUBMITTED BY

Ahammad Basith(33222801005)
Yousuf Saad(33222801055)
Akshay Kumar(33222801010)
Mohammed Sabith(33222801040)

CONTEXT

1.INTRODUCTION

ABSTRACT

The Crop Yield Prediction System is a data-driven web application aimed at helping farmers predict crop yields based on weather conditions and soil data. By utilizing machine learning algorithms, the platform processes data from weather APIs and soil sensors to generate accurate predictions, providing insights into how environmental factors affect crop production. The system assists farmers in making informed decisions about irrigation, fertilization, and harvest timing, ultimately improving productivity and sustainability. Built with Django for the backend and machine learning models for yield prediction, this platform is designed to help optimize agricultural practices

MODULES:

1. User Module (Farmers & Administrators)

Farmer Registration and Profile:

- Farmers can create and manage their accounts, including farm location and crop types.

Weather and Soil Data Input:

- Farmers can enter real-time weather conditions and soil data (e.g., moisture, pH, temperature) for yield predictions

Yield Prediction Dashboard:

- A personalized dashboard that displays predicted crop yields, historical data, and related weather conditions.

Recommendations:

- Actionable insights based on yield predictions, including optimal irrigation and fertilization strategies.

Historical Data Visualization:

- Access to graphs showing past predictions and actual yield results for better decision-making.

2. Admin Module

Admin Dashboard:

- A central dashboard to monitor the overall activity, including registered users, farms, and data inputs.

Weather and Soil Data Management:

- Admin can manage the weather data sources and ensure accuracy in the dataset.

- **Model Training and Monitoring:**
    - Admin can trigger the training of machine learning models using new data and track prediction accuracy.
- **Farm Data Monitoring:**
    - Monitor and audit farm inputs and predict crop yields across various regions.

3. Data Collection Module

- **External Weather API Integration:**
    - Real-time integration with external weather data providers to fetch information like temperature, humidity, and rainfall.
- **Soil Sensor Data Collection:**
    - Integration with IoT devices for soil monitoring, such as moisture, pH, and temperature.
- **Data Storage and Management:**
    - Store weather and soil data in Django models, ensuring proper relationships with farm and crop data.

4. Machine Learning and Prediction Module

- **Supervised Learning Model:**
    - Build and train machine learning models using historical weather and crop yield data (e.g., Linear Regression, Random Forests).
- **Prediction Engine:**
    - Use trained models to predict crop yields based on real-time weather and soil data inputs from farmers
- **Prediction Accuracy Feedback:**
    - Track the accuracy of predictions and fine-tune models for continuous improvement.

5. Visualization and Reporting Module

- **Yield Prediction Visualization:**
    - Present yield predictions using graphs and charts for easy interpretation.
- **Yield Prediction Visualization:**
    - Generate reports summarizing crop yields, weather conditions, and farming recommendations.

6. Notification Module

- **Alert System:**
  - Notify farmers about significant changes in weather conditions that could affect crop yields
- **Recommendation Alerts:**
  - Automated alerts for when farmers should take action (e.g., irrigation,fertilization) based on prediction models.

7. API Module

- **RESTful API for Data Fetching:**
  - Develop APIs to fetch weather data from external sources and provide crop yield predictions to mobile or external applications.
- **Integration with External Applications:**
  - Allow integration with other agricultural apps or platforms for broader data access.

USER WISE MODULES:

ADMIN MODULES:

- Farmer Control
- Complaint management
- Product control
- Online managing
- Subsidy related issue management

FARMER MODULES:

- Registration
- Product adding
- Booking checking
- Payment checking
- Subsidy products purchasing
- Communication
- Plant disease detection using algorithm
- Yield prediction using algorithm
- repayment

PUBLIC USER MODULES:

- Registration
- Purchase
- Payment
- Complaints
- Status adding

GOVERNMENT BODY MODULES:

- Farmer Details verification
- Document uploading
- Notification passing
- Communication
- Subsidy products adding
- Subsidy collection
- NOC Passing
- Final status confirmation

DELIVERY TEAM MODULES:

- Registration
- Availability adding
- Request Checking
- Address checking
- Delivery status adding
- Payment checking
- Return notification adding

2. COMPANY PROFILE

Softzane Solutions pvt ltd is a 7 year old IT company incorporated on 13-September-2016, having its registered office located at 2nd floor Christuraj Shopping Complex Anchal Road Ayoor Kollam Kerala. A company managed by highly professional and experienced team with a group of truly dedicated, hardworking, and sincere young professionals.

The major activity of Softzane Solutions offers a complex spectrum of custom software development services. Since entering the IT industry, we have gained exceptional experience in software development on Smartphone technologies as well as web application development,they make sure that their clients get leading innovative solutions that are both cost-efficient and dependable. They have an effective solution providing service team and an innovative management to organize the client solutions. Since inception, this institute has constantly worked towards raising the technical skills and standards in pursuit to offer the best technical infrastructure to the budding engineers. We have grown technically sound with state-of-the-art technical labs, equipment's and best infrastructure for academic and research activities

Today's managers and engineers need applicable knowledge and skills to navigate their organization through an ever-changing dynamic environment. We aim to develop the creativity of each student by providing them a motivating and stimulating environment enabling the students to look forwarding to a bright future with meaningful existence.We develop their skills so that they can contribute to the rapidly changing technology requirements making them ready for any competition in the industry. This is achieved through long years of perseverance, persistence and perspiration which make us a cynosure of other organization of similar nature.This has led to our acceptance as an excellent institution providing internship programs in campus training and skill up-gradation training with a view to enhance their employability in electronics as well as information technology.We offer various skill up-gradation programs to students during their semester breaks or as per the time allotted to various institutions at their own campus enabling each student to become job ready.

Softzane Solutions is a dynamic and innovative technology company specializing in providing software solutions tailored to meet the unique needs of its clients. With a focus on cutting-edge technologies and a commitment to customer satisfaction, Softzane Solutions aims to deliver high-quality products and services that drive business growth and efficiency.

Mission Statement:

Our mission at Softzane Solutions is to empower businesses with innovative software solutions that streamline operations, enhance productivity, and drive success. We are dedicated to delivering exceptional value to our clients through continuous innovation, collaboration, and excellence in execution. Your paragraph text

Core value :

Innovation: We embrace creativity and strive for continuous improvement in everything we do.

Integrity: We conduct business with honesty, transparency, and ethical standards.

Customer Centricity: We prioritize understanding and exceeding our clients' expectations to build long-term partnerships

Collaboration: We believe in the power of teamwork and collaboration to achieve shared goals.

Excellence: We are committed to delivering high-quality solutions and services that exceed industry standards

Softzane Solutions offers a comprehensive range of services, including:

Custom Software Development

- Internship Programmes
- Web and Mobile Application Development
- Enterprise Software Solutions
- Cloud Computing Services
- Data Analytics and Business Intelligence
- IT Consulting and Advisory Services
- Software Maintenance and Support

3. REQUIREMENT SPECIFICATION

3.1 SOFTWARE SPECIFICATION

A software requirements specification (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system. The software requirement specification document consistent of all necessary requirements required for project development. Your paragraph text

3.2 HARDWARE SPECIFICATION

The hardware for the system is selected considering the factors such as CPU processing speed, memory access speed, peripheral channel speed, printer speed; seek time & relational delay of hard disk and communication speed etc.

The hardware specifications are as follows:

Processor : Intel Pentium Dual Core

Frequency : >1.5 Ghz

Ram: 4 GB

Peripherals:

- Keyboard
- Mouse
- Monitor

3.3 SOFTWARE SPECIFICATION

The software requirement for the system is selected considering the factors such as platform independency, robustness, security, etc. When we use a technology, we should use the emerging technology. Because when we use any old technology, then it will affect the market and our system.

The minimum software specifications are as follows:

Operating System : Windows 8 or Linux

Browser: Any with javascript enabled

IDE: Visual Studio Code

Front End: HTML, CSS, Javascript

Server Side: Python

Database: SQL Lite3

4. SYSTEM ANALYSIS

4.1. Existing system

The existing system for crop yield prediction involves traditional farming practices and limited use of technology. Farmers primarily rely on personal experience, intuition, and historical farming methods to make decisions about irrigation, fertilization, and harvesting. While some farmers may use basic weather updates or soil testing, the process lacks integration, automation, and data-driven insights. Additionally, tools or systems currently available for yield prediction are either too expensive, complex to use, or not tailored to specific local farming conditions.

The use of technology in the current system is minimal and fragmented. Weather information is often accessed through separate sources, while soil conditions might be assessed manually or through outdated methods. There is no unified platform to combine these inputs and provide actionable insights, resulting in inefficient resource utilization, poor productivity, and an increased risk of crop failure. The absence of predictive analytics further hinders the ability to plan for optimal yields.

some drawbacks of existing system are:

- **Limited Precision:** Predictions may not be accurate at the field level due to local variations.
- **Dependence on Historical Data:** Models rely heavily on past data, which may not account for new variables
- **Lack of Localized Data:** Generalized models fail to consider field-specific conditions.
- **Limited Real-Time Updates:** Predictions are often based on outdated or infrequent data.
- **Cost and Complexity of Data Collection:** Gathering necessary data can be expensive and resource-intensive.

4.2 Proposed system

A proposed system for crop yield prediction using soil and weather data integrates real-time data collection, machine learning models, and decision support tools to generate accurate and localized predictions. By gathering weather data (temperature, rainfall, humidity, etc.) and soil characteristics (moisture content, pH, nutrients) through sensors, satellites, and other technologies, the system processes and analyzes these data to create predictive models. These models, trained on historical yield data, weather patterns, and soil properties, can provide short-term and long-term yield forecasts. The system also offers farmers actionable insights, such as optimal planting and irrigation schedules, and can be accessed through mobile apps or cloud platforms for real-time updates. By adapting predictions based on continuous feedback, the system helps farmers adjust practices to maximize yield potential and minimize risks associated with unpredictable weather or soil conditions.

Advantages:

- **Improved Accuracy:** Provides more precise predictions by considering both weather and soil factors.
- **Localized Insights:** Tailors predictions to specific fields, accounting for regional and microclimate differences.
- **Proactive Decision-Making:** Helps farmers optimize planting, irrigation, and fertilization schedules based on forecasts.
- **Real-Time Updates:** Allows for continuous monitoring and adjustment of predictions as weather and soil conditions evolve.
- **Increased Yield:** By providing actionable insights, the system helps improve crop productivity and resource efficiency.
- **Accessible Technology:** Can be integrated into mobile platforms, making it accessible to farmers, even in remote areas.
- **Risk Reduction:** Identifies potential yield risks early, helping farmers mitigate losses from extreme weather or poor soil conditions.
- **Sustainability:** Encourages resource-efficient practices, reducing waste and promoting environmentally friendly farming.

5. FEASIBILITY STUDY

5.1 Introduction

An initial investigation culminates in a proposal that determine whether an ultimate system is feasible. When a proposed system is made and approved it initiates a feasibility study. The purpose of the feasibility study is to identify various candidate systems and evaluates they are feasible by considering technical, economical, and operational feasibility and to recommend to best candidate system. The feasibility of such a program is listed in a simulated environment. Once all features are working property in a simulated environment, we can implement in a real platform.

A feasibility study aims to objectively and rationally uncover the strengths and weaknesses of an existing business or proposed venture, opportunities and threats present in the environment, the resources required to carry through, and ultimately the prospect for the successes. In the simplest terms the two criteria to judge feasibility are cost required and value to be attained. During product engineering, we considered following types of feasibility. The study is done in three phases.

- Operational Feasibility
- Economic Feasibility
- Technical Feasibility

5.1.1 Operational Feasibility

Operational feasibility evaluates the practicality of implementing the Crop Yield Prediction System in real-world scenarios, ensuring it aligns with user needs and daily workflows. The system is designed with a user-friendly interface, enabling farmers with varying technical skills to interact with it effortlessly. Features such as simple data input forms, visual dashboards, and clear recommendations ensure ease of use. To make it even more accessible, multi-language support can be incorporated, helping farmers from different regions utilize the system effectively.

The system integrates seamlessly with existing agricultural workflows by leveraging real-time weather APIs and IoT-based soil sensors. This ensures compatibility with devices such as soil monitors and mobile phones, making it easy to adopt without significant infrastructure changes. Administrators can efficiently manage the system, including updating machine learning models or monitoring user activities, without disrupting its operations. This seamless integration and adaptability make the system viable for long-term operational success.

Additionally, the system offers real-time functionality that empowers farmers to make timely decisions. It provides instant predictions and actionable insights, such as optimal irrigation or fertilization schedules, based on live data inputs. Notifications about critical weather changes ensure farmers stay informed and act promptly to mitigate risks. These capabilities enhance the relevance of the system in everyday farming operations, fostering productivity and sustainability.

.Lastly, the system is cost-effective and resource-efficient, making it operationally feasible for a wide range of users. Farmers only need basic internet access and compatible devices like smartphones or computers, reducing the need for expensive equipment. By optimizing resource use, such as water and fertilizers, the system not only lowers operational costs but also promotes sustainable practices. Combined with its scalability and ease of training, the system is well-suited for adoption in agricultural communities, ensuring long-term feasibility.Your paragraph text

5.1.2 Economical Feasibility

Economic feasibility assesses whether the proposed Crop Yield Prediction System is cost-effective and delivers sufficient benefits to justify its implementation. The system is designed to minimize costs while maximizing value for farmers, administrators, and other stakeholders, making it an economically viable solution for agricultural practices.

The initial development and deployment costs are manageable due to the use of open-source technologies such as Django for backend development and widely available machine learning libraries like Scikit-learn or TensorFlow. The integration of external weather APIs and IoT-based soil sensors requires minimal investment, as these technologies are increasingly affordable and readily available. By hosting the system on cloud platforms, the operational costs, including server maintenance and storage, are kept reasonable, ensuring long-term affordability.

For farmers, the system reduces expenses by optimizing the use of resources such as water, fertilizers, and pesticides. Accurate yield predictions and actionable recommendations prevent overuse of inputs, thereby lowering operational costs. The system’s ability to provide timely alerts about weather changes or potential risks helps mitigate crop losses, further improving its economic value. With these benefits, the return on investment (ROI) for farmers is substantial, as it directly contributes to increased productivity and profitability.

Furthermore, the system's scalability ensures that additional users or features can be added with minimal cost increases, making it economically sustainable as adoption grows. Training and support for farmers require limited investment, as the platform is designed to be intuitive and user-friendly. By balancing development and operational costs with tangible benefits, the Crop Yield Prediction System proves to be an economically feasible solution for improving agricultural efficiency and profitability.

5.1.3 Technical Feasibility

Technical feasibility examines whether the technology and resources required for the proposed Crop Yield Prediction System are available, reliable, and sufficient to implement the system effectively. The system utilizes proven technologies such as Django for backend development, machine learning algorithms for predictions, and IoT-based sensors for data collection, ensuring it is built on a robust and scalable technological foundation.

The system's backend, developed using Django, provides a secure, scalable, and efficient framework for managing data and handling requests. It supports integration with external weather APIs and IoT sensors, enabling real-time data collection. The use of RESTful APIs ensures seamless communication between various system components, external applications, and mobile platforms. Furthermore, the system's modular architecture allows for future enhancements, such as the addition of new machine learning models or support for more IoT devices.

Machine learning algorithms, such as Linear Regression and Random Forest, form the core of the prediction engine. These algorithms are widely used in agricultural analytics for their reliability and accuracy in handling large datasets. The system also includes provisions for retraining models with new data, ensuring continuous improvement in prediction accuracy. By leveraging cloud-based resources or local servers for data storage and processing, the system remains efficient even with growing user bases and data volumes

Additionally, the system's compatibility with commonly used devices like smartphones and computers ensures ease of adoption for farmers and administrators. The use of readily available sensors for soil data collection, along with integration capabilities for weather APIs, ensures that the necessary hardware and software components are accessible. Overall, the Crop Yield Prediction System is technically feasible, as it utilizes reliable, scalable, and widely adopted technologies to achieve its goals.

6. SYSTEM DESIGN

6.1 INTRODUCTION

Design is the second phase in the system development life cycle. Software design is the first of the three technical activities in the software development process such as design, code writing and testing. During this phase the analyst schedules design activities works with the user to determine the various data inputs to the system, plans how data flow through the system, design required outputs and write program specifications. Again, the analyst's activities on focus on solving a user's problem in logical terms. During this second step analysts employ a variety of tools such as data flow diagram, entity relationship, data dictionaries and grant chart. The system design converts the theoretical solutions introduced by the feasibility study into a logical reality.

6.2 INPUT DESIGN

Input design is essential for converting user input into a format that computers can understand. It starts with creating a diagram to map out how data flows and where it's stored. Data is organized neatly, often with numbers for clarity. Choosing the right way to input data is crucial for smooth processing. The main goal of input design is to make entering data easy and error-free. This includes checking data carefully to fix any mistakes quickly.

6.3 OUTPUT DESIGN

Output design starts early in a project and is vital as computer output is the primary way users access information. It involves tailoring outputs to meet user requirements, enhancing system-user relations, and aiding decision-making. Standard screen layouts are utilized, meticulously crafted to ensure simplicity, clarity, and relevance to users.

When designing system outputs, analysts face multiple decisions. Regardless of content, all reports should adhere to these guidelines:

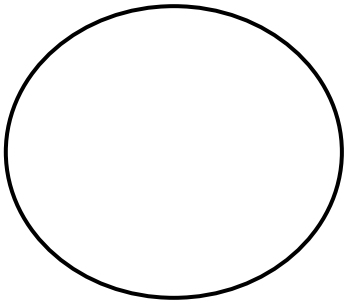
- Information must be clear, accurate, concise, and relevant.
- Reports should feature titles, descriptive headings, and numbered pages
- Content should be logically arranged for easy user navigation.

6.4 DATA FLOW DIAGRAM (DFD)

The Data Flow Diagram (DFD) is an indispensable tool in a system analyst's toolkit. It acts as a panoramic view, revealing the intricate pathways of data processing, transformations, and destinations. Picture it as a detailed roadmap illustrating how information navigates through the system's terrain. This visual aid serves as a universal language, bridging the communication gap between users and analysts.

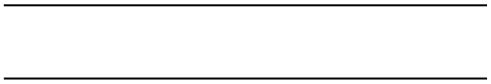
In simpler terms, a DFD is like a flowchart spotlighting the journey of data from start to finish. It's a handy guide that captures the expanding complexity of data flow and system functionalities. At its core, a level zero DFD, also known as the fundamental system model, simplifies the system's anatomy into a single bubble, with incoming arrows representing inputs and outgoing arrows signifying outputs.

DATA FLOW NOTATIONS



- Process

- Entity



- Datastores

- Datastores

1.Process

In a Data Flow Diagram (DFD), a process represents an activity or function that transforms input data into output data. Each process is depicted using a circle (in Yourdon-Coad notation) or a rectangle with rounded corners (in Gane-Sarson notation). Processes are the core components of a DFD, as they illustrate how data is manipulated within the system. They must have at least one input and one output, ensuring that data flows through the system in a meaningful way. For example, a process might take customer order details as input, validate the information, and then produce a confirmed order as output.

Processes are interconnected with data flows, data stores, and external entities. Data flows, represented by arrows, show the movement of data between processes, data stores, and external entities. Data stores, depicted as parallel lines or open-ended rectangles, represent places where data is stored within the system. External entities, shown as rectangles, are sources or destinations of data outside the system. By mapping out these interactions, DFDs provide a clear and structured visualization of how data is processed and transformed within a system, making it easier to understand and analyze the system's functionality

2.External Entities

External entities serve as the gateways for data entering or leaving the system. They are depicted as rectangles on the data flow diagram. These entities can appear multiple times on the diagram to prevent lines from crossing, ensuring clarity in illustrating the flow of data between the system and its external sources or recipients.

3.Data Stores

Data stores represents stores of data within the system within the system. Datastores may be long term files such as ledgers or may be short term accumulation. Each data stores should be given a reference followed by an arbitrary number.

4.Data Flow

A data flow in a data flow diagram depicts the movement of information from its origin to its destination. It is illustrated by a line with arrowheads indicating the direction of flow. Each data flow can be identified by referring to the processes or data stores from which it originates and where it terminates, or by providing a descriptive label. This labeling method ensures clarity in understanding the path and nature of the information being conveyed within the system.

DATAFLOW DIAGRAM

LEVEL 0/CONTEXT LEVEL

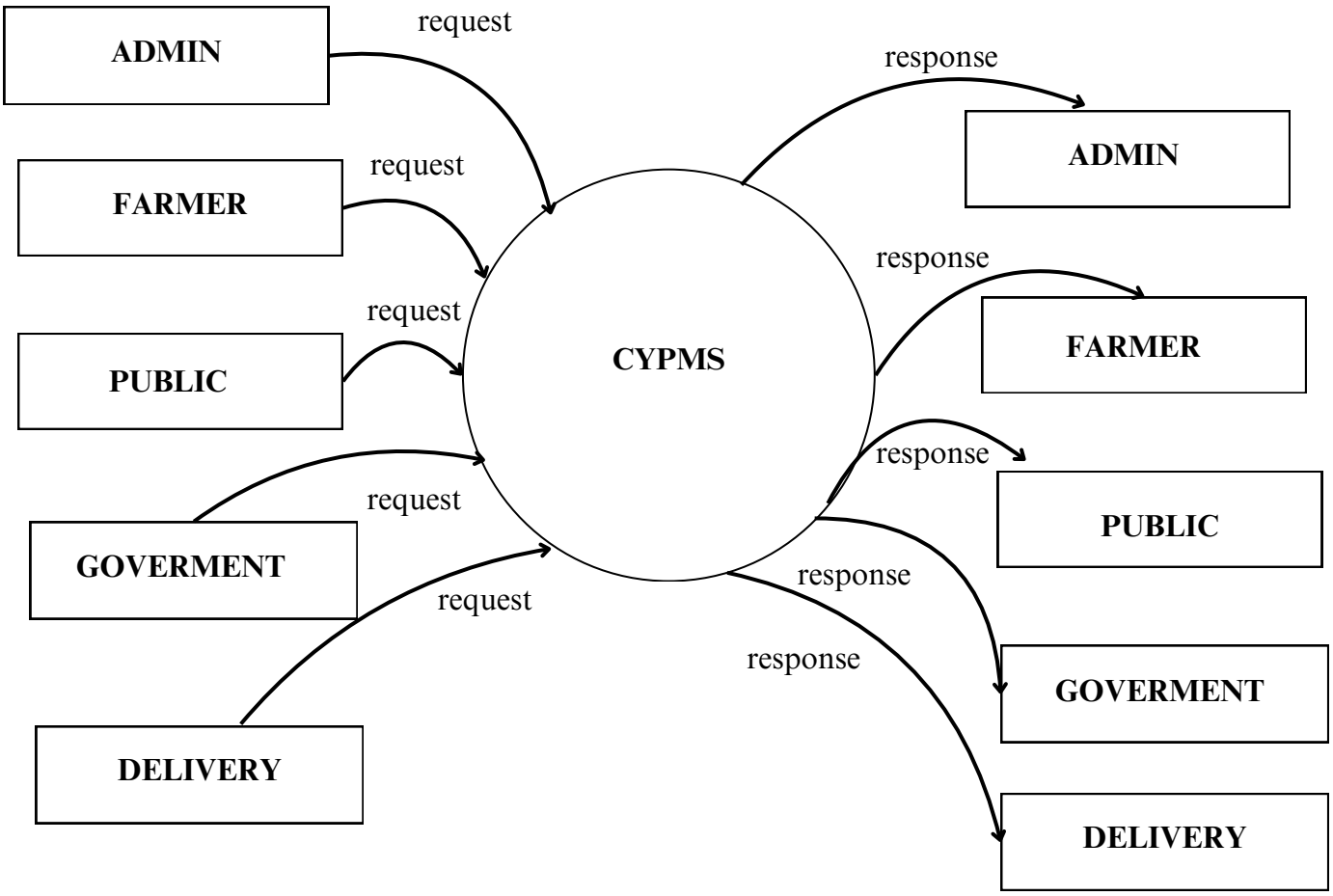


fig 6.1 level 0

LEVEL 1 [ADMIN]

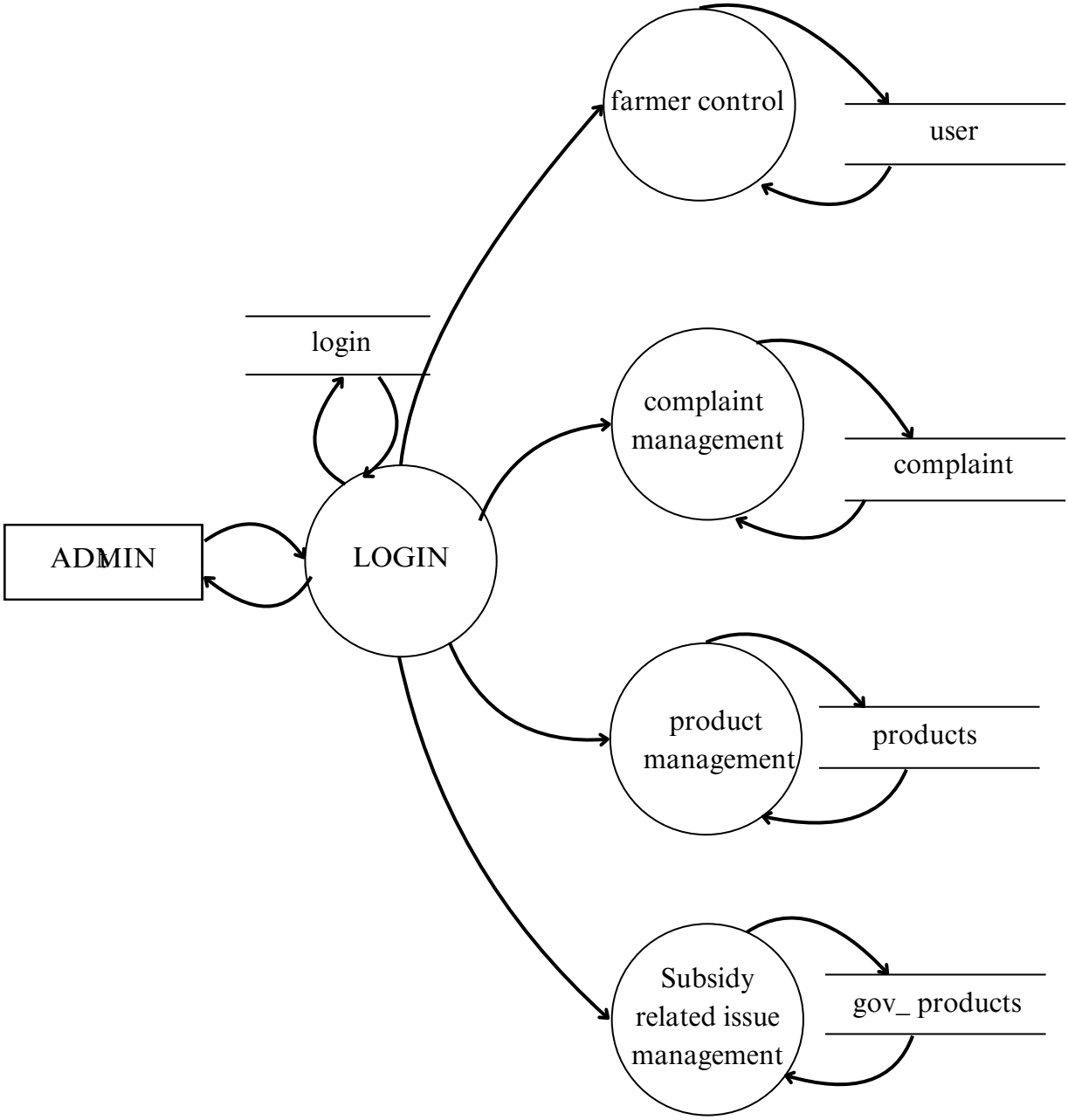


fig 6.2 level 1: admin

LEVEL 1 [FARMER]

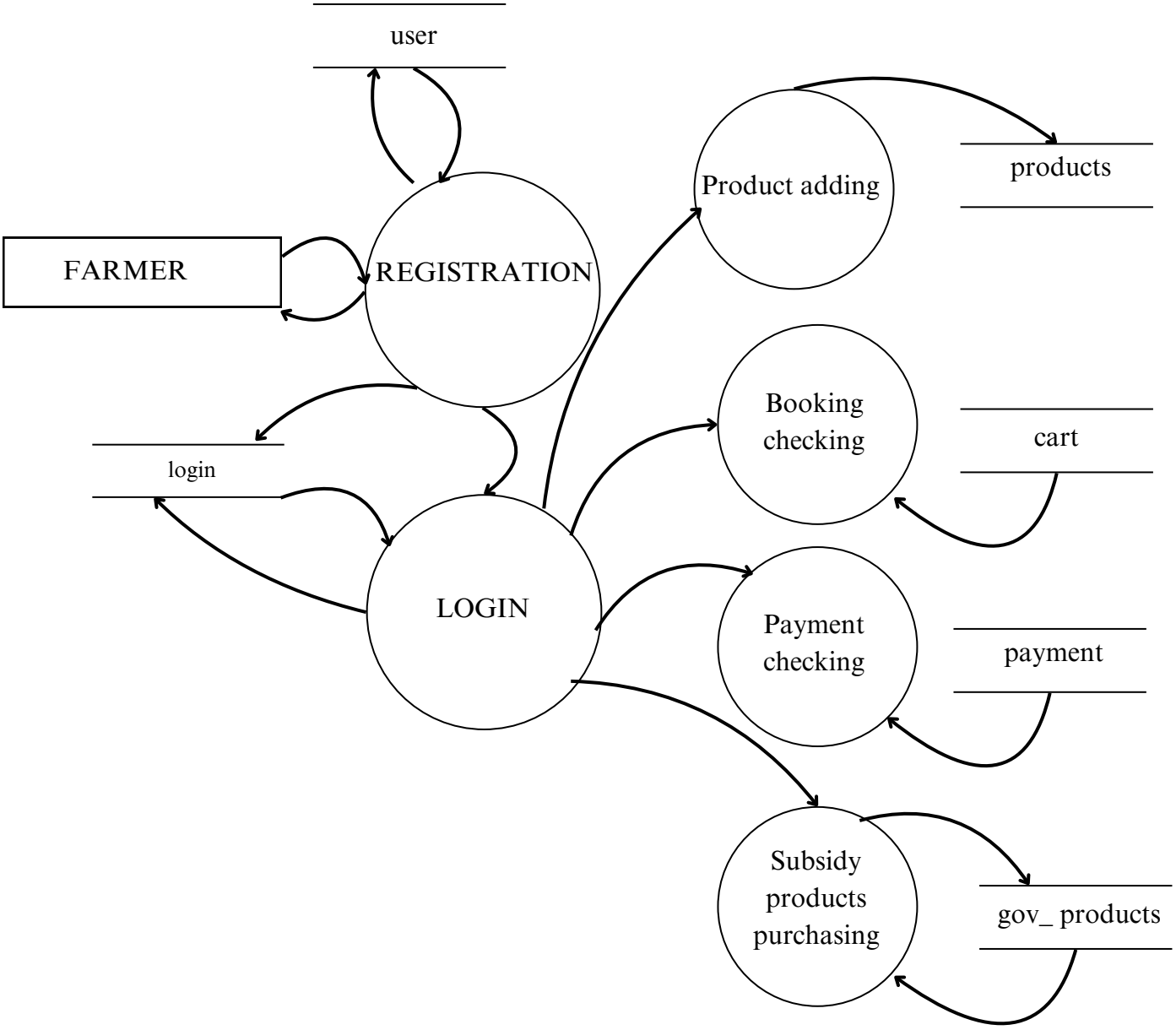


fig 6.3 level 1:farmer

LEVEL 1 [PUBLIC]

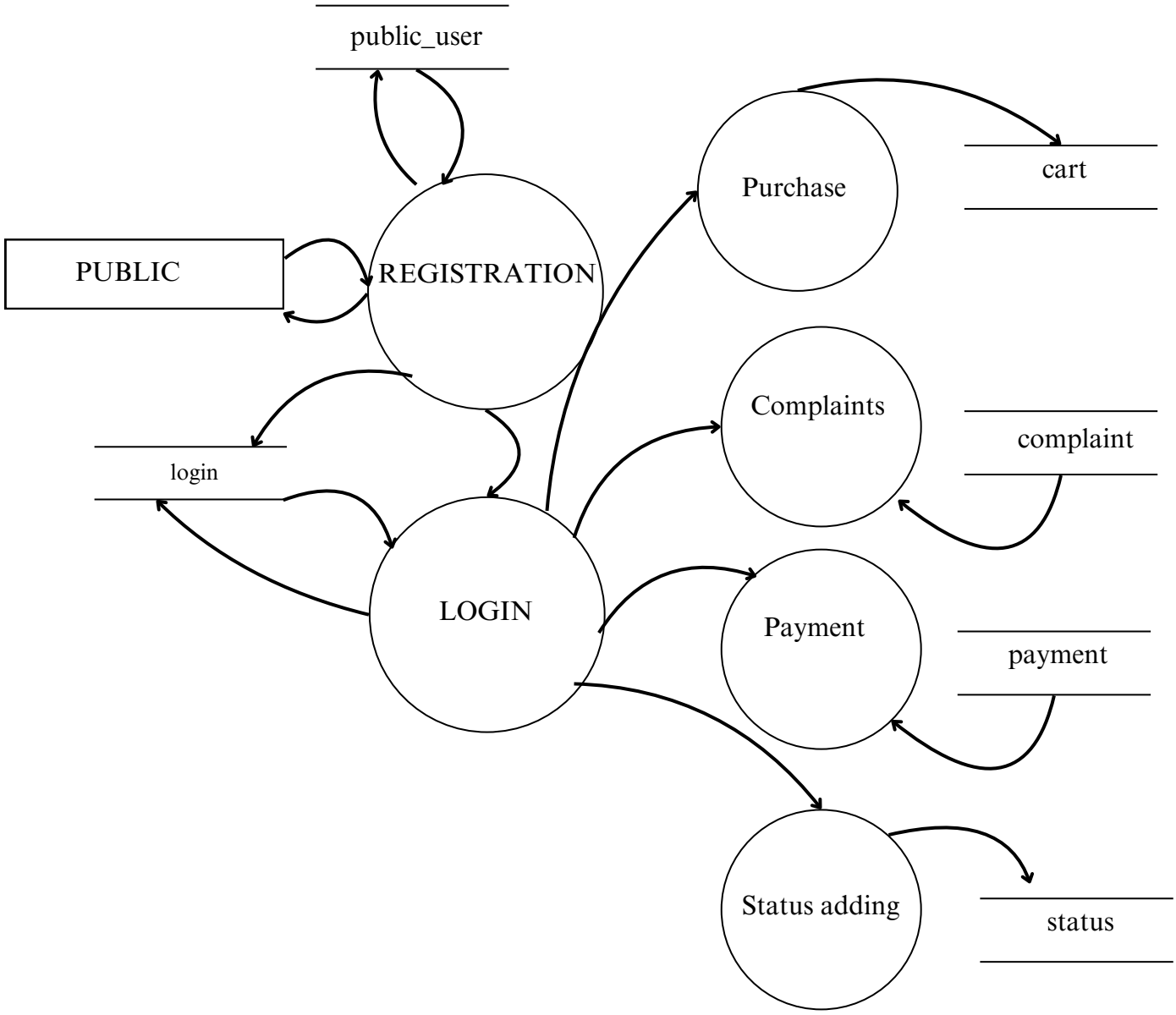


fig 6.4 level 1:public

LEVEL 1 [GOVERMENT]

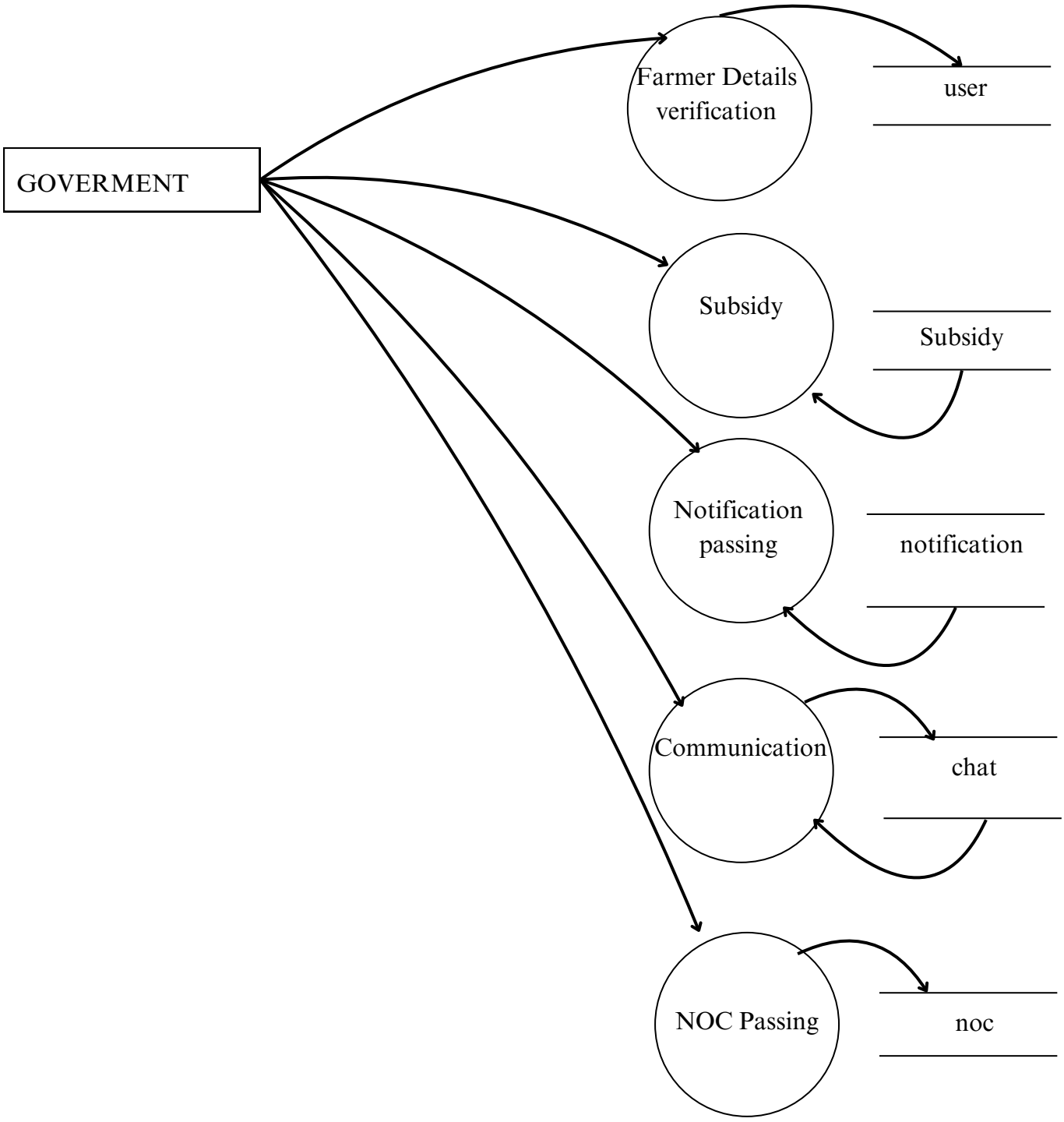


fig 6.5 level 1:government

LEVEL 1 [DELIVERY]

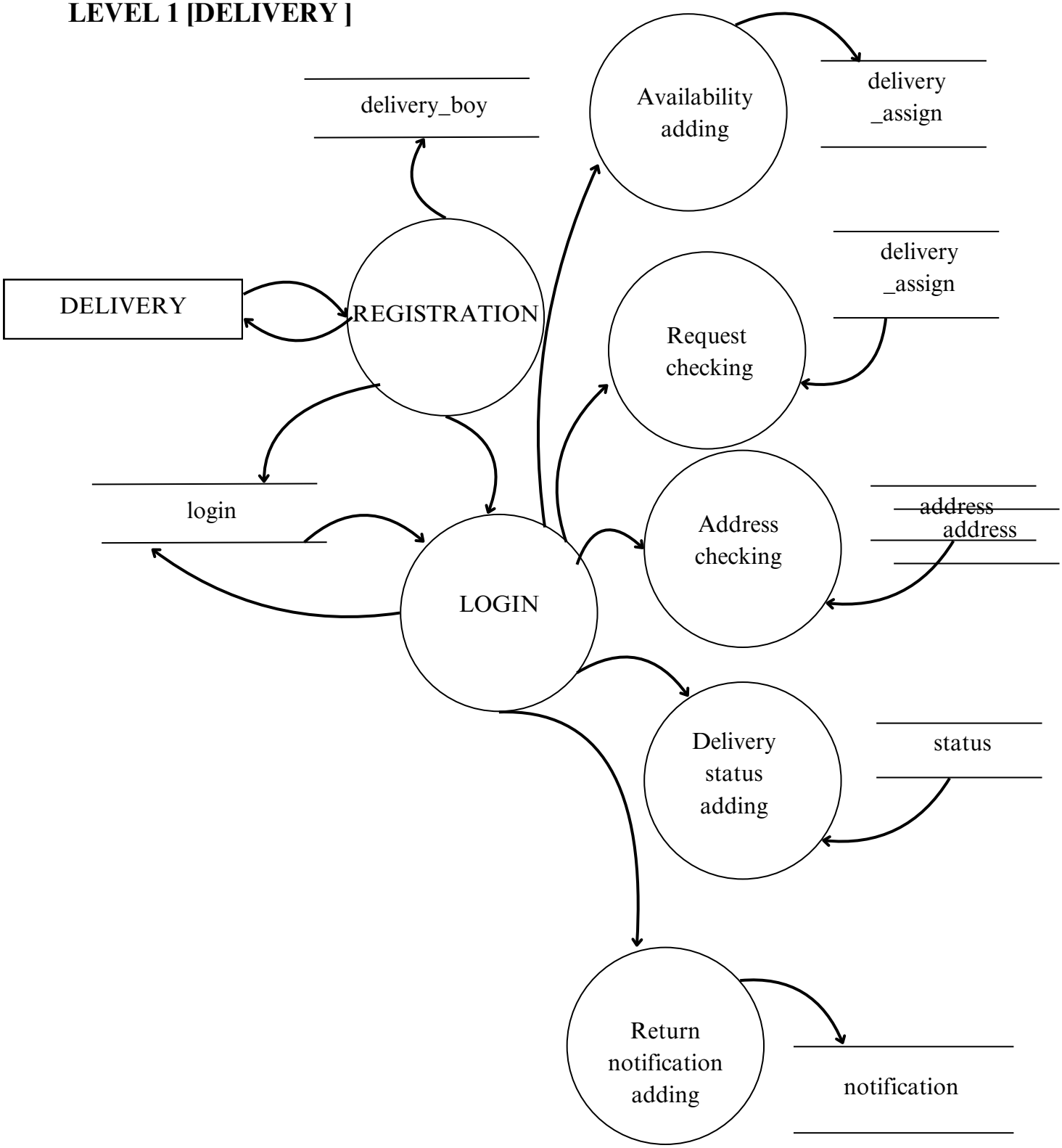


fig 6.6 level 1:delivery

STRUCTURE CHART

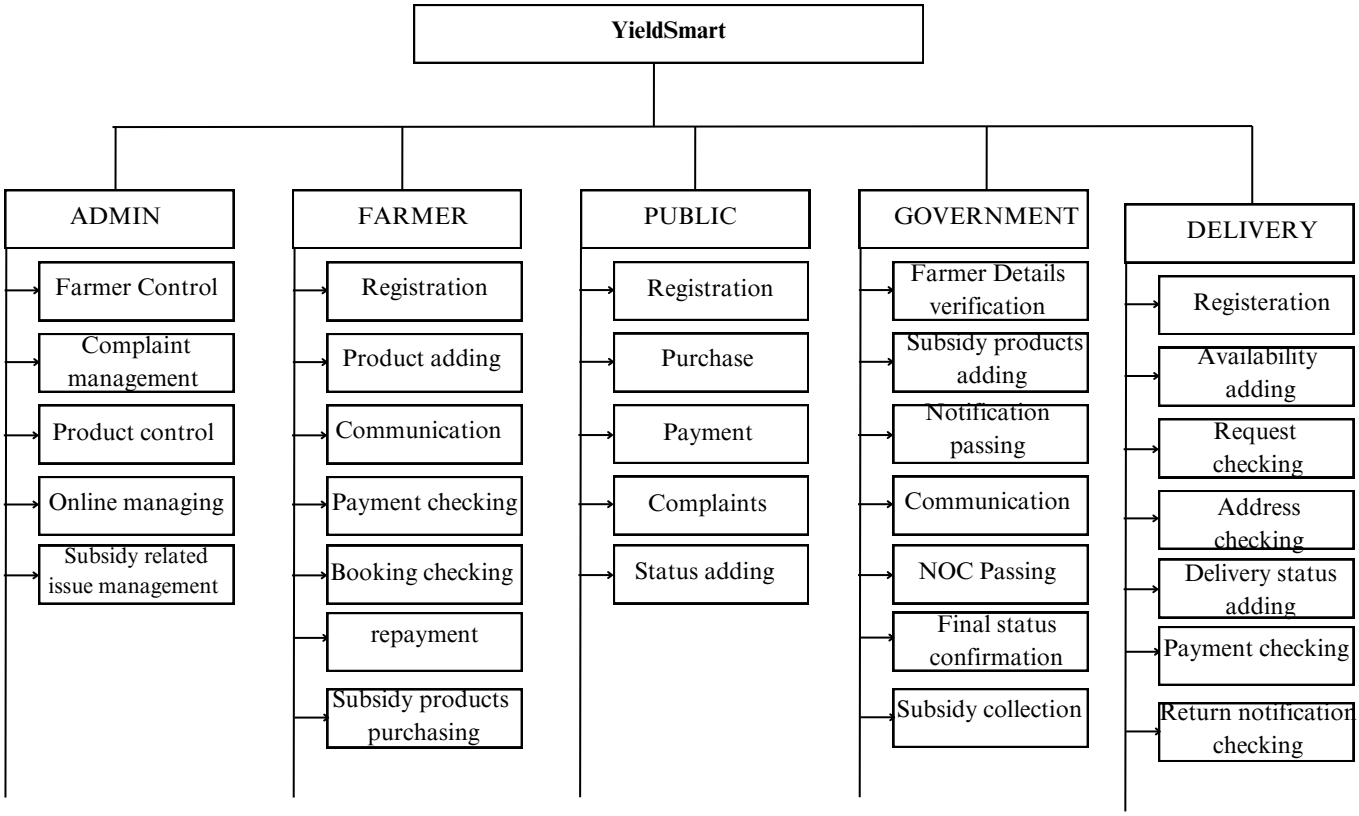
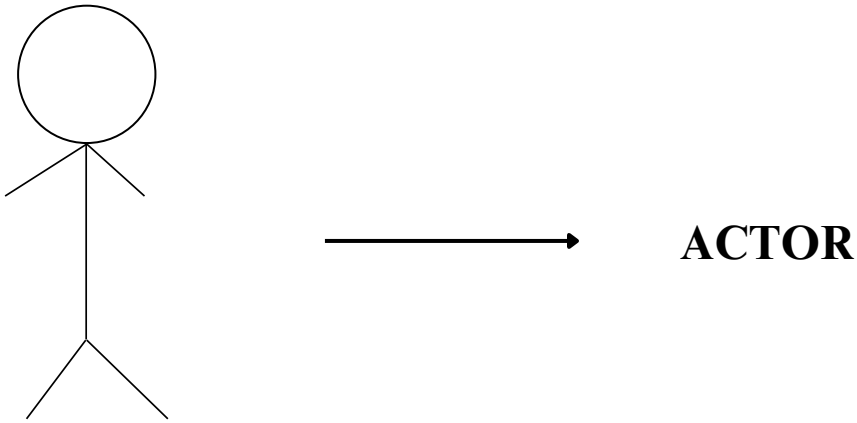
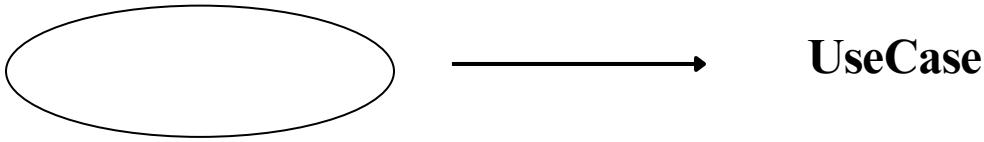


fig 6.7 structure chart

6.5 USE CASE DIAGRAM

A use case is a depiction of a system's behavior from the user's perspective, offering invaluable insights for system developers. It consists of a graphical representation featuring various actors and a collection of specific use cases enclosed within a system boundary. Each use case presents a series of scenarios illustrating interactions between users and the system. These scenarios outline actions taken by users to accomplish specific tasks. A use case diagram visualizes the relationships between actors (users or external systems) and the different use cases within the system. The key components of a use case diagram are the use cases themselves and the actors involved. Use cases portray the actions users might undertake to achieve certain objectives, providing an external view of the system's functionality. Meanwhile, actors represent the users or external systems interacting with the system being modeled.





USE CASE DIAGRAM

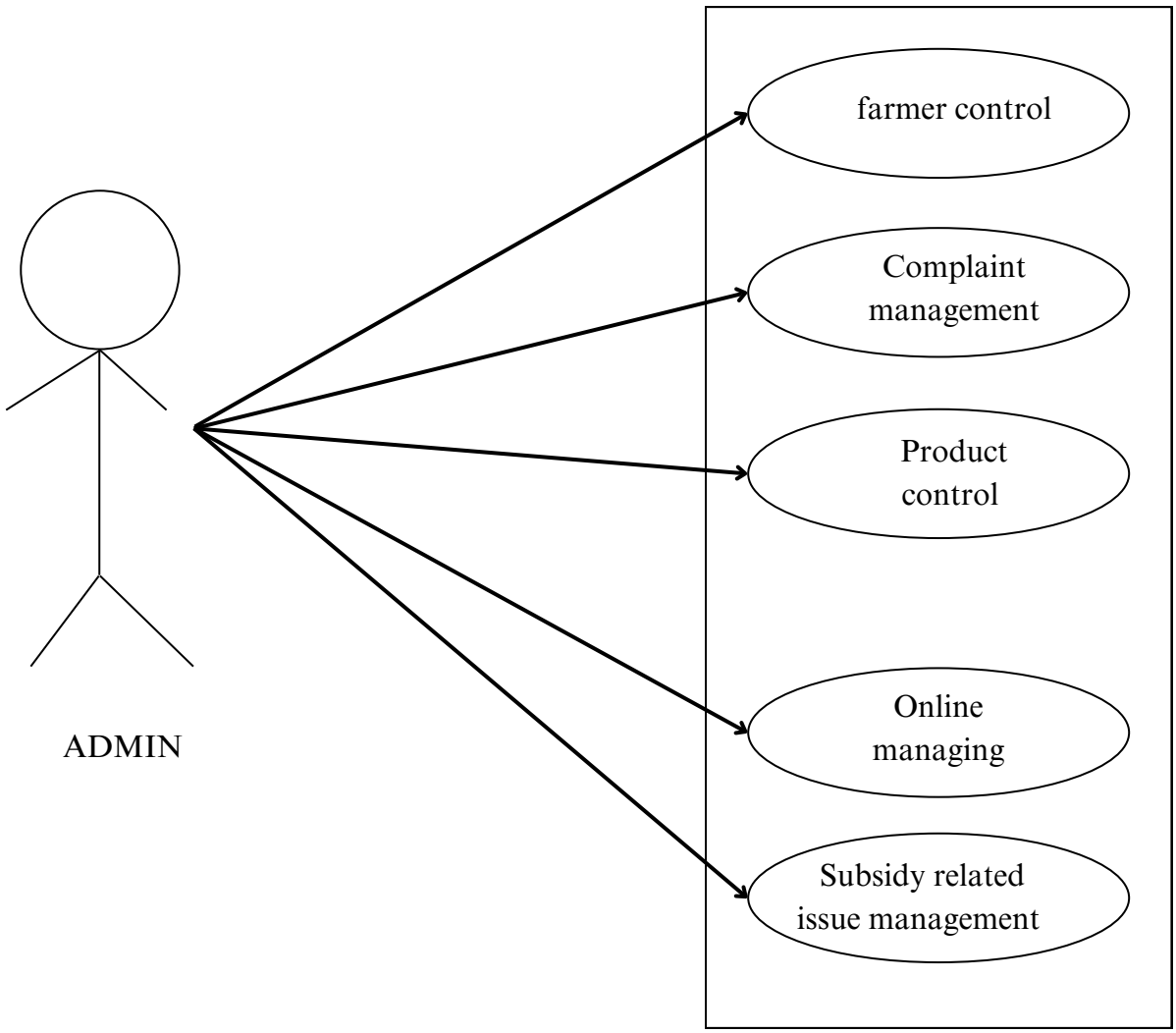


fig 6.8 use case :admin

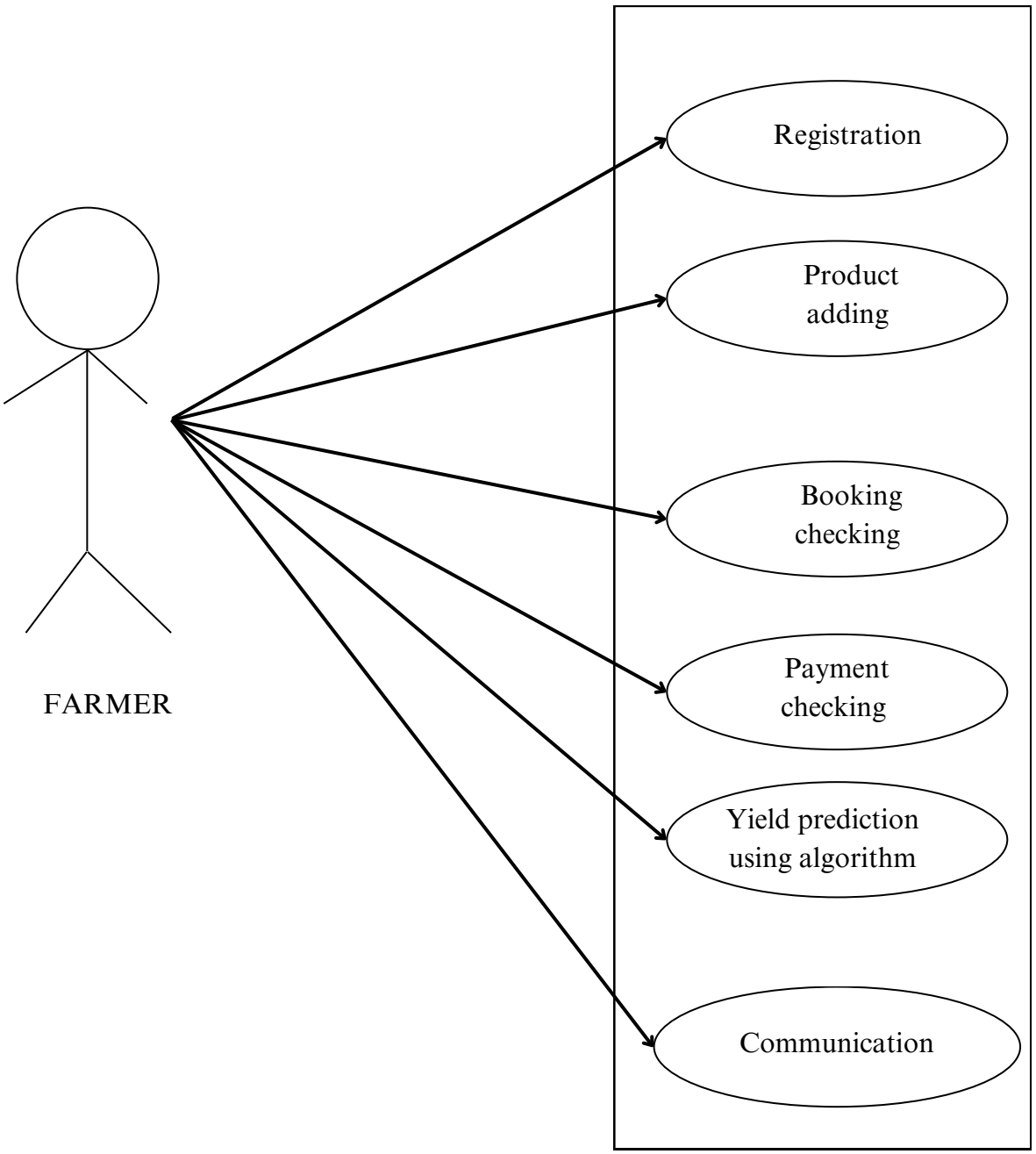


fig 6.9 use case:farmer

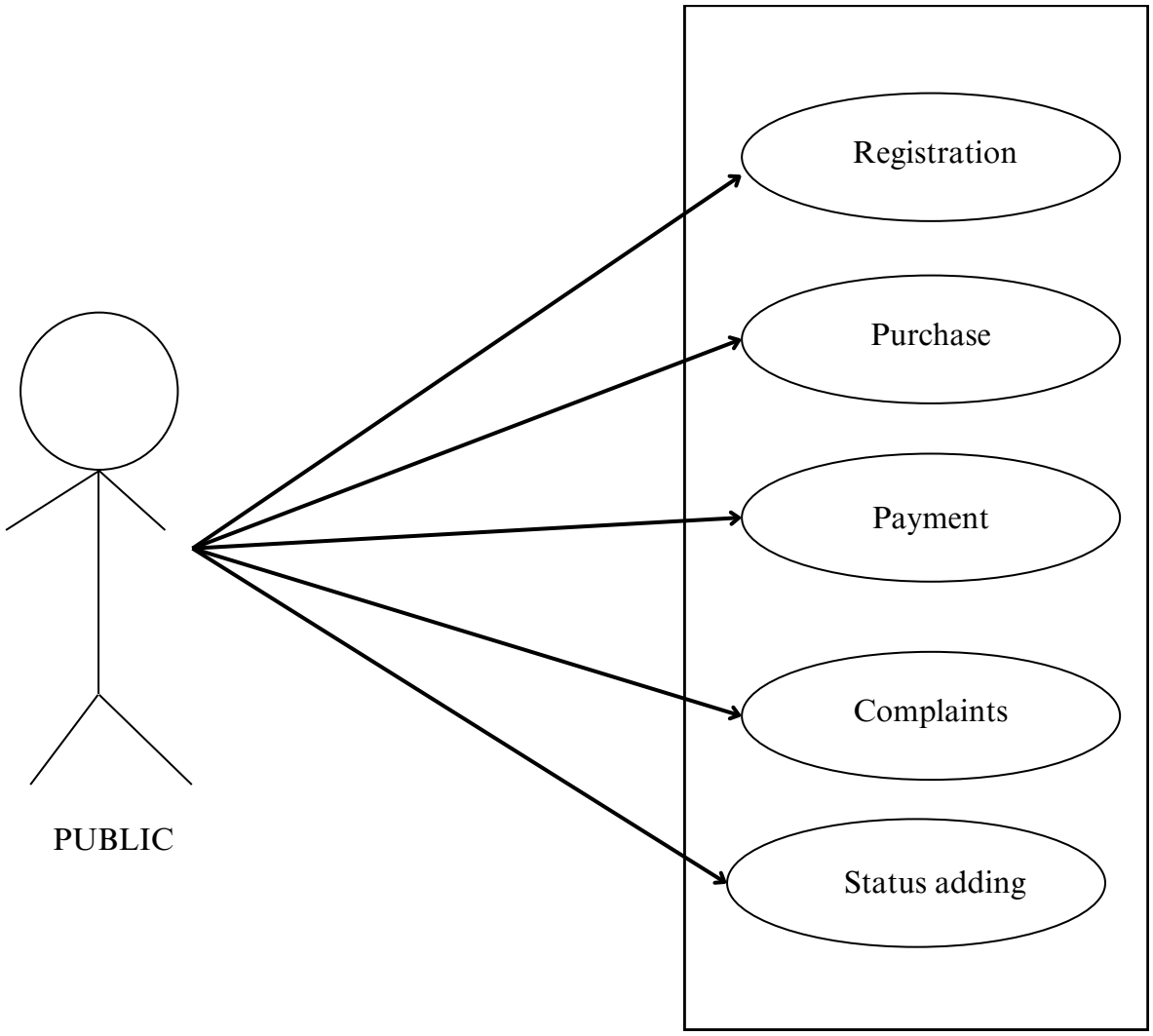


fig 6.10 use case:public

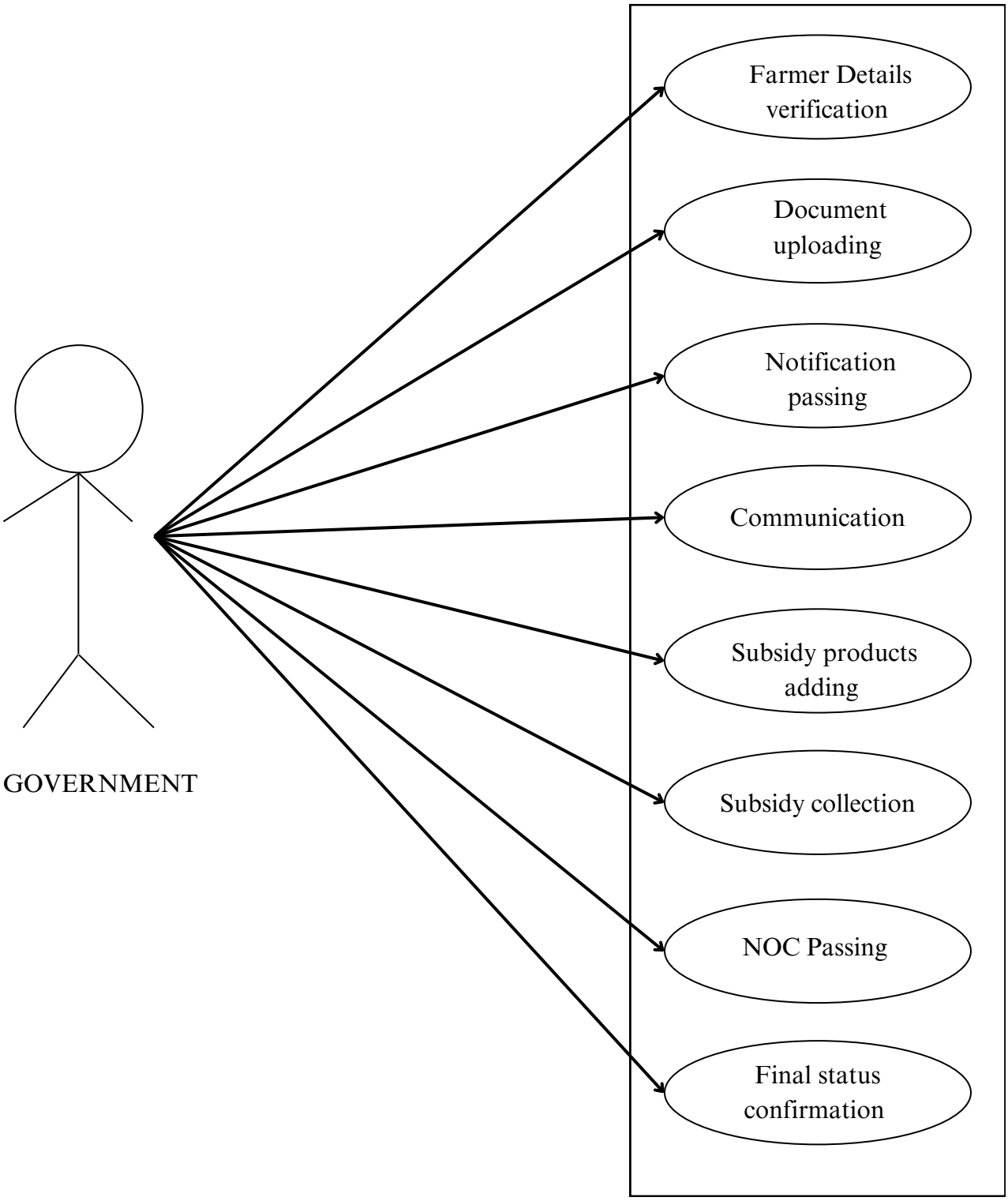


fig 6.11 use case:government

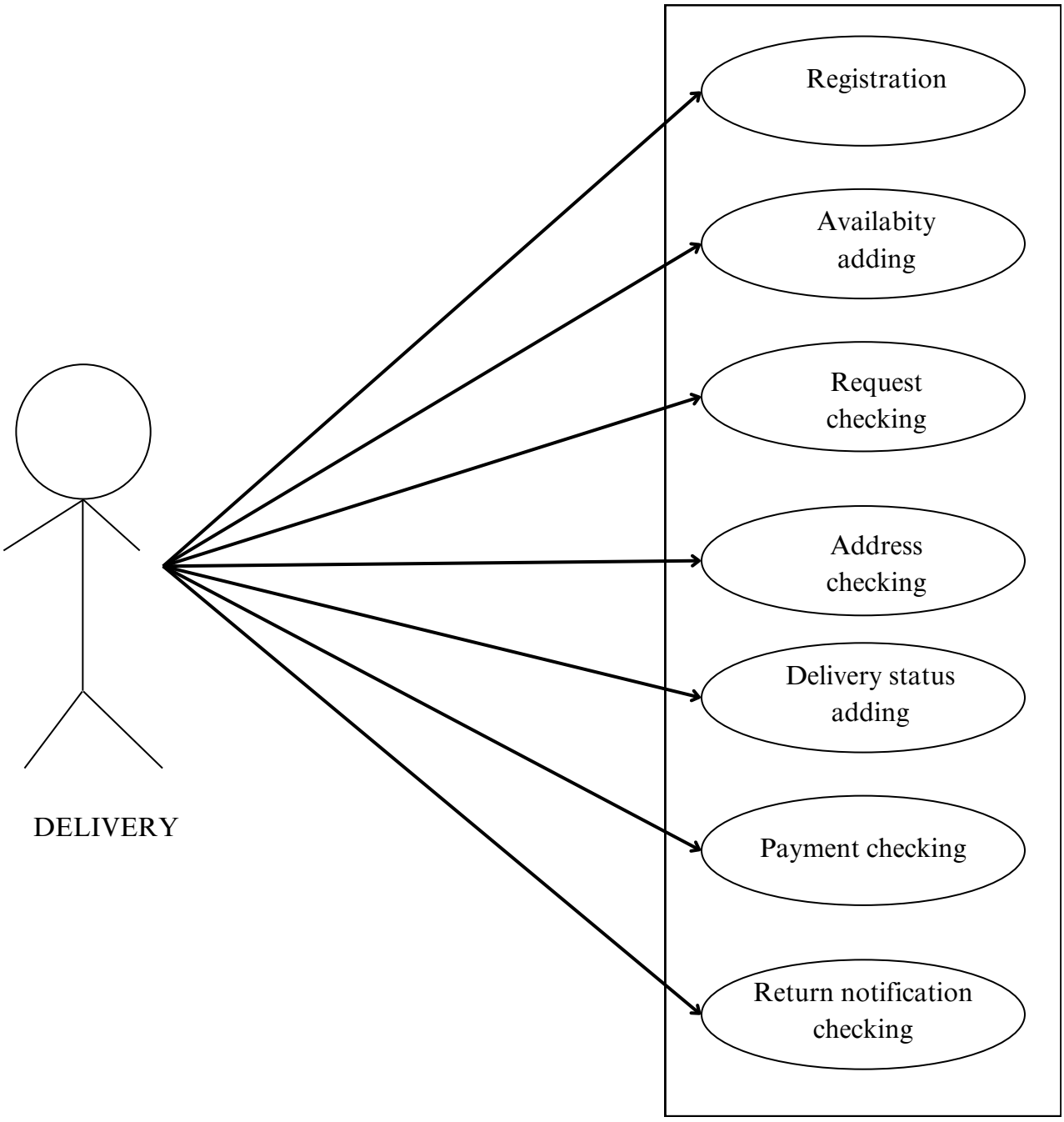


fig 6.12 use case:delivery

6.6 ENTITY RELATIONSHIP DIAGRAM

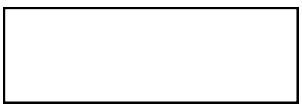
An Entity Relationship (ER) Diagram is a specialized graphic tool used to illustrate the interconnections between entities in a database. Typically, boxes are employed to represent entities, diamonds signify relationships, and ovals denote attributes. It serves as a visual representation of how entities relate to one another within a database structure. An entity refers to a specific piece of data, object, or concept stored in the database, while a relationship signifies how data is shared between these entities. The degree of a relationship indicates the number of entities involved in the relationship, with the general form for a degree of 'n' being an n-ary relationship.

Binary relationships, which involve two entities, are the most common type in practical applications. However, in cases where a binary relationship is insufficient, a ternary relationship involving three entities may be necessary.

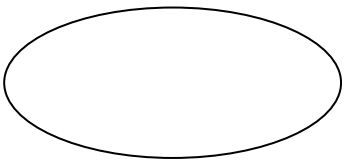
The connectivity of a relationship describes how instances of associated entities are mapped within the relationship. Cardinality, on the other hand, refers to the actual number of related occurrences for each entity involved in the relationship. The primary types of connectivity in relationships include one-to-one, one-to-many, and many-to-many.

In a one-to-one relationship, at most one instance of entity A is associated with one instance of entity B. In a one-to-many relationship, one instance of entity A can be related to one or multiple instances of entity B. Conversely, in a many-to-many relationship, one instance of entity A can be associated with many instances of entity B, and vice versa. This type of relationship, sometimes referred to as non-specific, represents a more complex data sharing scenario.

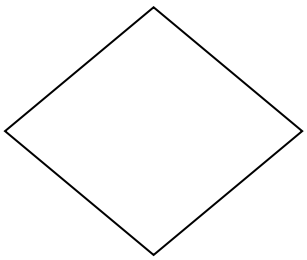
Symbols used in ER Diagram



- Entity



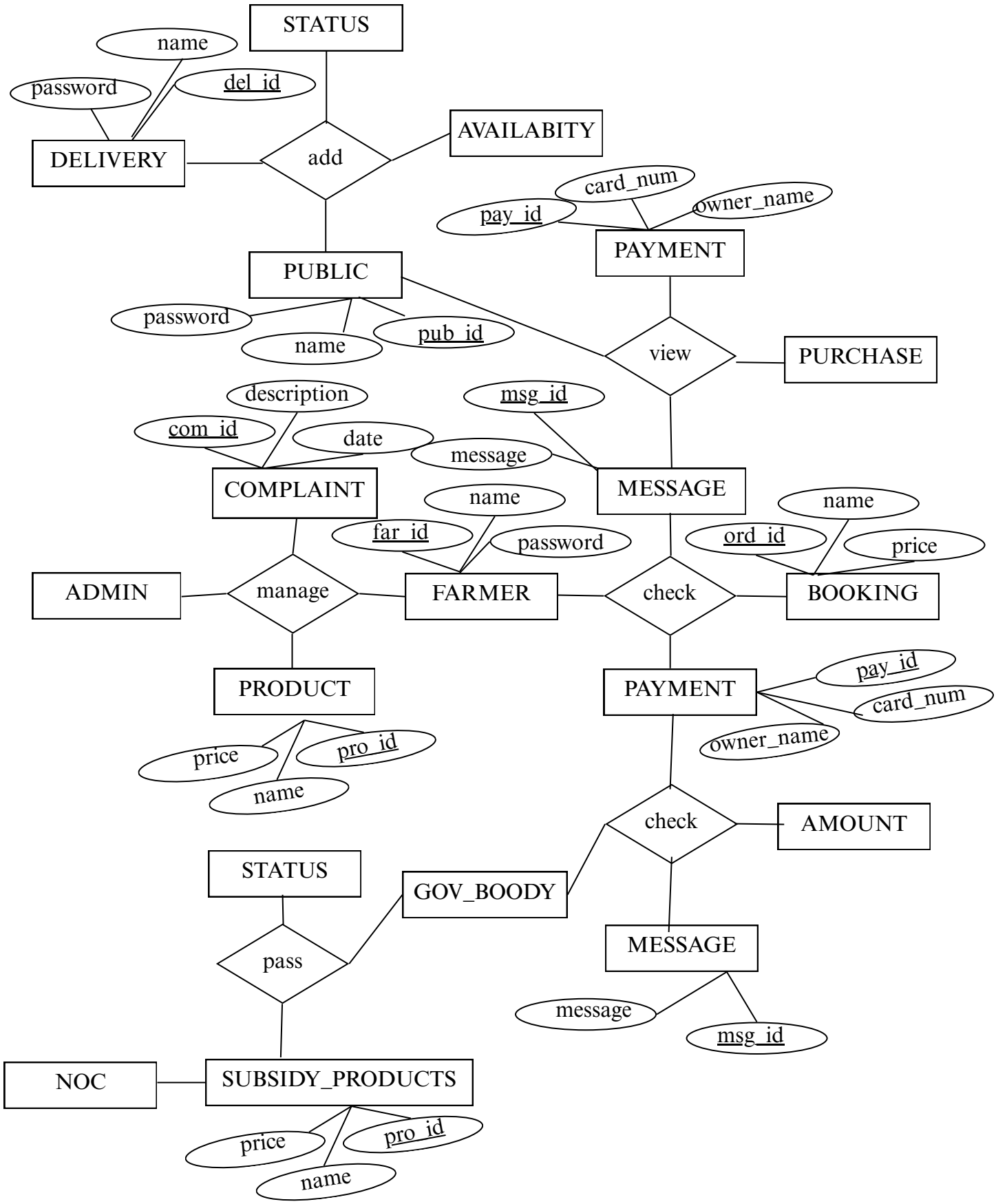
- Attribute



- Relationship

- Connecting lines

ER DIAGRAM


STATUS
name
password
del_id
DELIVERY
add
AVAILABITY
card_num
pay_id
owner_name
PAYMENT
PUBLIC
password
name
pub_id
view
PURCHASE
description
com_id
date
msg_id
message
COMPLAINT
MESSAGE
name
far_id
password
name
ord_id
price
ADMIN
manage
FARMER
check
BOOKING
PRODUCT
PAYMENT
pay_id
card_num
owner_name
price
pro_id
name
check
AMOUNT
STATUS
GOV_BOODY
pass
MESSAGE
message
msg_id
NOC
SUBSIDY_PRODUCTS
price
pro_id
name

6.7 DATABASE DESIGN

Database design is the backbone of modern information systems, prevalent across virtually all computer setups. At its core, it aims to consolidate and organize data effectively. Imagine it as a meticulously arranged library, where information is neatly categorized for easy access by all users.

To ensure an efficient database, it must fulfill several key criteria:

- Controlled redundancy: Avoid unnecessary duplication of data.
- User-friendly: Ensure simplicity and accessibility for all users.
- Data integrity: Maintain accuracy and reliability while allowing for flexibility.
- Privacy and security: Safeguard sensitive data from unauthorized access.
- Disaster recovery: Implement mechanisms to swiftly recover from system failures.
- Performance optimization: Ensure efficient data processing and retrieval.

The essence of a database lies in its simplicity and accessibility, akin to a well-structured library where information is readily available to all who need it.

Creating a database involves a step-by-step process:

- Identify the categories of data and how they interrelate.
- Determine the specific details to be stored within each category.
- Resolve any complex relationships between different data categories.
- Thoroughly validate the design to ensure coherence and effectiveness.
- Execute the finalized plan to construct the database system.

KEYS

A keys are essential for uniquely identifying records (rows) and establishing relationships between different tables. Various types of keys are :

Primary Key: Primary key is the column used to uniquely identify a particular row in a table. Every database table should have one or more columns designated as the primary key. The value this key holds should be unique for each record in the database.

Candidate Key: Candidate key is the combination to one or more columns, the values of which uniquely identify each row of a table.

Foreign Key: A candidate key is one or more columns whose values are based on the primary or candidate key of another table. These keys are used to create relationships between tables. Natural relationships exist between tables in most database structures.t

Super Key: A super key is a combination of attributes that can be uniquely used to identify record. A table might have super keys. Candidate keys are a special subset of super keys that do not have any extraneous information on it. Here the data are stored in tables and each module uses separate tables allotted for each.

DATABASE TABLES

1.login

Primary Key:id

Description: To store the login details of users

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
email	email	default	NOT NULL	email for login
password	int	8	NOT NULL	password for login
user_type	char	-	NOT NULL	type of the user

2.user

Primary Key: id

Description: To store the details of farmers

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
name	char	35	NOT NULL	name of the farmer
contact	int	10	NOT NULL	contact number of farmer
login_id	int	-	Foreign key	foreign key from login
adress	char	200	NOT NULL	address of the farmer
district	char	50	NOT NULL	district of the farmer
main_products	char	200	NOT NULL	product which mainly farm
farm_size	int	_	NOT NULL	size of the farm

3.Public_user

Primary Key: id

Description: To store the details of public users

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
name	char	35	NOT NULL	name of the public user
contact	int	10	NOT NULL	contact number of public user
login_id	int	-	Foreign key	foreign key from login

4.Notification

Primary Key: id

Description: To store the details of notification

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
message	char	150	NOT NULL	message to show
created_at	Datetime	-	NOT NULL	date in which message created

5.products

Primary Key: id

Description: To store the details of products

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
category	char	100	NOT NULL	category of products
name	char	50	NOT NULL	name of the products
image	default	-	-	image of the products
price	int	5	NOT NULL	price of the products
login_id	int	-	Foreign key	foreign key from login

6.cart

Primary Key: id

Description: To store the details of cart of users

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
product_id	int	default	Foreign key	foreign key from products
user_id	int	50	Foreign key	foreign key from login
payment _status	int	default	NOT NULL	payment status of purchase
current _date	Datetim e	default	-	current date of purchase
cancelation_st atus	int	default	NOT NULL	staus of cancelation process

7.payment

Primary Key: id

Description: To store the details of payment for the users

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
owner name	char	50	NOT NULL	name of buyer on card
card_no	int	50	NOT NULL	card number of buyer
cvv	int	3	NOT NULL	cvv number of card
exp_month	Date	default	NOT NULL	month of expiry of card
exp_year	Date	default	NOT NULL	year of expiry of card
cart_id	int	default	Foreign key	foreign key from cart
login_id	int	default	Foreign key	foreign key from login
current _date	datetime	default	default	current date of payment

8.alert

Primary Key: id

Description: To store the details of alert to farmers

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
message	text	default	NOT NULL	message shown in alert
created_at	Datetime	default	default	date in which alert updated

9.gov_products
Primary Key: id
Description: To store the details of subsidy product to farmers

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
category	char	100	NOT NULL	category of products
name	char	50	NOT NULL	name of the products
image	default	-	-	image of the products
subsidy_price	int	5	NOT NULL	subsidy price of the products
current_date	Datetime	-	-	date in which product added

10.farmer_cart
Primary Key: id
Description: To store the details of subsidy products in the cart

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
product_id	int	default	Foreign key	foreign key from products
user_id	int	50	Foreign key	foreign key from login
payment _status	int	default	NOT NULL	payment status of purchase
current _date	Datetime	default	-	current date of purchase
cancelation _status	int	default	NOT NULL	status of cancelation process

11.farmer_payment

Primary Key: id

Description: To store the details of payment for subsidy products

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
owner name	char	50	NOT NULL	name of buyer on card
card_no	int	50	NOT NULL	card number of buyer
cvv	int	3	NOT NULL	cvv number of card
exp_month	Date	default	-	month of expiry of card
exp_year	Date	default	-	year of expiry of card
cart_id	int	default	Foreign key	foreign key from cart
login_id	int	default	Foreign key	foreign key from login
current _date	datetime	default	-	current date of payment

12 address

Primary Key: id

Description: To store the details of address for delivery

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
name	char	50	NOT NULL	name of the buyer
contact	int	10	NOT NULL	contact number of the buyer
house_name	char	100	NOT NULL	house name of the buyer
area	char	100	NOT NULL	area of house located
landmark	char	100	NOT NULL	landmark used for delivery
pincode	int	10	NOT NULL	pincode of the user
login_id	int	default	Foreign key	foreign key from login
city	char	120	NOT NULL	city in which user located
state	char	120	NOT NULL	state in which user located

13 delivery_boy

Primary Key: id

Description: To store the details of delivery boys for delivery

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
name	char	150	NOT NULL	name of delivery boy
contact	int	10	NOT NULL	contact number of delivery boy
far_id	int	default	Foreign key	foreign key from login for farmer id
login_id	int	default	Foreign key	foreign key from login for delivery login

14delivery_assign

Primary Key: id

Description: To store the details of assigning delivery boys for delivery

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
created_at	datetime	-	-	date of assigning the delivery
cart_id	int	10	Foreign key	foreign key from cart for cart id
delivery_team _id	int	default	Foreign key	foreign key from delivey_boys for team id
status	int	default	NOT NULL	To store status of delivery

14 chat

Primary Key: id

Description: To store the details of chat with farmer and public

Field Name	Datatype	Size	Constraint	Description
id	int	default	Primary key	Unique identifier
farmer_sender_id	int	default	Foreign key	foreign key from user for farmer sender id
farmer_reciever_id	int	default	Foreign key	foreign key for user for farmer reciever id
public_sender_id	int	default	Foreign key	foreign key from public_user for public sender id
Public_reciever_id	int	default	Foreign key	foreign key from public_user for public reciever_id

6.8 NORMALIZATION

To ensure efficiency and integrity, databases often undergo normalization—a process of structuring data to reduce redundancies. This technique optimizes database design by adhering to guidelines that eliminate duplicate information. Normalization enhances storage efficiency, data integrity, and scalability while reducing anomalies like data repetition or loss.

A poorly designed database can lead to various issues, including redundant information, inability to represent certain data, or loss of crucial information. Normalization addresses these concerns by restructuring data, making it easier to maintain and ensuring that related activities within an organization are streamlined.

Benefits of normalization include:

- Minimization of duplicated data, leading to efficient storage usage.
- Flexibility to support different functional requirements.
- Ease of translating the model into a database design

By achieving a well-normalized database design, organizations can enhance database efficiency, streamline operations, and maintain data integrity effectively

1. First Normal Form (1NF)

- A relation is in 1NF if all its attributes are based on a single domain, ensuring that each entry in the table contains a single value.
- The objective is to eliminate repeating groups and ensure atomicity of data

2. Second Normal Form (2NF):

- A table is in 2NF if it's in 1NF and every attribute is functionally dependent on the entire primary key, rather than just a part of it.
- This eliminates partial dependencies within the table

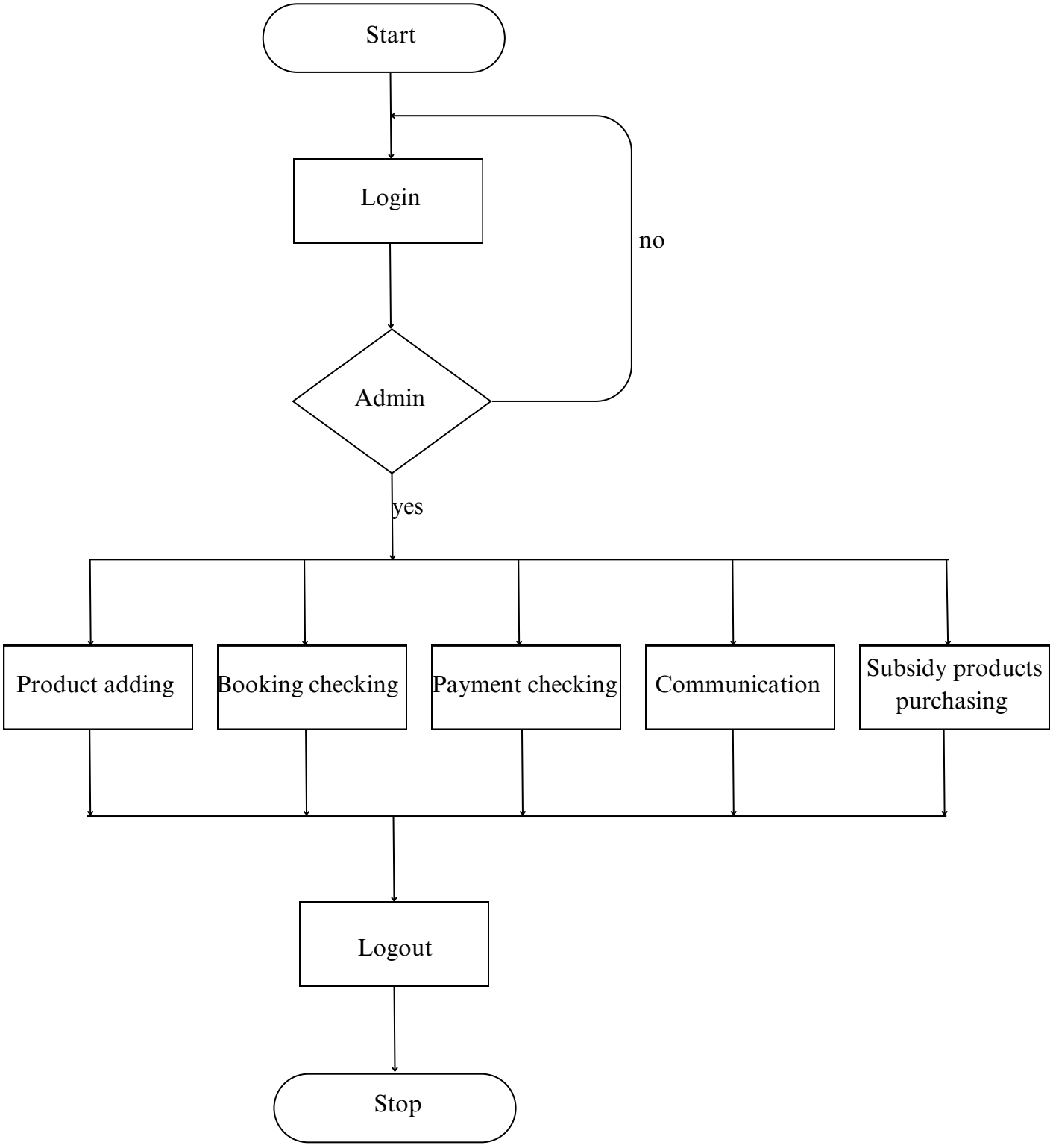
3. Third Normal Form (3NF):

- A table is in 3NF if it's in 1NF and no non-prime attribute is transitively dependent on the primary key.
- This eliminates transitive dependencies, further improving data integrity.

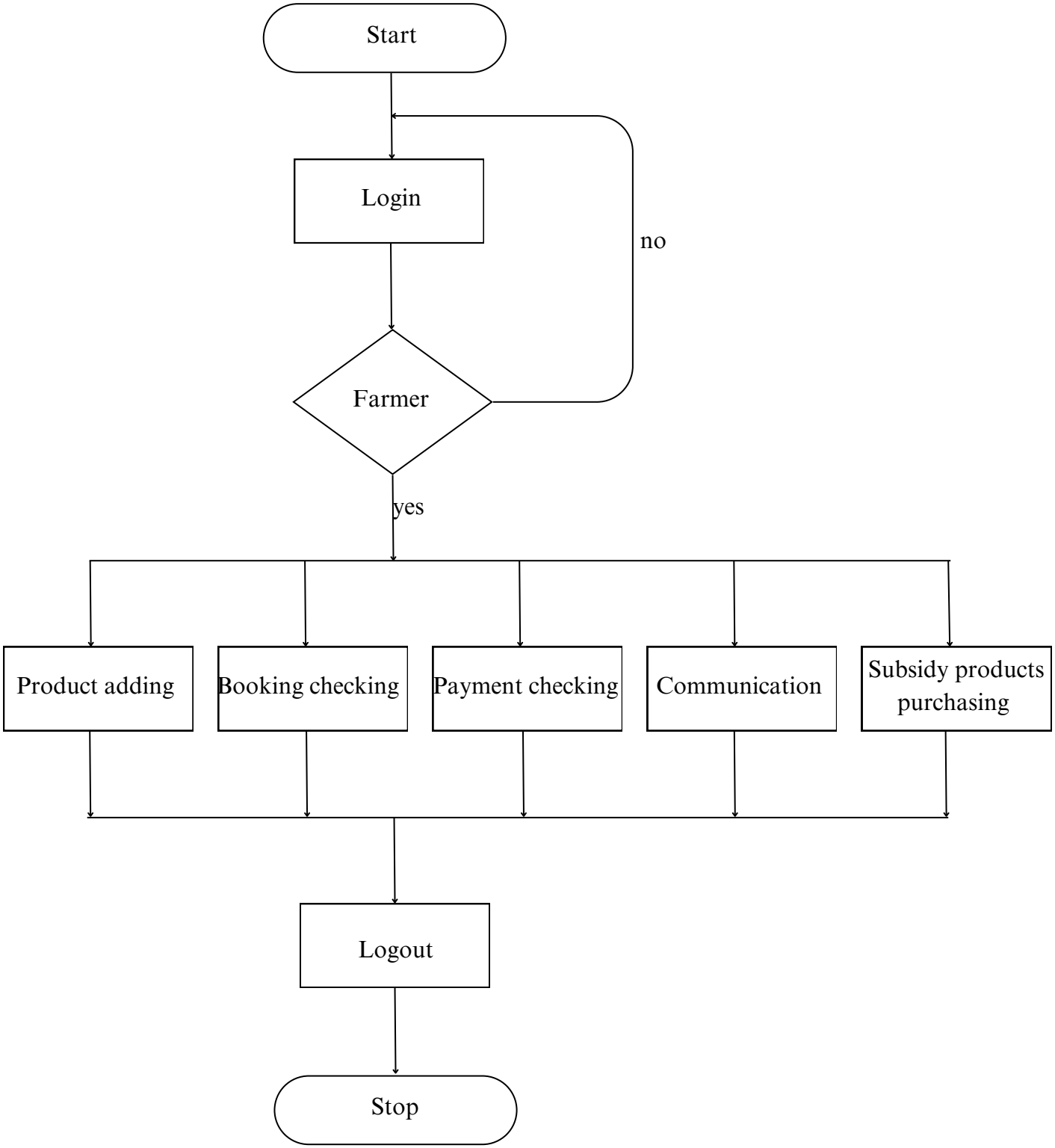
The database used in the system, named "campus," comprises five tables. These tables adhere to the principles of normalization, ensuring atomic values, absence of partial and non-key dependencies.

- Benefits of normalization include:
- Simplifies table structure, making it easier to manage.
- Eliminates data redundancy, saving storage space.
- Enhances overall database organization.
- Maintains data consistency within the database.
- Allows for a more flexible database design, accommodating changes efficiently

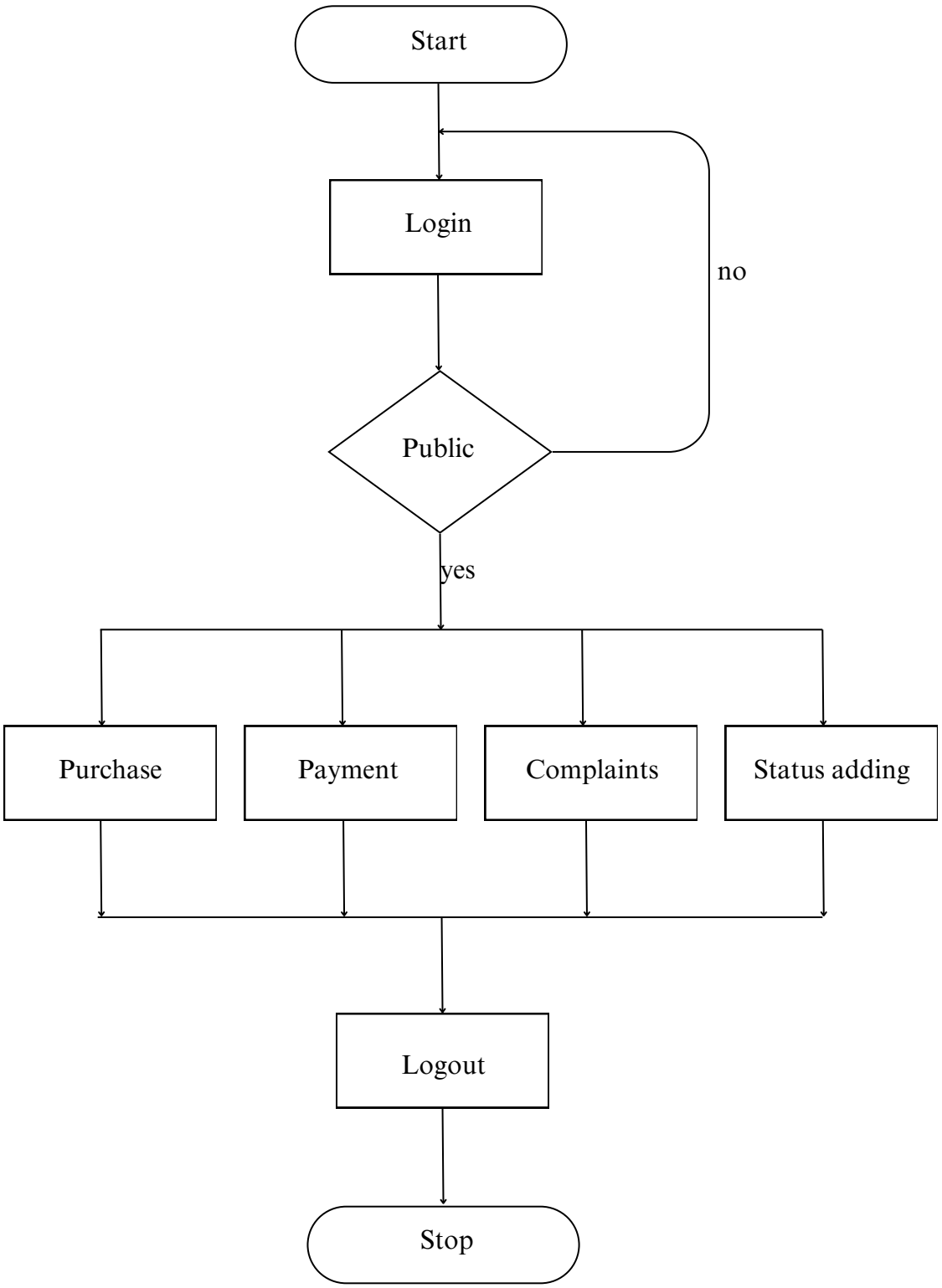
FLOWCHART FOR ADMIN



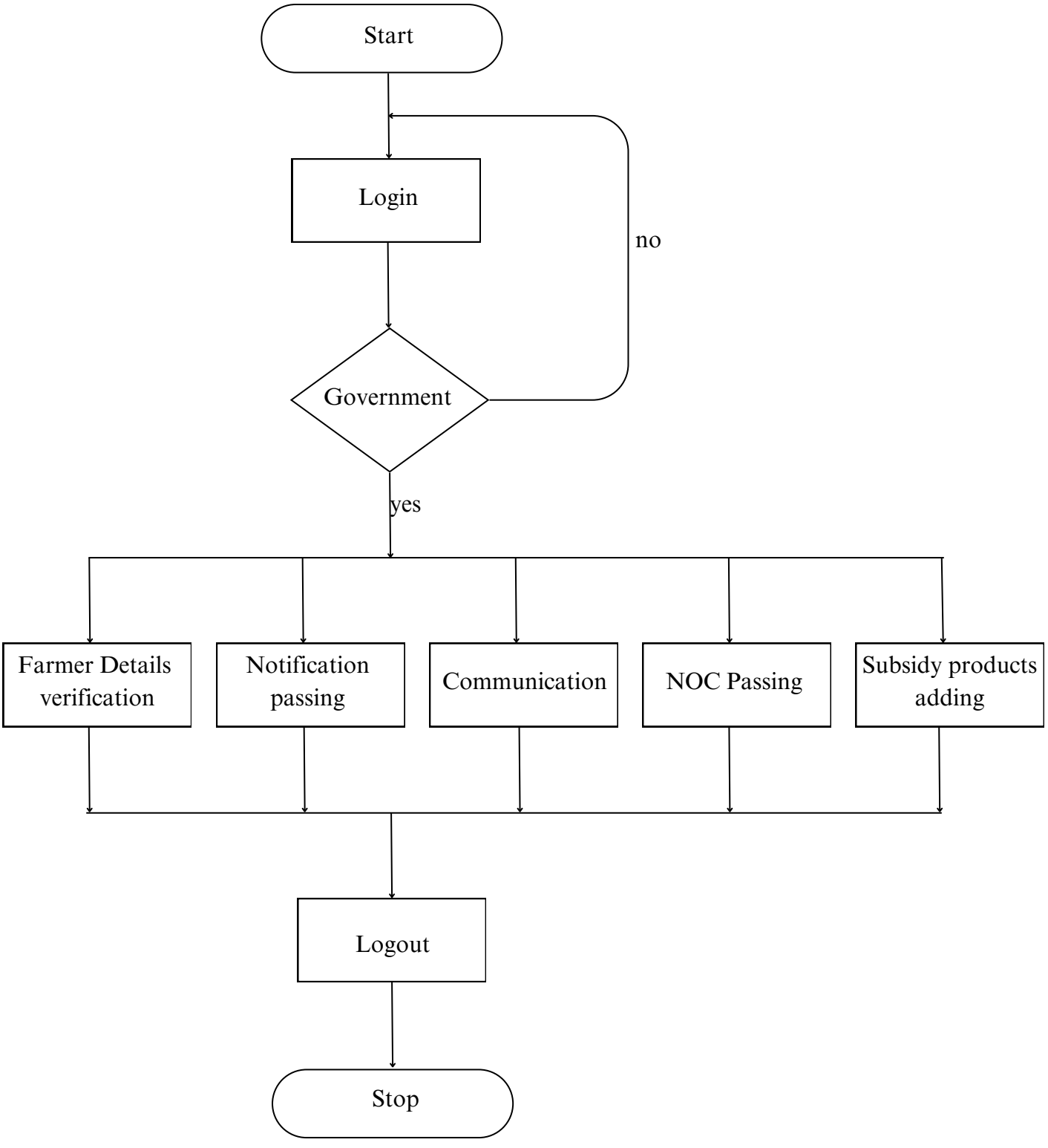
FLOWCHART FOR FARMER



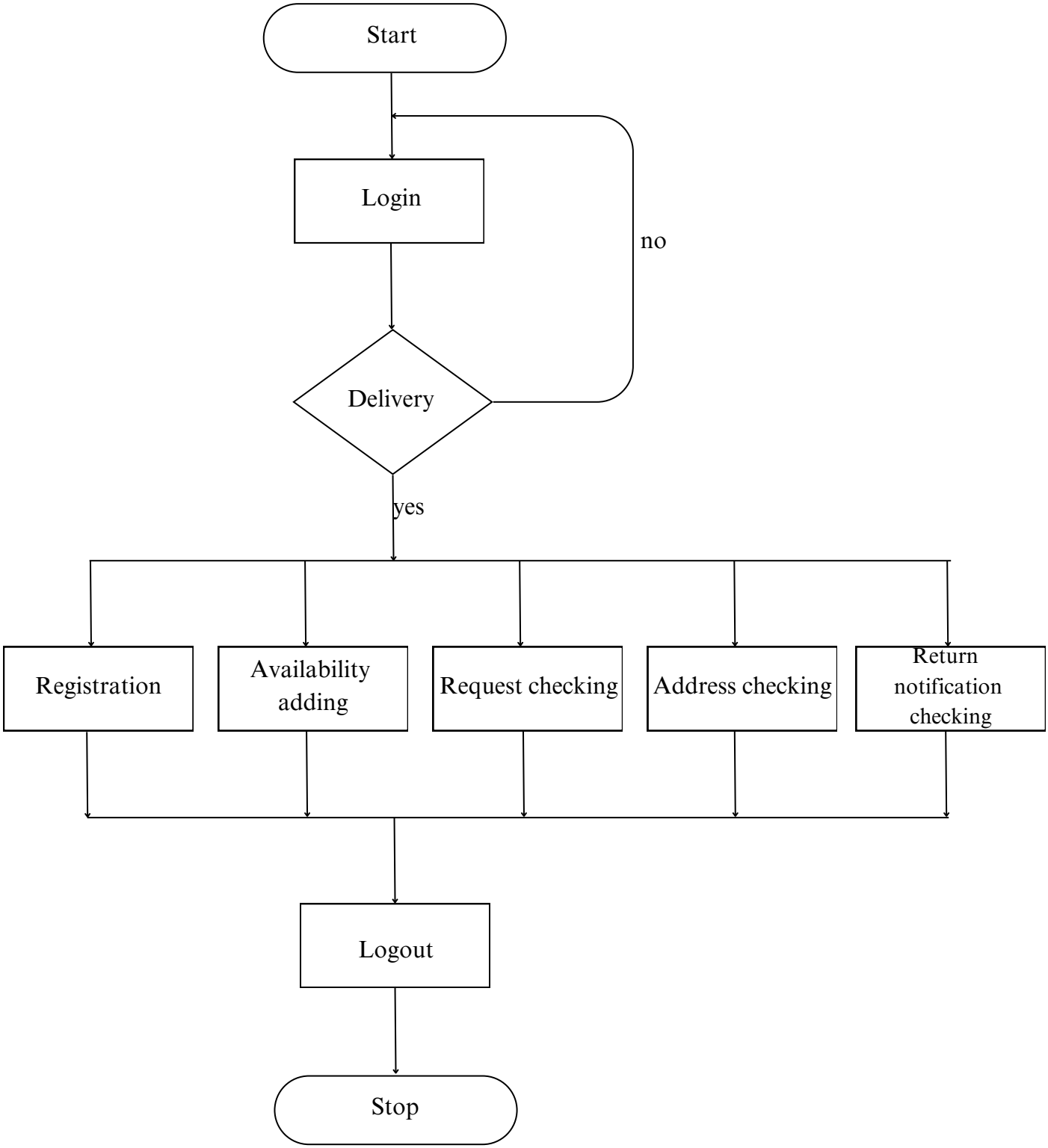
FLOWCHART FOR PUBLIC



FLOWCHART FOR GOVERNMENT



FLOWCHART FOR DELIVERY



6.SYSTEM CODING

6.1 INTRODUCTION

When considered as a step in software engineering, coding is viewed as a natural consequence after design. However, programming language characteristics and coding style can profoundly affect software quality and maintainability. The coding step translates a detailed representation into a programming language realization. The translation process continues when a compiler accepts source code as input and produces machine independent object code as output

6.2 FEATURES OF LANGUAGE

Python is a simple, readable, and dynamically typed, object-oriented, interpreted, byte-coded, portable, garbage-collected, and highly extensible programming language with strong exception handling and support for building scalable and distributed applications. Python is a high-level, general-purpose programming language, similar to C, Java, JavaScript, Ruby, and many others. You can use Python to develop web applications, automate tasks, process data, build artificial intelligence models, create games, and perform many other computing tasks. Python also allows developers to create special programs called scripts that can be executed without compilation, making it an excellent language for rapid development and automation. The following features make Python one of the most widely used and versatile programming languages today

- It is simple and easy to learn – Python's syntax is clear and concise, making it beginner friendly.
- It is object-oriented and supports multiple paradigms – Python allows procedural, functional, and object-oriented programming.
- It supports dynamic typing and automatic memory management – Variables do not require explicit type declarations, and garbage collection handles memory management efficiently.
- It is highly portable and platform-independent – Python code runs on Windows, macOS, Linux, and even embedded systems.
- It has a vast standard library – Built-in modules support a wide range of tasks, from file handling to networking and machine learning.
- It is secure and robust – Python's strong exception handling and security features make it suitable for enterprise applications.
- It supports multithreading and concurrent programming – Python provides threading and multiprocessing capabilities for performance optimization.
- It is widely used for web development and internet programming – Frameworks like Django and Flask simplify building powerful web applications.
- It is ideal for data science and artificial intelligence – Libraries such as NumPy, Pandas, TensorFlow, and Scikit-learn make Python the top choice for machine learning and data analysis

Python was created by Guido van Rossum in 1989 while working at Centrum Wiskunde & Informatica (CWI) in the Netherlands. The language was officially released in 1991 and was named after the British comedy group Monty Python. Unlike other programming languages that focus on performance alone, Python was designed to emphasize code readability and developer productivity. Over the years, Python has grown into one of the most popular and widely used programming languages across industries.

Python code is typically interpreted and executed by the Python Interpreter, which can run on any system with a compatible implementation. The most widely used implementation is CPython, but other implementations like PyPy, Jython (for Java integration), and IronPython (for .NET integration) also exist. Python's philosophy, often summarized by the Zen of Python, promotes simplicity, readability, and maintainability, making it a favorite choice for developers worldwide.

Today, Python is maintained by the Python Software Foundation (PSF) and continues to evolve with regular updates and new features. Its combination of simplicity, power, and versatility makes Python a leading programming language for software development, automation, web applications, scientific computing, and more

2.Django

Django is a high-level, open-source web framework for building robust and scalable web applications in Python. It follows the Model-View-Template (MVT) architectural pattern and is designed to accelerate development by promoting clean, reusable, and maintainable code. Django simplifies web application development by handling much of the complexity, allowing developers to focus on writing business logic rather than reinventing common functionalities.

Django provides a built-in admin interface, authentication system, database ORM (Object Relational Mapping), and security features that make it an ideal choice for developing secure, data-driven web applications.With features such as URL routing, form handling, and templating, Django streamlines the entire web development process. The framework also supports integration with various front-end technologies, enabling the creation of dynamic and interactive applications.

One of Django's key advantages is its extensibility and modular design. Developers can customize and extend its functionality through reusable Django apps, middleware, and third party packages. Configuration is simple, as Django follows the "convention over configuration" principle, minimizing boilerplate code. Additionally, Django's seamless integration with AJAX, RESTful APIs, and modern JavaScript frameworks makes it suitable for building responsive and feature-rich applications

Django also places a strong emphasis on security, providing built-in protection against SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and other common vulnerabilities. Its automatic database migration system simplifies schema changes, while its scalability and caching mechanisms make it ideal for enterprise-level applications.

With a strong community, comprehensive documentation, and a wide range of third-party plugins, Django continues to be a preferred choice for web developers looking to build secure, scalable, and efficient applications in Python. Whether you're developing content management systems, e-commerce platforms, social networks, or RESTful APIs, Django provides a powerful and efficient toolset to bring your ideas to life.

3.Sqlite3

SQLite3 is a widely used, open-source relational database management system (RDBMS) known for its simplicity, efficiency, and self-contained nature. Unlike traditional client-server databases like MySQL or PostgreSQL, SQLite is an embedded database, meaning it operates without a separate server process. It is designed to be lightweight, requiring minimal setup and configuration, making it an excellent choice for applications that need a fast, reliable, and low-overhead database solution.

SQLite fully supports SQL (Structured Query Language) and provides essential database functionalities, including transactions, ACID (Atomicity, Consistency, Isolation, Durability) compliance, and indexing. It stores data in a single disk file, making it easy to manage and deploy. Despite its compact size, SQLite is capable of handling substantial datasets and complex queries, making it ideal for embedded systems, mobile applications, desktop software, and small-to-medium-sized web applications.

One of SQLite3's key strengths is its portability and ease of integration. Since it does not require a separate database server, applications can use it without complex configuration or additional dependencies. It is widely supported across different programming languages and platforms, including Python, C, Java, and more. Additionally, SQLite3 offers features such as full-text search, JSON support, and encryption extensions, enhancing its functionality for modern application development.

SQLite3 is highly reliable and efficient, making it a preferred choice for applications like mobile apps (Android and iOS), IoT devices, web browsers, caching systems, and lightweight web applications. Its minimal footprint, zero-configuration nature, and strong performance ensure that it remains a popular database solution for developers seeking a fast, scalable, and easy-to-use embedded database system.

views.py

```
from django.shortcuts import render,redirect,get_object_or_404
from .models import *
from .forms import *
from django.contrib import messages

# Create your views here.

def farmer_home(request):
    return render( request ,'farmer_home.html')

def gov_home(request):
    return render( request ,'gov_home.html')

def landing_page(request):
    return render( request ,'index.html')

def admin_page(request):
    return render( request ,'admin.html')

def login_page(request):
    return render( request ,'login.html')

def public_home(request):
    return render( request ,'public.html')

def deliver_home(request):
    return render(request ,'deliver_home.html')

def logout_view(request):
  request.session.flush() # Clears all session data
  return redirect('login') # Redirect to login page
```

```
def register(request):
  if request.method == 'POST':
    detl=Reg_Form(request.POST)
    paswrd=login_form(request.POST)
    if detl.is_valid() and paswrd.is_valid():
      login_data=paswrd.save(commit=False)
      login_data.user_type = 'farmer'
      login_data.save()
      reg_data=detl.save(commit=False)
      reg_data.login_id=login_data
      reg_data.save()
      return redirect ('login')
  else:
    detl=Reg_Form()
    paswrd=login_form()
  return render (request,'register.html',{'detl':detl,'paswrd':paswrd})

def public_register(request):
  if request.method == 'POST':
    detl=public_Form(request.POST)
    paswrd=login_form(request.POST)
    if detl.is_valid() and paswrd.is_valid():
      login_data=paswrd.save(commit=False)
      login_data.user_type = 'public'
      login_data.save()
      reg_data=detl.save(commit=False)
      reg_data.login_id=login_data

      reg_data.save()
      return redirect ('login')
  else:
    detl=public_Form()
    paswrd=login_form()
  return render (request,'public_registration.html',{'detl':detl,'paswrd':paswrd})
```

```
def deliver_register(request):
  logid=request.session.get('farmer_id')
  farid=get_object_or_404(login,id=logid)
  if request.method== 'POST':
    detl=deliver_Form(request.POST)
    paswrd=login_form(request.POST)
    if detl.is_valid() and paswrd.is_valid():
      login_data=paswrd.save(commit=False)
      login_data.user_type = 'delivery'
      login_data.save()
      reg_data=detl.save(commit=False)
      reg_data.login_id=login_data
      reg_data.far_id=farid

      reg_data.save()
      return redirect ('farmer')
  else:
    detl=deliver_Form()
    paswrd=login_form()
  return render (request,'delivery_registration.html',{'detl':detl,'paswrd':paswrd})
```

```
def logins(request):
  if request.method =='POST':
    form=login_verify(request.POST)
    if form.is_valid():
      email =form.cleaned_data['email']
      password =form.cleaned_data['password']
      try:
        user = login.objects.get(email=email)
        if user.password == password:
          if user.user_type=='public':
            request.session['public_id']=user.id
            return redirect('public')
          elif user.user_type=='farmer':
            request.session['farmer_id']=user.id
            return redirect('farmer')
          elif user.user_type=='gov':
            request.session['gov_id']=user.id
            return redirect('gov')
          elif user.user_type=='delivery':
            request.session['deliver_id']=user.id
            return redirect('delivery')
        else:
          messages.error(request, 'Invalid password')
      except login.DoesNotExist:
        messages.error(request, 'User does not exist')
  else:
    form = login_verify()
  return render(request,'login.html',{'form':form})
```

```
def delivery_profile_view(request):
  user_id=request.session.get('deliver_id')

  user_login_data=get_object_or_404(login,id=user_id)
  user_data=get_object_or_404(delivery_boy,login_id=user_login_data.id)

  if request.method == 'POST':
    form=deliver_Form(request.POST,instance=user_data)
    logform=login_edit_form(instance=user_login_data)
    if form.is_valid() and logform.is_valid():
      form.save()
      logform.save()
      return redirect('delivery')
  else:
    form=user_edit_form(instance=user_data)
    logform = login_edit_form(instance = user_login_data)
  return render(request,'deliver_pro.html',{'form':form,'logform':logform})

def table_view(request):
  farmer=user.objects.all()
  return render(request,'datatable.html',{'farmers' : farmer})

def usre_edit(request,id):
  user = get_object_or_404(login,id=id)
  if request.method == 'POST':
    form = Reg_Form(request.POST,instance=user)
    if form.is_valid():
      form.save()
      return redirect('userData')
  else:
    form=Reg_Form(instance=user)
  return render(request,'edit_page.html',{'form':form})
```

```
def delivery_profile_view(request):
  user_id=request.session.get('deliver_id')

  user_login_data=get_object_or_404(login,id=user_id)
  user_data=get_object_or_404(delivery_boy,login_id=user_login_data.id)

  if request.method == 'POST':
    form=deliver_Form(request.POST,instance=user_data)
    logform=login_edit_form(instance=user_login_data)
    if form.is_valid() and logform.is_valid():
      form.save()
      logform.save()
      return redirect('delivery')
  else:
    form=user_edit_form(instance=user_data)
    logform = login_edit_form(instance = user_login_data)
  return render(request,'deliver_pro.html',{'form':form,'logform':logform})

def table_view(request):
  farmer=user.objects.all()
  return render(request,'datatable.html',{'farmers' : farmer})

def usre_edit(request,id):
  user = get_object_or_404(login,id=id)
  if request.method == 'POST':
    form = Reg_Form(request.POST,instance=user)
    if form.is_valid():
      form.save()
      return redirect('userData')
  else:
    form=Reg_Form(instance=user)
  return render(request,'edit_page.html',{'form':form})
```

```
def profile_view(request):
  # user = get_object_or_404(login,id=id)
  user_id=request.session.get('farmer_id')
  user_login_data = get_object_or_404(login,id=user_id)
  user_data = get_object_or_404(user,login_id=user_login_data)

  if request.method == 'POST':
    form = user_edit_form(request.POST,instance=user_data)
    logform=login_edit_form(request.POST,instance=user_login_data)
    if form.is_valid() and logform.is_valid():
      form.save()
      logform.save()
      return redirect('farmer')
  else:
    form=user_edit_form(instance=user_data)
    logform = login_edit_form(instance = user_login_data)
  return render(request,'edit.html',{'form':form,'logform':logform})

def public_profile_view(request):
  # user = get_object_or_404(login,id=id)
  user_id=request.session.get('public_id')
  user_login_data = get_object_or_404(login,id=user_id)
  user_data = get_object_or_404(public_user,login_id=user_login_data)

  if request.method == 'POST':
    form = public_Form(request.POST,instance=user_data)
    logform=login_edit_form(request.POST,instance=user_login_data)
    if form.is_valid() and logform.is_valid():
      form.save()
      logform.save()
      return redirect('public')
  else:
    form=user_edit_form(instance=user_data)
    logform = login_edit_form(instance = user_login_data)
  return render(request,'public_pro_edit.html',{'form':form,'logform':logform})
```

```
def product_add(request):
  user_id=request.session.get('farmer_id')
  user_login_data = get_object_or_404(login,id=user_id)
  if request.method=='POST':

    product=products_form(request.POST,request.FILES)
    if product.is_valid():

      pro_data=product.save(commit=False)
      pro_data.login_id=user_login_data
      pro_data.save()
      return redirect('product_view')
  else:
    product=products_form()
  return render(request,'products.html',{'product':product})

def product_view(request):
  user_id=request.session.get('farmer_id')
  user_login_data = get_object_or_404(login,id=user_id)
  all_product=products.objects.filter(login_id=user_login_data)
  return render(request,'product_view.html',{'all_products' : all_product})

def product_del(request,id):
  product=get_object_or_404(products,id=id)
  product.delete()
  return redirect('product_view')

def product_edit(request,id):
  product=get_object_or_404(products,id=id)
  if request.method=='POST':
    form=products_form(request.POST,request.FILES,instance=product)
    if form.is_valid():
      form.save()
    return redirect('product_view')
  else:
    form=products_form(instance=product)
  return render(request ,'product_edit.html',{'product' : form})
```

```
def public_pro_view(request):
  all_product=products.objects.all()
  return render(request,'public_product_view.html',{'all_products' : all_product})

def add_to_cart(request,p_id):
  u_id=request.session.get('public_id')
  user_login_data = get_object_or_404(login, id=u_id)
  product=get_object_or_404(products, id=p_id)
  if cart.objects.filter(product_id=product,user_id=user_login_data,cancelation_status=0).exists():
    messages.success(request,'product already exists')
  else:
    cart_data = cart.objects.create(
      product_id = product,
      user_id = user_login_data
    )
  return redirect('public_pro_view')

def cart_view(request):
  user_id=request.session.get('public_id')
  user_login_data = get_object_or_404(login,id=user_id)
  cart_product=cart.objects.filter(user_id=user_login_data,payment_status=0)
  return render(request,'cart.html',{'cart_products' : cart_product})

def cart_product_del(request,id):
  cart_product=get_object_or_404(cart,id=id)
  cart_product.delete()
  return redirect('cart_view')
```

```
def payment_dtel(request,id,cartid):
  user_id=request.session.get('public_id')
  user_login_data = get_object_or_404(login,id=user_id)
  crt_id = get_object_or_404(cart,id=cartid)
  if request.method=='POST':

    payment=payment_form(request.POST)
    if payment.is_valid():

      payment_data=payment.save(commit=False)
      payment_data.login_id=user_login_data
      payment_data.cart_id=crt_id
      payment_data.save()
      # c=cart.objects.get(id=crt_id)
      # c= get_object_or_404(cart,id=cartid)
      crt_id.payment_status=1
      crt_id.save()
      return redirect('cart_view')
  else:
    payment=payment_form()

  return render( request ,'payment.html',{'payment':payment})

def my_order(request):
  user=request.session.get('public_id')
  users_id = get_object_or_404(login,id=user)
  myord = cart.objects.filter(user_id=users_id,payment_status=1)
  return render(request,'my_orders.html',{'myords':myord})

def farmer_order_view(request):
  farmer = request.session.get('farmer_id')
  farmer_id = get_object_or_404(login, id=farmer)
prt=cart.objects.filter(product_id__login_id=farmer_id,payment_status=1).select_related('us
er_id__us')
  return render(request, 'farmer_order_view.html', {'prts': prt})
```

```
def my_order_cancel(request,id):
  user=request.session.get('public_id')
  users_id = get_object_or_404(login,id=user)
  crt_id = get_object_or_404(cart,id=id)
  crt_id.cancelation_status=1
  crt_id.save()
  return redirect('public_order_view')

def alert_add(request):
  if request.method == "POST":
    form = AlertForm(request.POST)
    if form.is_valid():
      form.save()
      return redirect('ad_alert_view')
  else:
    form = AlertForm()
  return render(request, 'add_alert.html', {'form': form})

def admin_alert_view(request):
 alerts = alert.objects.all()
 return render(request, 'admin_alert_view.html',{'alerts':alerts})

def edit_alert(request,id):
  alerts = get_object_or_404(alert,id=id)
  if request.method == 'POST':
    form = AlertForm(request.POST,instance=alerts)
    if form.is_valid():
      form.save()
    return redirect('ad_alert_view')
  else:
    form = AlertForm(instance=alerts)
  return render(request,'edit_alert.html',{'forms':form})

def del_alert(request,id):
  alerts=get_object_or_404(alert,id=id)
  alerts.delete()
  return redirect('ad_alert_view')
```

```
def fa_alert_view(request):
  alerts = alert.objects.all().order_by('-created_at')
  return render(request, 'farmer_alert_view.html', {'alerts': alerts})

def gov_product_add(request):
 if request.method=='POST':

   product=gov_products_form(request.POST,request.FILES)
   if product.is_valid():
     product.save()
     return redirect('gov_view_pro')
 else:
   product=gov_products_form()
 return render(request,'gov_prod_add.html',{'product':product})

def gov_product_view(request):
 gov_product=gov_products.objects.all()
 return render(request,'gov_product_view.html',{'gov_product':gov_product})

def gov_pro_edit(request,id):
  products=get_object_or_404(gov_products,id=id)
  if request.method=='POST':
    form=gov_products_form(request.POST,request.FILES,instance=products)
    if form.is_valid():
      form.save()
    return redirect('gov_view_pro')
  else:
    form=gov_products_form(instance=products)
  return render(request ,'gov_prod_edit.html',{'product' : form})

def del_gov_pro(request,id):
 gov_pro=get_object_or_404(gov_products,id=id)
 gov_pro.delete()
 return redirect('gov_view_pro')

def far_subsidy_view(request):
 all_product=gov_products.objects.all()
 return render(request,'farmer_product_view.html',{'all_products' : all_product})
```

```
def f_add_to_cart(request,id):
  f_id=request.session.get('farmer_id')
  farmer_login_data = get_object_or_404(login, id=f_id)
  product=get_object_or_404(gov_products, id=id)
if
farmer_cart.objects.filter(product_id=product,user_id=farmer_login_data,cancelation_status
=0).exists():
    messages.success(request,'product already exists')
  else:
    cart_data = farmer_cart.objects.create(
      product_id = product,
      user_id =farmer_login_data
    )
  return redirect('frmr_pro_view')

def far_cart_view(request):
  far_id=request.session.get('farmer_id')
  user_login_data = get_object_or_404(login,id=far_id)
  cart_product=farmer_cart.objects.filter(user_id=user_login_data,payment_status=0)
  return render(request,'frmr_pro_view.html',{'cart_products' : cart_product})

def cart_product_del(request,id):
  cart_product=get_object_or_404(farmer_cart,id=id)
  cart_product.delete()
  return redirect('cart_view')
```

```
def far_payment_dtel(request,id):
  far_id=request.session.get('farmer_id')
  farmer_login_data = get_object_or_404(login,id=far_id)
  crt_id = get_object_or_404(farmer_cart,id=id)
  if request.method=='POST':

    payment=farmer_payment_form(request.POST)
    if payment.is_valid():

      payment_data=payment.save(commit=False)
      payment_data.login_id=farmer_login_data
      payment_data.cart_id=crt_id
      payment_data.save()
      # c=cart.objects.get(id=crt_id)
      c= get_object_or_404(farmer_cart,id=id)
      c.payment_status=1
      c.save()
      return redirect('frmr_pro_view')
  else:
    payment=farmer_payment_form()

  return render( request ,'farmer_payment.html',{'payment':payment})

def far_order(request):
  far=request.session.get('farmer_id')
  far_id = get_object_or_404(login,id=far)
  myord = farmer_cart.objects.filter(user_id=far_id,payment_status=1)
  return render(request,'far_my_orders.html',{'myords':myord})

def far_order_cancel(request,id):
  far=request.session.get('farmer_id')
  far_id = get_object_or_404(login,id=far)
  crt_id = get_object_or_404(farmer_cart,id=id)
  crt_id.cancelation_status=1
  crt_id.save()
  return redirect('farmer_order_view')
```

```
def gov_orders(request):
  prt = farmer_cart.objects.filter( payment_status=1).select_related('user_id__us')
  return render(request, 'gov_orders_views.html', {'prts': prt})

def notification_add(request):
 if request.method == "POST":
   form = NotificatioForm(request.POST)
   if form.is_valid():
     form.save()
     return redirect('gov_notify_view')
 else:
   form = NotificatioForm()
 return render(request, 'add_notification.html', {'form': form})

def gov_notification_view(request):
 alerts = notification.objects.all()
 return render(request, 'gov_notification_view.html',{'alerts':alerts})

def edit_notification(request,id):
 alerts = get_object_or_404(notification,id=id)
 if request.method == 'POST':
   form = NotificatioForm(request.POST,instance=alerts)
   if form.is_valid():
     form.save()
   return redirect('gov_notify_view')
 else:
   form = NotificatioForm(instance=alerts)
 return render(request,'notification_edit.html',{'forms':form})

def del_notification(request,id):
 alerts=get_object_or_404(notification,id=id)
 alerts.delete()
 return redirect('gov_notify_view')
```

```
def fa_notification_view(request):
  alerts = notification.objects.all().order_by('-created_at')
  return render(request, 'farmer_notification_view.html', {'alerts': alerts})

def address_add(request,id):
 user=request.session.get('public_id')
 logid=get_object_or_404(login,id=user)
 cart_id = get_object_or_404(cart,id = id )
 if request.method =="POST":
   form=address_form(request.POST)
   print(form)
   if form.is_valid():
     a = form.save(commit=False)
     a.login_id = logid
     a.save()
     return redirect('payment',a.id,cart_id.id)
 else:
   form=address_form()
   return render(request, 'public_address.html',{'form':form})

def all_delivery_boys(request):
  user=request.session.get('farmer_id')
  # logid=get_object_or_404(login,id=user)
  # delivery = get_object_or_404(delivery_boy,far_id = logid )
  delivery = delivery_boy.objects.filter(far_id=user)
  return render(request, 'farmer_delivery_view.html',{'delivery':delivery})

def del_cancelled(request,id):
  crt=get_object_or_404(cart,id=id)
  crt.delete()
  return redirect('public_order_view')

def ord_avil(request, id):
  farmer = request.session.get('farmer_id')
  farmer_id = get_object_or_404(login, id=farmer)
  prt = cart.objects.filter(product_id__login_id=farmer_id, payment_status=1 ,delivery_status
= 0).select_related('user_id__us')
  return render(request, 'avilable_pro.html', {'prts': prt, 'delivery_id': id})
```

```
def assigning_delivery(request, id, cartid):
  cart_instance = get_object_or_404(cart, id=cartid)
  delivery_team = get_object_or_404(delivery_boy, id=id)

  if delivery_assign.objects.filter(delivery_team_id=delivery_team,status=0).exists:
    delivery_assign.objects.create(
      cart_id=cart_instance,
      delivery_team_id=delivery_team,
      status=1
    )
    cart_instance.delivery_status = 1
    cart_instance.save()
  else:
    messages.error(request, 'The Delivery Boy is Busy')
  return redirect('fa_delivery_view')

def del_reqs(request):
  deliv = request.session.get('deliver_id')
  deliv_id = get_object_or_404(delivery_boy, login_id=deliv)
  delivery_req = delivery_assign.objects.filter(delivery_team_id=deliv_id)
  # Attach address object to each delivery request
  for req in delivery_req:
    user = req.cart_id.user_id # user related to the cart
    req.address = address.objects.filter(login_id=user).first() # Get address for that user
  return render(request, 'delivery_req.html', {'prts': delivery_req})

def delevery_complete(request,id):
  # dele_boy = request.session.get('deliver_id')
  # print(dele_boy)
  deliv_id = get_object_or_404(delivery_assign, id=id)
  ad = deliv_id.cart_id.id
  deliv_id.status=0
  deliv_id.save()
  del_cart = get_object_or_404(cart,id=ad)
  del_cart.delivery_status=2
  del_cart.save()
  return redirect('delivery_request')
```

forms.py

```
from django import forms
from .models import *
class Reg_Form(forms.ModelForm):

  class Meta:
    model = user
    fields = ['name', 'contact','far_address','district','state','farm_size','main_products','id_proof']

class login_form(forms.ModelForm):

  class Meta:
    model = login
    fields = ['email', 'password']

class login_verify(forms.Form):
  email = forms.CharField(max_length=100)
  password = forms.CharField(widget=forms.PasswordInput)

class user_edit_form(forms.ModelForm):

  class Meta:
    model = user
    fields = ['name', 'contact']
    widgets = {
      'name':forms.TextInput(attrs={'class':'form-control'}),
      'contact':forms.TextInput(attrs={'class':'form-control'})
    }

class login_edit_form(forms.ModelForm):

  class Meta:
    model = login
    fields = ['email']
    widgets = {
      'email':forms.TextInput(attrs={'class':'form-control'})
    }
```

```
class public_Form(forms.ModelForm):

  class Meta:
    model = public_user
    fields = ['name', 'contact']

class products_form(forms.ModelForm):
  class Meta:
    model=products
    fields=['category','name','image','price']

class payment_form(forms.ModelForm):
  class Meta:
    model=payment
    fields=['onwer_name','card_no','cvv','exp_month','exp_year']

class farmer_payment_form(forms.ModelForm):
  class Meta:
    model=farmer_payment
    fields=['onwer_name','card_no','cvv','exp_month','exp_year']

class AlertForm(forms.ModelForm):
  class Meta:
    model = alert
    fields = ['message']
    widgets = {
      'message': forms.Textarea(attrs={'placeholder': 'Enter your alert message...', 'rows': 4}),
    }


  class NotificatioForm(forms.ModelForm):
    class Meta:
      model = notification
      fields = ['message']
      widgets = {
        'message': forms.Textarea(attrs={'placeholder': 'Enter the notification...', 'rows': 4}),
      }
  class address_form(forms.ModelForm):
    class Meta:
      model=address
      fields=['name','contact','house_name','area','landmark','pincode','city','state']
```

models.py

```
from django.db import models

# Create your models here.
class login(models.Model):
  email=models.EmailField(unique=True)
  password=models.CharField(max_length=100)
  user_type = models.CharField(max_length=20)

class user(models.Model):
  name=models.CharField(max_length=100)
  contact=models.CharField(max_length=15)

login_id=models.OneToOneField(login,on_delete=models.CASCADE,null=True,blank=Tru
e,related_name='far')
  far_address=models.CharField(max_length=200)
  district=models.CharField(max_length=50)
  state=models.CharField(max_length=50)
  farm_size=models.IntegerField(default=0)
  main_products=models.CharField(max_length=200)
  id_proof = models.FileField(upload_to='id_proofs/')

class public_user(models.Model):
  name=models.CharField(max_length=100)
  contact=models.CharField(max_length=15)
login_id=models.OneToOneField(login,on_delete=models.CASCADE,null=True,blank=Tru
e,related_name='us')
class products(models.Model):
  category = models.CharField(max_length=100)
  name = models.CharField(max_length=50)
  image = models.ImageField(upload_to='images')
  price = models.CharField(max_length=5)
  login_id=models.ForeignKey(login,on_delete=models.CASCADE,null=True,blank=True)

class alert(models.Model):
  message = models.TextField()
  created_at = models.DateTimeField(auto_now=True)

class notification(models.Model):
  message = models.TextField()
  created_at = models.DateTimeField(auto_now=True)
```

```
class cart(models.Model):
product_id=models.ForeignKey(products,on_delete=models.CASCADE,null=True,blank=T
rue)
user_id=models.ForeignKey(login,on_delete=models.CASCADE,null=True,blank=True,relat
ed_name='users')
  payment_status = models.IntegerField(default=0)
  cancelation_status = models.IntegerField(default=0)
  delivery_status = models.IntegerField(default=0)
  current_date = models.DateTimeField(auto_now_add=True)

class payment(models.Model):
  onwer_name = models.CharField(max_length=25)
  card_no = models.CharField(max_length=15)
  cvv = models.CharField(max_length=5)
  exp_month = models.IntegerField()
  exp_year = models.IntegerField()
  amount = models.IntegerField(default=0)
  cart_id = models.ForeignKey(cart,on_delete=models.CASCADE,null=True,blank=True)
  login_id = models.ForeignKey(login,on_delete=models.CASCADE,null=True,blank=True)
  current_date = models.DateTimeField(auto_now_add=True)

class address(models.Model):
  name=models.CharField(max_length=100)
  contact=models.CharField(max_length=15)
  house_name=models.CharField(max_length=100)
  area=models.CharField(max_length=100)
  landmark=models.CharField(max_length=100)
  pincode=models.CharField(max_length=10)
  city=models.CharField(max_length=100)
  state=models.CharField(max_length=100)
login_id=models.ForeignKey(login,on_delete=models.CASCADE,null=True,blank=True,rel
ated_name='public')

class delivery_boy(models.Model):
  name=models.CharField(max_length=100)
  contact=models.CharField(max_length=15)
far_id=models.ForeignKey(login,on_delete=models.CASCADE,related_name='farmer_as_ta
ble',null=True,blank=True)
login_id=models.ForeignKey(login,on_delete=models.CASCADE,related_name='login_as_t
able',null=True,blank=True)
```

```
class cart(models.Model):
product_id=models.ForeignKey(products,on_delete=models.CASCADE,null=True,blank=T
rue)
user_id=models.ForeignKey(login,on_delete=models.CASCADE,null=True,blank=True,relat
ed_name='users')
  payment_status = models.IntegerField(default=0)
  cancelation_status = models.IntegerField(default=0)
  delivery_status = models.IntegerField(default=0)
  current_date = models.DateTimeField(auto_now_add=True)

class payment(models.Model):
  onwer_name = models.CharField(max_length=25)
  card_no = models.CharField(max_length=15)
  cvv = models.CharField(max_length=5)
  exp_month = models.IntegerField()
  exp_year = models.IntegerField()
  amount = models.IntegerField(default=0)
  cart_id = models.ForeignKey(cart,on_delete=models.CASCADE,null=True,blank=True)
  login_id = models.ForeignKey(login,on_delete=models.CASCADE,null=True,blank=True)
  current_date = models.DateTimeField(auto_now_add=True)

class address(models.Model):
  name=models.CharField(max_length=100)
  contact=models.CharField(max_length=15)
  house_name=models.CharField(max_length=100)
  area=models.CharField(max_length=100)
  landmark=models.CharField(max_length=100)
  pincode=models.CharField(max_length=10)
  city=models.CharField(max_length=100)
  state=models.CharField(max_length=100)
login_id=models.ForeignKey(login,on_delete=models.CASCADE,null=True,blank=True,rel
ated_name='public')

class delivery_boy(models.Model):
  name=models.CharField(max_length=100)
  contact=models.CharField(max_length=15)
far_id=models.ForeignKey(login,on_delete=models.CASCADE,related_name='farmer_as_ta
ble',null=True,blank=True)
login_id=models.ForeignKey(login,on_delete=models.CASCADE,related_name='login_as_t
able',null=True,blank=True)
```

urls.py

```
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static
from . import views
urlpatterns = [
  path('farmer/', views.farmer_home, name='farmer'),
  path('gov/', views.gov_home, name='gov'),
  path('', views.landing_page, name='index'),
  path('index', views.landing_page, name='index'),
  path('admins/', views.admin_page, name='admins'),
  path('login/', views.logins, name='login'),
  path('logout/', views.logout_view, name='logout'),
  path('register/', views.register, name='register'),
  path('far-table/', views.table_view, name='far-table'),
  path('profile/',views.profile_view,name='profile'),
  path('public_profile_view/',views.public_profile_view,name='public_profile_view'),
  path('public_register/', views.public_register, name='public_register'),
  path('product_view/', views.product_view, name='product_view'),
  path('add_pro/', views.product_add, name='add_pro'),
  path('del_pro/<int:id>/',views.product_del,name='del_pro'),
  path('public/', views.public_home, name='public'),
  path('public_pro_view/', views.public_pro_view, name='public_pro_view'),
  path('edit_pro/<int:id>/',views.product_edit,name='edit_pro'),
  path('cart/<int:p_id>/',views.add_to_cart,name='cart'),
  path('cart_view',views.cart_view,name='cart_view'),
  path('cart_del/<int:id>/',views.cart_product_del,name='cart_del'),
  path('payment/<int:id>/',views.payment_dtel,name='payment'),
  path('order_view/',views.farmer_order_view,name='order_view'),
 path('alert',views.alert_add,name='alert'),
  path('ad_alert_view',views.admin_alert_view,name='ad_alert_view'),
  path('alert_edit/<int:id>/',views.edit_alert,name='alert_edit'),
  path('alert_del/<int:id>/',views.del_alert,name='alert_del'),
  path('far_alert_view/',views.fa_alert_view,name='far_alert_view'),
  path('gov_add_pro/', views.gov_product_add, name='gov_add_pro'),
  path('gov_view_pro/', views.gov_product_view, name='gov_view_pro'),
  path('gov_edit_pro/<int:id>/', views.gov_pro_edit, name='gov_edit_pro'),
  path('gov_pro_del/<int:id>/',views.del_gov_pro,name='gov_pro_del'),
  path('subsidy_pro',views.far_subsidy_view,name='subsidy_pro'),
  path('farmer_cart/<int:id>/',views.f_add_to_cart,name='farmer_cart'),
  path('far_payment/<int:id>/',views.far_payment_dtel,name='far_payment'),
  path('frmr_pro_view/',views.far_cart_view,name='frmr_pro_view'),
```

```
path('gov_order_view/',views.gov_orders,name='gov_order_view'),
  path('notify/',views.notification_add,name='notify'),
  path('gov_notify_view/',views.gov_notification_view,name='gov_notify_view'),
  path('gov_notify_edit/<int:id>',views.edit_notification,name='gov_notify_edit'),
  path('gov_notify_del/<int:id>',views.del_notification,name='gov_notify_del'),
  path('add_address/<int:id>',views.address_add,name='add_address'),
  path('payment/<int:id>/<int:cartid>',views.payment_dtel,name='payment'),
  path('delivery_register/',views.deliver_register,name='delivery_register'),
  path('delivery/',views.deliver_home,name='delivery'),
   path('deliver_profile_view/', views.delivery_profile_view, name='deliver_profile_view'),
    path('fa_notification_view/',views.fa_notification_view,name='fa_notification_view'),
    path('fa_delivery_view/',views.all_delivery_boys,name='fa_delivery_view'),
    path('cancel_del/<int:id>',views.del_cancelled,name='cancel_del'),
    path('avilable_orders/<int:id>',views.ord_avil,name='avilable_orders'),
    path('assignig/<int:id>/<int:cartid>',views.assigning_delivery,name='assignig'),
    path('delivery_request',views.del_reqs,name='delivery_request'),
    path('completed/<int:id>',views.delevery_complete,name='completed'),
    path('farmer_public_chat/<int:id>',views.farmer_user_chat,name='farmer_public_chat'),
    path('public_farmer_chat/<int:id>',views.user_farmer_chat,name='public_farmer_chat')
 ]+static(settings.MEDIA_URL,document_root = settings.MEDIA_ROOT)
```

7. SYSTEM TESTING

7.1 INTRODUCTION

Software testing is a critical element of quality assurance and represents the ultimate Previews of specification, design, and coding. Testing represents an interesting anomaly for the software. During the earlier definition and development phase it was attempted to build software from an abstract concept to a tangible implementation. The testing phase involves the testing of the developed system using various test data. After preparing the test data the system understudies the test using those test data. While testing the system by using test data, errors were found and corrected. Thus, a series of tests were performed for the proposed system before the system was ready for implementation. Testing is a set of activities that can be planned and conducted systematically. Before going for testing the following things are taken into consideration.

- To ensure that the information is properly placed in and out of the program.
- To find out whether the local data structures maintain their integrity during all spells in an algorithm execution.
- To ensure that the module operates properly at boundaries established to limit are restrict processing.
- To find out whether error-handling paths are working correctly are not.

The following testing methods are used for testing the software.

- Unit Testing
- Integration Testing
- Validation Testing
- User Acceptance Testing
- Black Box Testing
- White Box Testing

7.2 UNIT TESTING

Unit testing focuses verification effect on the smallest limit of software design. It comprises the set of tests performed by an individual programmer before the integration of a unit into a large system. Using the unit test plan prepared in the design phase of the system, important control paths are tested to uncover the errors within the module. This testing was carried out during the coding itself. In this testing step, each module is going to be working satisfactorily as the expected output from the module. Project aspect: Front-end design consists of various forms. They are tested for data acceptance. Similarly, the back end that is the database was also tested for successful acceptance and retrieval of data.

7.3 INTEGRATION TESTING

Integration testing is the systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with the interface. The objective is to take unit-tested modules and build the program structures that have been dictated by design.

All modules are combined in this testing step. Then the entire program is tested. Project aspect: Using integrated test plans prepared in the design phase of the system developed as s guide, the integration was carried out. All the errors found in the system were corrected for the next testing steps.

7.4 VALIDATION TESTING

At the end of integration testing, the software is completely assembled as a package, interfacing errors have been uncovered and corrected and a final series of software validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in a manner that can be reasonably accepted by the user/customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirements. After the validation test has been completed, one of the following two possible conditions exists. The function or performance characteristics confirm specification and are accepted. A deviation from the specification is found and a deficiency list is created. Project aspect: The proposed system under consideration has been tested by using validation testing and found to be automatically logged off when his time slot is over. The option for idle system shutdown is also validated.

7.5 USER ACCEPTANCE TESTING

User acceptance of a system is a key factor in the success of any system. The system under consideration was tested for user acceptance by constantly keeping in touch with the prospective system user at the time of development and making changes wherever required.

Project aspect: User acceptance testing is done on the following points.

- Input screen design.
- Output screen design
- Online message to guide the user.

7.6 WHITE BOX TESTING

White box testing of software is predicated on a close examination of procedural detail. The status of the program may be tested at various points to determine whether the expected asserted status corresponds to the actual status.

Project aspect: Using the following test cases can be derived

- Exercise all logical conditions on their true and false side.
- Exercise all loops within their boundaries and their operation bounds.
- Exercise internal data structure to ensure their validity.

8.SYSTEM IMPLEMENTATION

8.1 INTRODUCTION

Implementation is the process in which the working product is installed at the client and configured and customized to its operational environment. Implementation is less creative than software design, but it may tend to be the source of new bugs if procedures aren't followed strictly. Almost patience and very good technical are a must for the Application Developer. Implementation procedure normally begins with preparing the target machines. The risk of crashes and failures can be reduced to bare minimum if the correct system configurations are strictly enforced

The new system may be totally new replacing of existing manual or automated system or it may be major modification to existing system. The methods of implementation and time scale adopted are fond out initially. The system is tested properly and at the same time the admin are trained in the new procedure. Proper implementation is essential to provide a reliable system to meet organization requirements. Successful implementations may not guarantee improvement in the organization using the new system, but it will prevent improper installation.

Implementation stage involves following tasks:

- Careful planning.
- Investigation of system and constraints.
- Design of methods to achieve the changeover.
- Evaluation of changeover method.
- Training of the staff in changeover phase

The method of implementation and the time scale to be adopted are found out initially. Next the system is tested properly and the same users are trained in new environment.

8.2 IMPLEMENTATION PROCEDURE

Implementation of software refers to the final installation of the package in its real environment, to the satisfaction of intended users and operation of the system. In the initial stage, the doubt about the software but have to ensure that the resistance does not built up, as one has to make sure that:

- Active user must be aware of benefits of using a new system.
- Their confidence in software is built-up.
- Proper guidance is imparted to the user so that is comfortable in using that system.
- Periodical changes must be adapted in to the system to make customers always using the system.

8.3 MAINTENANCE

It is possible to produce systems of any size which do not need to be changed. Over the lifetime of a system, its original requirements will be modified to reflect the changing user. After implementation, maintenance is the important process. Usually once the system is

implemented, the software developers and customer would sign a contract. According to the time mentioned in the contract all errors and requirements would be done free of cost. Once the maintenance period is over all the logical errors will be corrected free of cost were as all extra requirements would be charged. During the contract period we would frequently visit the site where the system is implemented and check the system performance such as response time and also how it works at peak hours. If any problem is found it is corrected. Software development does not freeze at the moment of delivery. Usually, software must grow and change over time.

These activities are collectively referred to as software maintenance. Application updates are part of normal maintenance phase of development life cycle. A modification effort is actually a small project and must proceed through all the phases of development process. There are many reasons for software modification and continued development after the first release. Application may need additional features not discovered during the original analysis and design. Ease if maintenance is a part of every step in development. If the analysis is complete, users will find the most important features in the first release of software.

If the design and coding is done perfectly, then it will be very easy to maintain later. Some of the suggestions are group the changes and deliver another release rather than incremental changes, so that it will force to be more through about new researches. Give more to small requirements. The experience in coding will be an added advantage because integrating a new code with existing code is normally difficult. The working of the system is observed in a local machine and intranet and it was found satisfactory

The four types of maintenance activities are listed below:

8.3.1 Corrective Maintenance

This is concerned with fixing reported errors in the software. Coding errors are cheap to correct; design errors are more expensive as they may involve the rewriting of several program components. Requirements errors are the most expensive to repair because of the extensive system redesign which may be necessary

8.3.2 Adaptive Maintenance

This means changing the software to some new environment such as different hardware platform or for use with different operating system. The software functionality does not radically change. Any system that involves JVM can run this software.

8.3.3 Perfective Maintenance

This involves implementing a new functional or non functional system requirement. These are generated by software consumer as their organization or business changes.

8.3.4 Preventive Maintenance

This occurs when software is changed to improve future maintainability or reliability or to provide a better basis for future enhancements. In the current project, all the above maintenance was implemented

9. SECURITY, BACKUP AND RECOVERY MECHANISMS

9.1 INTRODUCTION

Security is an important consideration in application. By default Django applications are available to any user who can connect to our server. Although this is ideal for many applications it is not always appropriate. The first step in securing our application is deciding where you need security and what it needs to protect. Security concept:

9.2 AUTHENTICATION

This is the process of determining a user's identity and forcing users to prove they are who they claim to be, usually this involves entering credential (email and password) in some sort of login or windows

9.3 AUTHORIZATION

Once the user is authenticated, authorization is the process of determining where that user has sufficient permission to perform a given action, such as viewing a page or retrieving information from database. Backup facility is used in this software for backing up of data. If any error occurs in the database due to any database error or software error or if the database is detected in any fault operation, you can copy the backup file to solve this problem. For protecting the system from any kind of loss or damage, backup facility is offered. The entire program that is associated with the database can be saved into CD or DVD or floppy disc for the purpose of future use. If the program is lost due to some kind of system failure, the backup copy of program on any of the above disc can be used. The data or those discs will not be lost due to any usual failure

Backup and Recovery Mechanism: The Administrator can create dump files of MySQL Server database everyday and keep it in secondary storage devices. When the system crashes or failures, the administrator should follow the system installation procedures and restore this dump file into MySQL Server database. Then the system will work as normally

10. ONLINE HELP AND USER MANUAL

10.1 INTRODUCTION

"Yieldsmart" is a user friendly system. This site provides online to the users of the system. Walk with us site is topic oriented, procedural or reference information delivered through computer software. It is a form of user assistance. Most online is designed to give assistance in the user of software or operating system, but can also be to represent information on the broad range of subjects. The user manual provides the detailed description regarding the usage of the software.

10.2 USER MANUAL

- The User Manuals provides the detailed description regarding the usage of the software. This helps the user in having more clarity and the purpose of the website. It includes the functionalities, features, do's and dont's and along with how to use the software. Thus, the complexity will be reduced. The main user tips are:
- All the required operations are specified in various links.
- Never share your user id and password.
- Do not write your user id and password down in an unsecured environment.
- Do not send extremely confidential information as autograph to any user.
- Change your password periodically.
- If your browser prompts to save user id and password, cancel as it is not safe to store.

To Log on for the first time, please follow the steps below:

- Go to the user log on page of the website and click on the hyperlink for sign up or create new account.
- Enter the personal details and also give preferred user-id and password

11.CONCLUSION

"**YieldSmart**" is an innovative crop yield prediction system designed to transform traditional farming through data-driven insights. By integrating machine learning algorithms with real-time weather data and soil analytics, the system provides farmers with accurate yield forecasts. This enables them to make well-informed decisions regarding irrigation, fertilization, and harvesting, ultimately improving agricultural efficiency and sustainability. The system's ability to deliver personalized recommendations and real-time alerts ensures that farmers can respond proactively to environmental changes, minimizing risks and maximizing productivity.

The user-friendly interface of "**YieldSmart**", combined with its scalable and modular design, makes it accessible to farmers of all technical backgrounds. Its integration with IoT sensors and weather APIs allows for seamless data collection, ensuring precise predictions without requiring complex manual inputs. The platform's economic feasibility also makes it a cost-effective solution for modern agriculture, helping farmers optimize resource usage while reducing operational costs. By incorporating historical data and real-time monitoring, "**YieldSmart**" ensures continuous improvements in accuracy and performance.

Looking ahead, "**YieldSmart**" has the potential to revolutionize precision farming through continuous innovation. Future enhancements could include more advanced machine learning models, expanded datasets, and AI-driven decision-making tools to further refine yield predictions. Additionally, integrating blockchain for secure data management and collaboration with agricultural organizations could enhance trust and usability. With its commitment to technological advancement, "**YieldSmart**" is set to empower farmers with smarter, more efficient, and sustainable farming practices.

12.APPENDIX

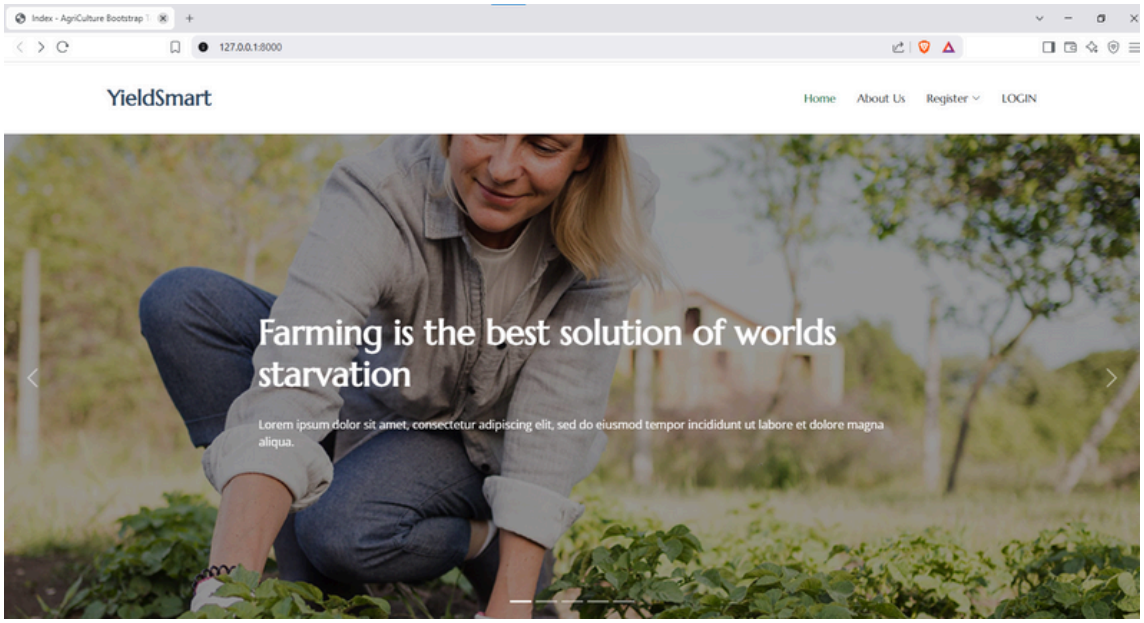


fig 12.1 home page

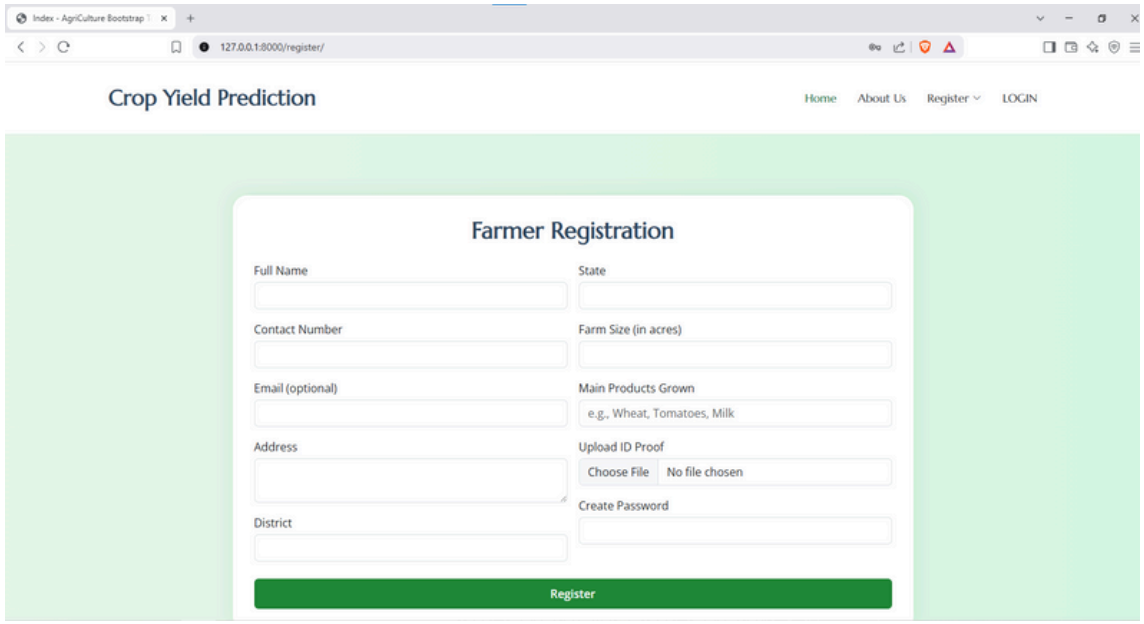


fig 12.2 farmer registration page

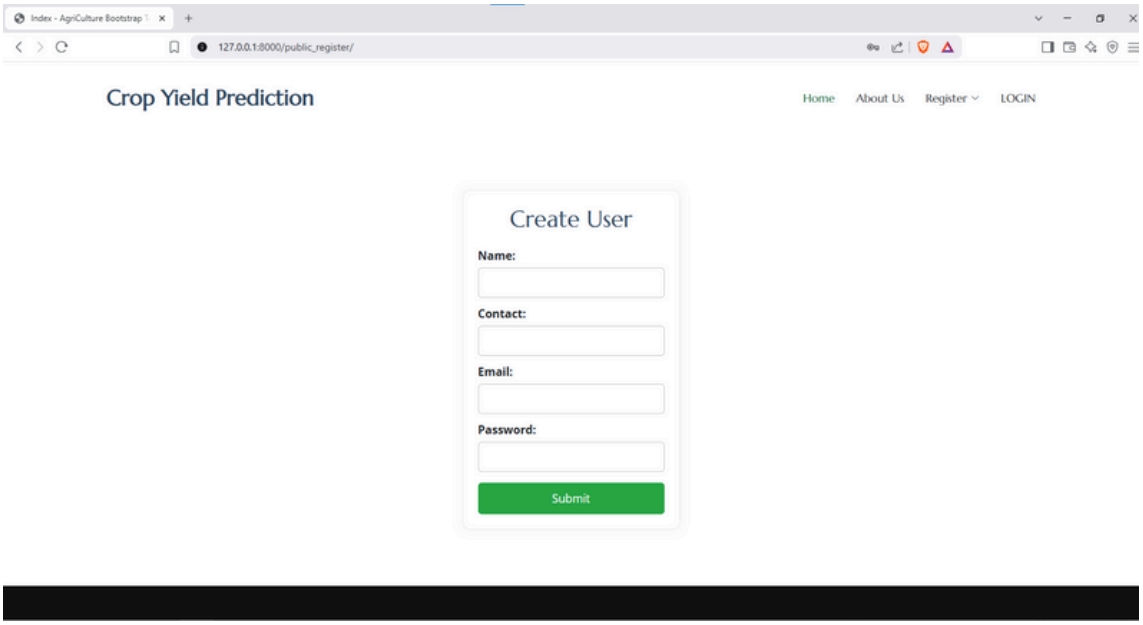


fig 12.3 public registration page

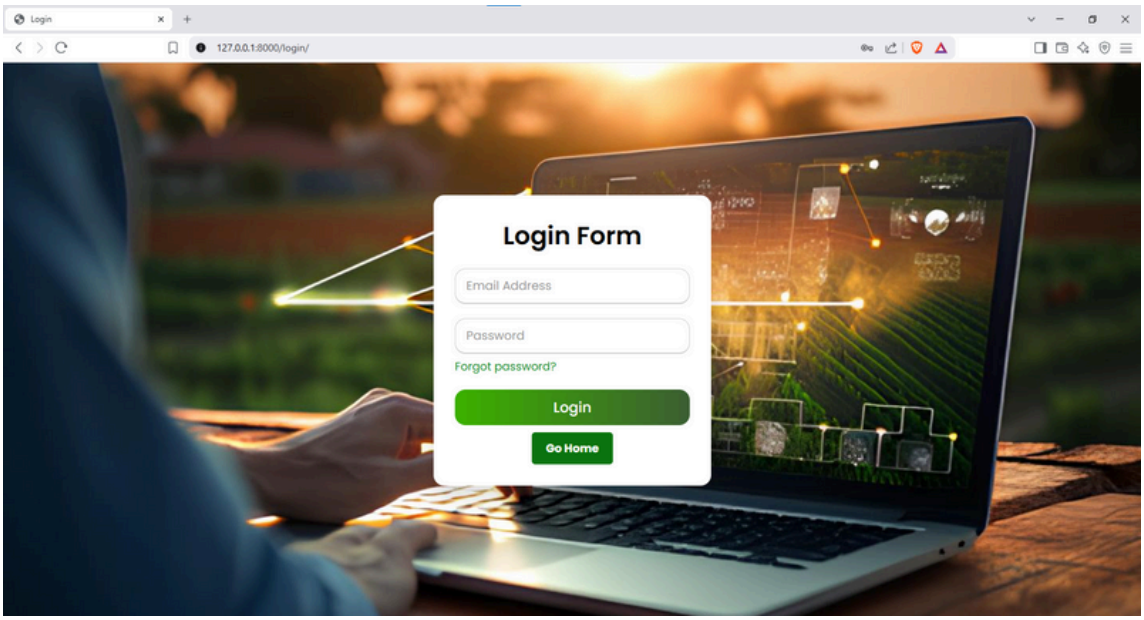


fig 12.4 login page

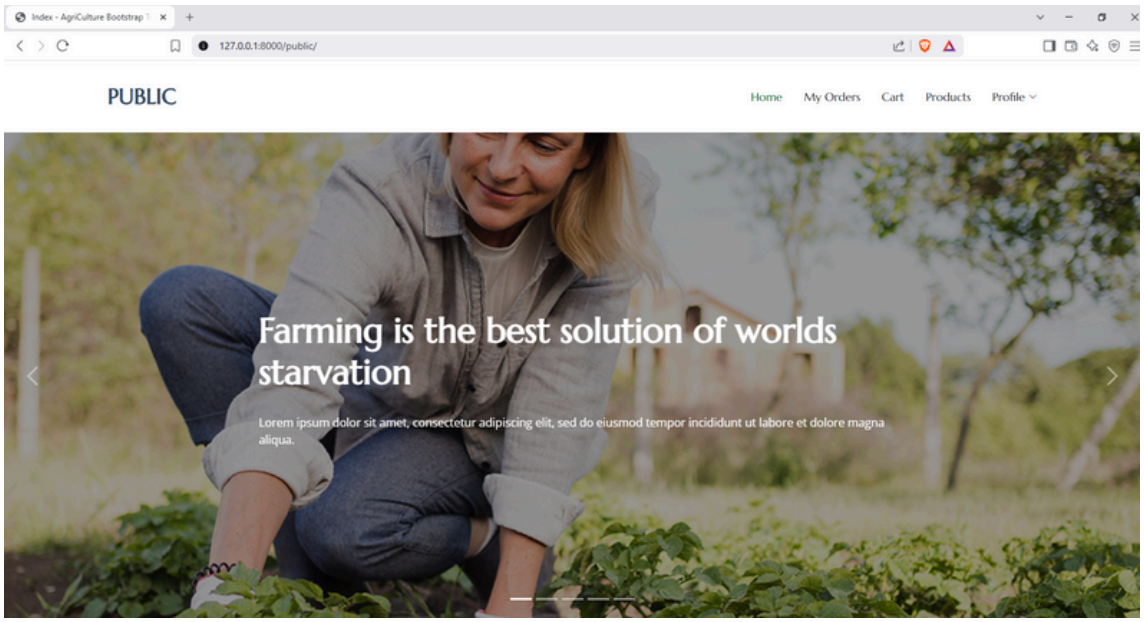


fig 12.5 public home page

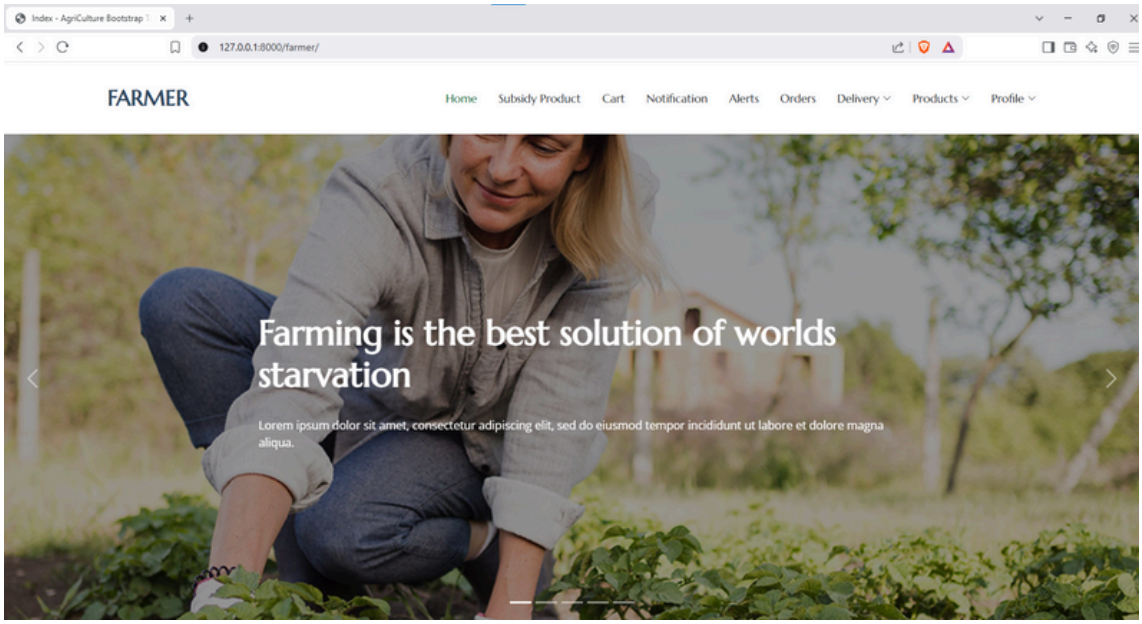


fig 12.6 public home page

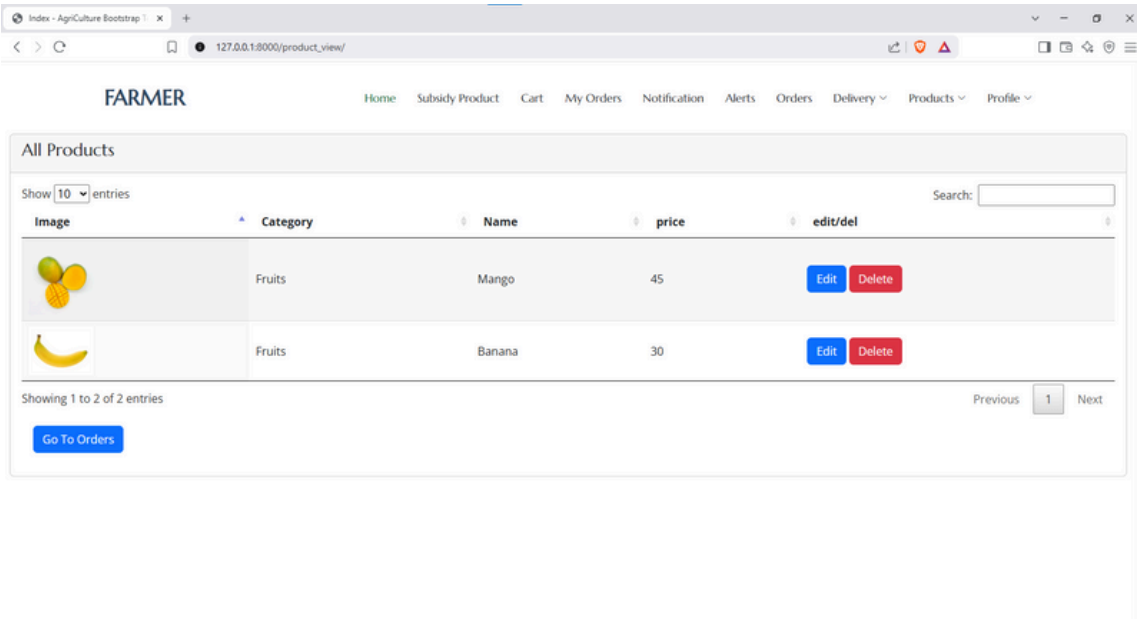


fig 12.7 products page

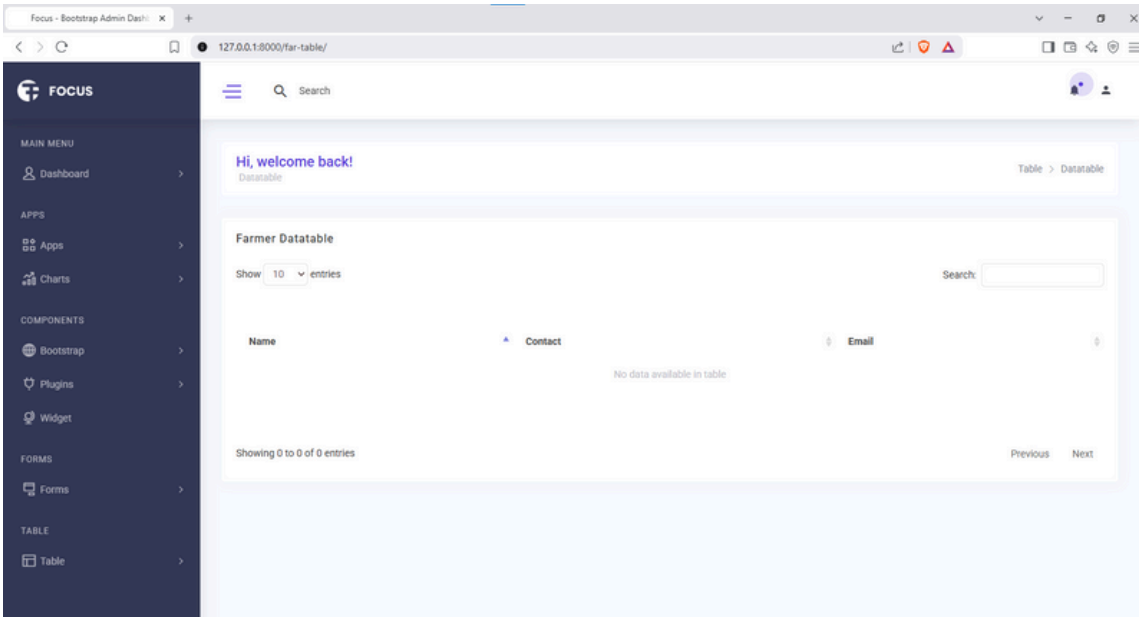


fig 12.8 admin page

13. GANTT CHART

Gantt chart shows time relationship between events of the production program has regarded as revolutionary in management. Gantt chart recognize the total program goals and it should be regarded as a series of inter-related supporting plan (or events), that people can comprehend and follow.

The following figure is the Gantt chart of YIELDSMART. The plan explains the task versus the time will take to complete.

Sl No	Task Name	Duration	Start Date	End Design	December				January				February				March				April			
					W 1	W 2	W 3	W 4	W 1	W 2	W 3	W 4	W 1	W2	W 3	W 4	W1	W2	W 3	W 4	W 1	W2	W 3	W 4
1	Analysis	5	23.12.24	27.12.24																				
2	System Design	28	28.12.25	24.1.25																				
3	Code Design	33	25.1.25	26.2.25																				
4	Testing	20	1.3.25	20.3.25																				
5	Documentation	34	24.3.25	26.4.25																				