

SOFTWARE FOR COMPUTERIZATION OF FARMER LAND DETAILS ALONG WITH BENEFICIARY SCHEME DETAILS

An Internship Project Report Submitted in partial fulfillment of the requirements for the award
of the degree of

BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY

Submitted by

Mr. A. Sai Karthik (21071A1202)

Mr. A. Surya Teja (21071A1203)

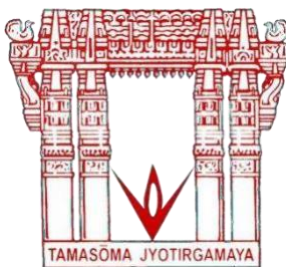
Mr. A.D.S Krishna Teja (21071A1204)

Mr. M. Sai Raj (21071A1239)

Under the guidance of

Dr. B. Jalender

(Associate Professor, Department of IT, VNR VJIET)



DEPARTMENT OF INFORMATION TECHNOLOGY

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

An Autonomous Institute, NAAC Accredited with 'A++' Grade NBA Accredited for CE,
EEE, ME, ECE, CSE, EIE, IT, AME B. Tech Courses Approved by AICTE, New Delhi,
Affiliated to JNTUH Recognized as "College with Potential for Excellence" by UGC
ISO 9001:2015 Certified, QS I GUAGE Diamond Rated

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India

**VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF
ENGINEERING AND TECHNOLOGY**

An Autonomous Institute, NAAC Accredited with 'A++' Grade NBA Accredited for CE,
EEE, ME, ECE, CSE, EIE, IT, AME B, Tech Courses Approved by AICTE, New Delhi,
Affiliated to JNTUH Recognized as "College with Potential for Excellence" by UGC
ISO 9001:2015 Certified, QS I GUAGE Diamond Rated

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India

DEPARTMENT OF INFORMATION TECHNOLOGY



CERTIFICATE

This is to certify that the project report entitled **"SOFTWARE FOR COMPUTERIZATION OF FARMER LAND DETAILS ALONG WITH BENEFICIARY SCHEME DETAILS"** is a bonafide work done under our supervision and is being submitted by Mr. A.Sai Karthik (21071A1202), Mr.A.Surya Teja (21071A1203), Mr. A.D.S.Krishna Teja (21071A1204) Mr. M.Sai Raj (21071A1239) in partial fulfillment for the award of the degree of Bachelor of Technology in Information Tehenology, of the VNRVJIET, Hyderabad during the academic year 2023-2024. Certified further that to the best of our knowledge the work presented in this thesis has not been submitted to any other University or Institute for the award of any Degree or Diploma.


Dr. B. Jalender
Associate Professor,
IT Department,
VNR VJIET


Dr. D Srinivasa Rao
Associate Professor
and HOD,
IT Department,
VNR VJIET

**VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF
ENGINEERING AND TECHNOLOGY**

An Autonomous Institute, NAAC Accredited with 'A++' Grade NBA Accredited for CE,
EEE, ME, ECE, CSE, EIE, IT, AME B. Tech Courses Approved by AICTE, New Delhi.
Affiliated to JNTUH Recognized as "College with Potential for Excellence" by UGC
ISO 9001:2015 Certified, QS I GUAGE Diamond Rated

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India

DEPARTMENT OF INFORMATION TECHNOLOGY



DECLARATION

We declare that the major project work entitled **"SOFTWARE FOR COMPUTERIZATION OF FARMER LAND DETAILS,ALONG WITH BENIFICIARY SCHEME DETAILS"** submitted in the department of Information Technology, Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology, Hyderabad, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Information Technology** is a bonafide record of our own work carried out under the supervision of **Dr B.Jalender, Associate Professor, Department of IT, VNRVJIET**. Also, we declare that the matter embodied in this thesis has not been submitted by us in full or in any part thereof for the award of any degree/diploma of any other institution or university previously.

Place: Hyderabad

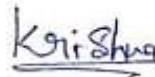

A.Sai Karthik

(21071A1202)



A.Surya Teja

(21071A1203)



A.D.S.Krishna Teja

(21071A1204)



M.Sai Raj

(21071A1239)

ACKNOWLEDGEMENT

Firstly, we would like to express our immense gratitude towards our institution VNR Vignana Jyothi Institute of Engineering and Technology, which created a great platform to attain profound technical skills in the field of Computer Science, thereby fulfilling our most cherished goal.

We are very much thankful to our Principal, **Dr. Challa Dhanunjaya Naidu**, and our Head of Department, **Dr. D Srinivasa Rao**, for extending their cooperation in doing this project within the stipulated time.

We extend our heartfelt thanks to our guide, **Dr. B. Jalender**, and the project coordinators **Manoj Kumar Vemula and Jakkula Sravanthi** for their enthusiastic guidance throughout the course of our project.

Last but not least, our appreciable obligation also goes to all the staff members of the Computer Science & Engineering department and to our fellow classmates who directly or indirectly helped us.

Mr. A. Sai Karthik	(21071A1202)
Mr. A. Surya Teja	(21071A1203)
Mr. A.D.S. Krishna Teja	(21071A1204)
Mr. M. Sai Raj	(21071A1239)

ABSTRACT

The Agricultural Land Computerization Software is a powerful tool designed to revolutionize the management of farmer land details while integrating beneficiary scheme information. It simplifies the process of recording and updating land records, offering farmers an accessible platform to manage their property. Simultaneously, it seamlessly connects with government beneficiary schemes, ensuring eligible farmers can easily access subsidies and incentives.

Furthermore, the IALMS will incorporate an integrated module for beneficiary scheme details.

By interfacing with existing agricultural welfare programs, the system will link eligible farmers with relevant schemes, subsidies, and support mechanisms. This integration will not only enhance the efficiency of government initiatives but also ensure that the benefits reach the intended recipients.

INDEX

1. Introduction	11
2. Literature Survey/ Existing System	12
2.1 Feasibility Study	12
2.1.1 Organizational Feasibility	12
2.1.2 Economic Feasibility	12
2.1.3 Technical Feasibility	12
2.1.4 Behavioral Feasibility	12
2.2 Literature Review	13
2.3 Existing System	14
2.4 Drawbacks Of the Existing System	14
3. Software Requirement Analysis	15
3.1 Introduction	15
3.1.1 Document Purpose	15
3.1.2 Definitions	15
3.2 System Architecture	16
3.3 Functional Requirements	17
3.4 System Analysis	18
3.5 Software Requirement Specification	19
3.6 Software Requirements	19
3.7 Hardware Requirements	19

4. Software Design	20
4.1 UML Diagrams	21
4.1.1 Use Case Diagram	22
4.1.2 Sequence Diagram	26
4.1.3 Activity Diagram	30
5. Proposed System	32
5.1 Methodology	32
5.2 Functionalities	33
5.2.1 Secure login for farmers	33
5.2.2 Evaluating Data	33
5.2.3 Display result	33
5.2.4 Availing	33
5.3 Advantages Of Proposed System	33
6. Coding/Implementation	34
6.1 FrontEnd Implementation	34
7. Testing	35
7.1 Types Of Testing	35
7.1.1 Manual Testing	35
7.1.2 Automated Testing	35
7.2 Software Testing Methods	36
7.2.1 Black Box Testing	36
7.2.2 Gray Box Testing	36
7.2.3 White Box Testing	37
7.3 Testing Levels	38
7.3.1 Non-Functional Testing	38
7.3.1.1 Performance Testing	38

7.3.1.2 Stress Testing	38
7.3.1.3 Security Testing	38
7.3.1.4 Portability Testing	38
7.3.1.5 Usability Testing	39
7.3.2 Functional Testing	39
7.3.2.1 Integration Testing	39
7.3.2.2 Regression Testing	39
7.3.2.3 Unit Testing	39
7.3.2.4 Alpha Testing	40
7.3.2.5 Beta Testing	40
7.4 Test Cases	40
8. Results	41
9. Conclusion And Further Work	43
10. References	43

List of Tables

Table	Page No
Table 4.1.2.1 Sequence diagram Symbols	27
Table. 7.4.1 Test cases	47

List of Figures

Figure	Page No
Fig 3.2.1 System Architecture	16
Fig 3.4.1 Analysis of the system	17
Fig 4.1.1.1 Use Case diagram for the application	24
Fig 4.1.2.1 Sequence diagram for the system	28
Fig 4.1.3.1 Activity Diagram for the system	30
Fig 5.1 Methodology	31
Fig 6.2.1 Libraries used	34
Fig 6.2.2 Preprocessing of data	35
Fig 6.2.3 Naïve bayes classifier	35
Fig 6.2.4 SVM classifier	36
Fig 6.2.5 Gradient Boosting classifier	36
Fig 6.2.6 Logistic Regression	37
Fig 7.2.1.1 Black Box Testing	42
Fig 7.2.1.2 Grey Box Testing	42
Fig 7.2.3.1 White Box Testing	45
Fig 8.1 Comparison of different models	47

Fig 8.2 Pictorial representation of comparison	47
Fig 8.3 User Interface of sentiment analysis	48
Fig 8.4 Result for Positive Sentiment	48
Fig 8.4 Result for Negative Sentiment	48

1. INTRODUCTION

In many countries, especially in rural and agricultural regions, the management of farmer land details and beneficiary schemes has traditionally relied on manual record-keeping systems. These systems are often inefficient, error-prone, and lack transparency, leading to various challenges for both farmers and government authorities. Due to lack of awareness and illiteracy most of the farmers are not being able to utilize and claim benefits from their lands and crop. This Software acts as interface and helps farmers and their families providing the knowledge about the benefits and scheme they could claim from the government for their respective land and the type of crop being grown. This software initially consists of a form to be filled by the farmers to provide the necessary details and based upon it the calculations are done and farmer gets a clear picture of the available beneficiary schemes that could be availed.

2. LITERATURE SURVEY/ EXISTING SYSTEM

2.1 FEASIBILITY STUDY

The feasibility study for the computerization of farmer land details and beneficiary schemes establishes a strong foundation for project viability. From a technical perspective, the assessment indicates a seamless integration with existing technology and infrastructure, ensuring a smooth implementation process. Financially, the study justifies the initial investment by highlighting the anticipated long-term benefits and cost-effectiveness of streamlined data management.

2.1.1 ORGANIZATIONAL FEASIBILITY

The organizational feasibility study for the computerization of farmer land details and beneficiary schemes indicates a favorable environment for project implementation. The existing organizational structures and processes align well with the proposed system, ensuring a smooth integration into daily operations. The study recognizes the adaptability of the staff to the new technology, minimizing potential resistance to change.

2.1.2 ECONOMIC FEASIBILITY

The economic feasibility study for the computerization of farmer land details and beneficiary schemes demonstrates a sound financial rationale. The initial investment in hardware, software, training, and implementation is justified by the projected long-term cost savings and efficiency gains. The system's potential to enhance productivity and reduce manual efforts contributes to a positive return on investment.

2.1.3 TECHNICAL FEASIBILITY

The technical feasibility study for the computerization of farmer land details and beneficiary schemes indicates a robust compatibility with existing technology and infrastructure. The project aligns seamlessly with current systems, leveraging available hardware and software resources. It is designed to iterate smoothly into established workflows, minimizing disruptions during implementation.

2.1.4 BEHAVIORAL FEASIBILITY

The behavioral feasibility study provides insights into the human and organizational factors influencing the successful implementation. Decision-makers can use these findings to tailor strategies that enhance user acceptance, manage change effectively.

2.2 LITERATURE REVIEW

In Literature survey, we have gone through other similar works that are implemented in Software for computerization of farmer land details along with beneficiary scheme details.

Benefits of Agricultural Software Adoption:

Various studies highlight the potential benefits of adopting digital solutions in agriculture, emphasizing increased efficiency, improved decision-making, and enhanced productivity and ease in accessing the website.

Government Initiatives and Beneficiary Schemes:

Government-led initiatives to digitize land records and streamline beneficiary schemes are discussed, emphasizing the importance of effective implementation and stakeholder Collaboration.

Financial Inclusion and Agriculture Finance:

Literature highlights the potential for agricultural software to contribute to financial Inclusion by providing farmers with access to loans, subsidies, and other financial resources.

Challenges in Agricultural Technology Adoption:

Research acknowledges challenges in technology adoption within the agricultural sector, including issues related to digital literacy, resistance to change, and inadequate infrastructure

2.2 EXISTING SYSTEM

The existing system of availing benefits from the farmer land details involves lots of complication and will hardly be able to be filled or understood by the customer, in order to make it simpler in terms of usage and interface we have built this web page in order for the farmer to easily fill details of his land and get appropriate and available schemes out of all the released schemes related to farmers across the country can be easily accessed.

2.3 DRAWBACKS OF THE EXISTING SYSTEM

- Farmers in rural areas may lack access to the necessary technology or have limited digital literacy, creating a digital divide. This can hinder their ability to take advantage of online platforms, access information, and benefit from government schemes.
- The accuracy and completeness of data depend on the farmers' ability to input information correctly. Errors in data entry or incomplete information can lead to inaccuracies, affecting the effectiveness of land records and beneficiary schemes.

3. SOFTWARE REQUIREMENT ANALYSIS

3.1 INTRODUCTION

Following elicitation, requirement analysis is an important and necessary process. To create uniform and unambiguous requirements, we examine, improve, and scrutinize the obtained requirements. This exercise goes over all of the requirements and may show a graphical representation of the full system.

3.1.1 DOCUMENT PURPOSE

To clearly define the purpose and scope of implementing a webpage providing farmers ease of availing the schemes they are benefitted by.

3.1.2 DEFINITIONS

phpMyAdmin

phpMyAdmin is a GUI-based application which is used to manage MySQL database.

User Centric Design

User-centric design is an approach that prioritizes the needs and preferences of end-users throughout the design and development process.

Automation

Automation reduces manual efforts, minimizes errors, and ensures schemes.

3.2 SYSTEM ARCHITECTURE

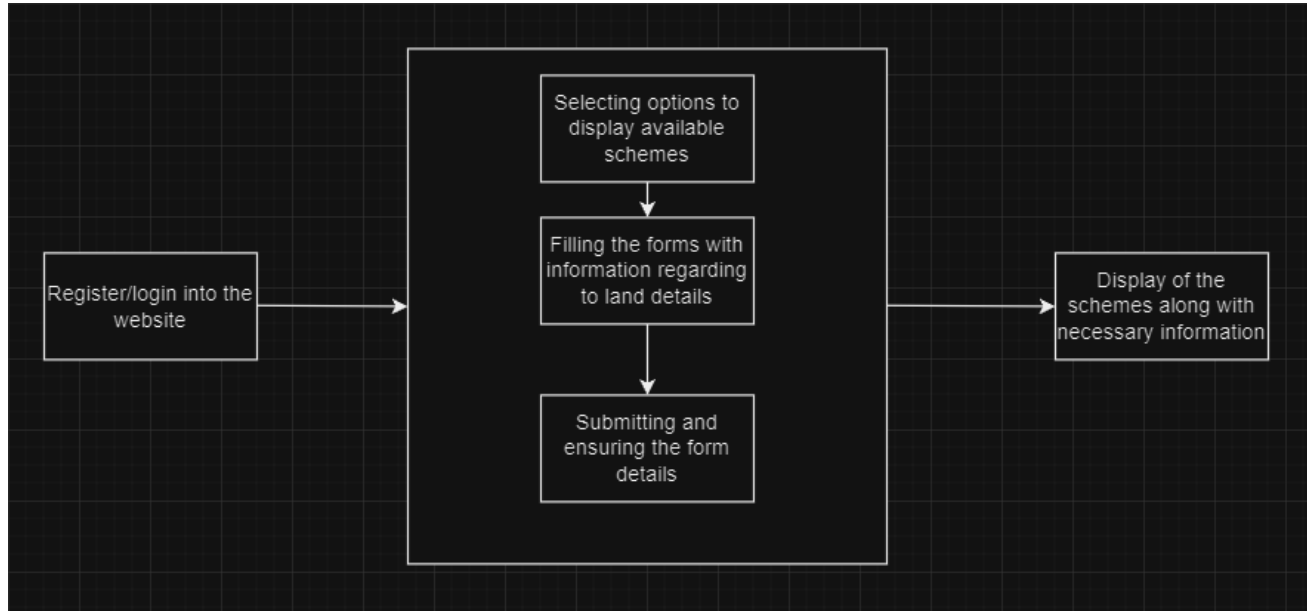


Fig 3.2.1: System Architecture

It is a conceptual model that describes the structure, viewpoints, and behavior of a system. A formal description and representation of a system that is organized in a way that facilitates reasoning about the system's structures and behaviors is known as an architecture description. A system architecture consists of system components and created sub-systems that work together to implement the overall system. The proposed System, System architecture starts with creating accounts for farmer . The system stores the data regarding the credentials in the database.

3.3 FUNCTIONAL REQUIREMENTS

- User Registration and Authentication:
 - Farmers should be able to create user accounts.
 - The system must authenticate users securely.
- Profile Management:
 - Farmers should be able to edit and update their personal information.
 - Options to reset passwords securely.
- Land Details Input:
 - Farmers must have a user-friendly interface to input and update their land details.
 - The system should validate and store land-related information, such as size, location, and crops grown.
- Beneficiary Scheme Matching:
 - The system must match farmer profiles and land details with eligible government schemes.
 - Regularly updated database of available schemes and their criteria.
- Dashboard:

Farmers should have a personalized dashboard displaying their registered information, land details, and relevant notifications

3.4 SYSTEM ANALYSIS

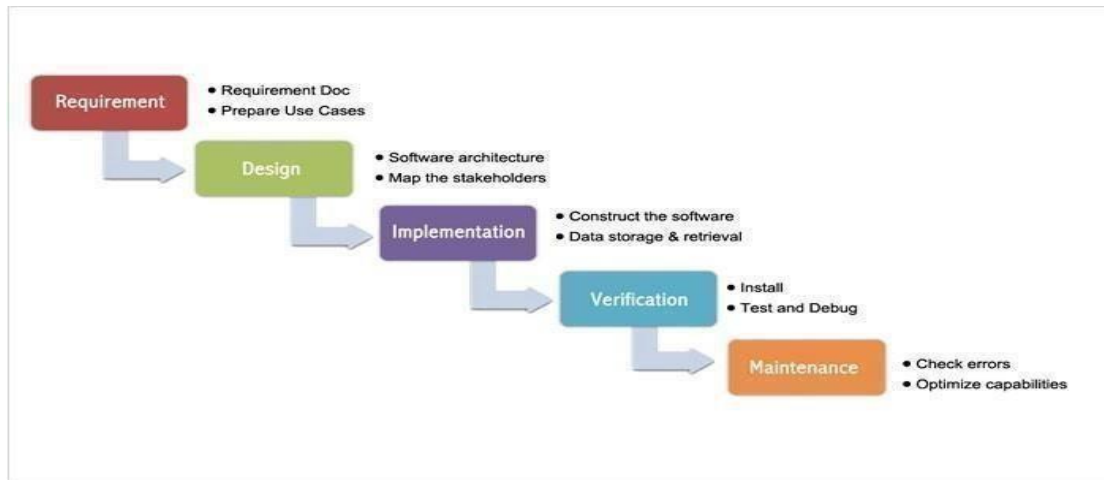


Fig:3.4.1 Analysis of the system

It was the first Process Model to be introduced. A linear sequential life cycle model is another name for it. It's quite simple to use and understand. Phases do not overlap in this paradigm, and each phase must be finished before the next one begins. The first SDLC the approach used during software development was the waterfall model.

The model shows that the development of software is linear and is a sequential process. Only after one phase of the development is completed, we can go to the next phase. In this waterfall paradigm, the phases do not overlap.

The steps in the waterfall model are explained below.

Requirements: The search has become more intense and concentrated on the software's requirements at this time. To comprehend the nature of the programs to be developed, the software engineer must first comprehend the software's information domain, which includes the required functionalities, user interface, and so on. The customer must be informed about the second activity, which must be recorded and presented.

Design: This step is used to transform the above criteria as a representation in the form of "blueprint" software before coding begins. The design must be able to meet the criteria laid out in the previous stage.

Implementation: The design was converted into a machine-readable format in order for it to be interpreted by a computer in some circumstances, i.e., through the coding process into a programming language. This was the stage in which the programmer will put the technical design phase into action.

Verification: It, like anything else constructed, must first be put to the test. The same may be said for software. To ensure that the application is error-free, all functions must be checked, and the results must closely comply to the previously specified requirements.

Maintenance: Software maintenance, including development, is essential since the software that is being generated is not always exactly like that. It may still have minor faults that

were not identified previously when it runs, or it may require additional capabilities that were not previously available in the software.

Useful factors: The Waterfall model has its advantages like it is simple to use. Additionally, while using the model all the system requirements can be defined as a whole, explicitly and at the start the product can run without many issues.

It is economic to make changes in the early stages of the project when there are problems with system requirements then when the problems which arise in later stages.

3.5 SOFTWARE REQUIREMENT SPECIFICATIONS

HTML/CSS

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analyzing data from social network websites.

JavaScript:

It is an open-source artificial intelligence package that builds models using data flow of the graphs. Enables developers to build large-scale neural networks with several layers.

TensorFlow is mostly used for classification, perception, comprehension, discovery, prediction, and creation.

php:

It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. It is called Bayes because it depends on the principle of Bayes' Theorem. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

3.6 SOFTWARE REQUIREMENTS

- Operating System : Windows, mac, Linux
- Frontend : HTML, CSS, JavaScript
- Backend : PHP
- Database : PhpMyAdmin

3.7 HARDWARE REQUIREMENTS

- Minimum 8GB Ram Laptop, minimum 10GB HDD
- Camera: Good quality, 3MP
- GPU:4GB dedicated
- Processor: Intel Pentium 4 or higher
- Mouse: Scroll or Optical mouse or Touchpad
- Internet Connection

4. SOFTWARE DESIGN

4.1 UML DIAGRAMS

The Device Architecture Manual describes the application requirements, operating state, application and subsystem functionality, documents and repository setup, input locations, yield types, human-machine interfaces, management reasoning, and external interfaces. The Unified Modeling Language (UML) assists software developers in expressing an analysis model through documents that contain a plethora of syntactic and semantic instructions. A UML context is defined as five distinct viewpoints that present the system from a particularly different point of view.

The components are similar to modules that can be combined in a variety of ways to create a complete UML diagram. As a result, comprehension of the various diagrams is essential for utilizing the knowledge in real-world systems. The best method to understand any complex system is to draw diagrams or images of it. These designs have a bigger influence on our understanding. Looking around, we can see that info-graphics are not a new concept, but they are frequently utilized in a variety of businesses in various ways.

User Model View

The perspective refers to the system from the clients' point of view. The exam's depiction depicts a situation of utilization from the perspective of end-clients. The user view provides a window into the system from the perspective of the user, with the system's operation defined in light of the user and what the user wants from it.

Structural model view

This layout represents the details and functionality of the device. This software design maps out the static structures. This view includes activity diagrams, sequence diagrams and state machine diagrams

Behavioral Model View

It refers to the social dynamics as framework components, delineating the assortment cooperation between various auxiliary components depicted in the client model and basic model view. UML Behavioral Diagrams illustrate time-dependent aspects of a system and communicate the system's dynamics and how they interact. Behavioral diagrams include interaction diagrams, use case diagrams, activity diagrams and state-chart diagrams.

Implementation Model View

The essential and actions as frame pieces are discussed in this when they are to be manufactured. This is also referred to as the implementation view. It uses the UML Component diagram to describe system components. One of the UML diagrams used to illustrate the development view is the Package diagram.

Environmental Model View

The systemic and functional component of the world where the program is to be introduced was expressed within this. The diagram in the environmental view explains the software model's after-deployment behavior. This diagram typically explains user interactions and the effects of software on the system. The following diagrams are included in the environmental model: Diagram of deployment.

The UML model is made up of two separate domains:

- Demonstration of UML Analysis, with a focus on the client model and auxiliary model perspectives on the framework.
- UML configuration presenting, which focuses on demonstrations, usage, and natural model perspectives.

4.1.1 USE CASE DIAGRAM

The objective of a use case diagram is to portray the dynamic nature of a system. However, because the aim of the other four pictures is the same, this description is too broad to characterize the purpose. We'll look into a specific purpose that distinguishes it from the other four diagrams.

The needs of a system, including various factors, are collected using use case diagrams. The majority of these specifications are design specifications. As a result, use cases are constructed and actors are identified when examining a system to gather its functions.

Use case diagrams are made to represent the outside view once the primary task is completed.

In conclusion, use case diagrams are useful for the following purposes:

- Used to collect a system's requirements.
- Used to get a bird's-eye view of a system.
- Determine various factors that are influencing the system.
- Display the interaction of the requirements as actors.

Use case diagrams are used to analyze a system's high-level requirements. The functionality of a system is recorded in use cases when the requirements are examined. Use cases can be defined as "system functionalities written in a logical order." The actors are the second pillar of use cases that is important. Any entities that interact with the system are referred to as actors.

Internal applications, human users and external applications can all be actors.

The following factors should be kept in mind when constructing a use case diagram.

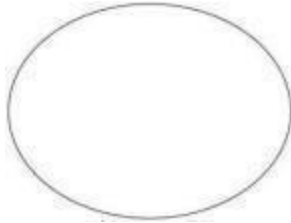
- As a use case, functionalities will be represented.
- Actors.
- Relationships between use cases and actors.

Use Cases

A use case is a written depiction of how visitors will execute tasks on your website. From the standpoint of a user, it defines how a system responds to a request. Each use case is characterized by a sequence of basic actions that start with the user's goal and finish when that goal is achieved.

Graphical Representation

Use cases are represented by an oval shape.



The following is a more precise analysis of a use case:

- A pattern of behavior displayed by the system.
- A series of related transactions performed by an actor as well as the system.
- Delivering something useful to the actor.

You can utilize use cases to document system requirements, connect with top users and domain experts, and test the system. Looking at the actors and defining what they can accomplish with the system is the greatest way to uncover use cases.

Flow of events

A sequence of times can be thought of as a collection of interactions (or opportunities) carried out by the system. They provide daily point-by-point details, published in terms of what the framework can do rather than whether the framework performs the task.

- When and how the employment case begins and ends.
- Interactions between the use case and the actor.
- Information required by the employment case.
- The employment case's normal sequence of events.
- A different or exceptional flow.

Construction of Use case

The behavior of the framework is graphically illustrated in use-case outlines. These graphs show how the framework is utilized at a high level, when seen through the perspective of an untouchable (actor). A utilization case graph can depict all or some of a framework's work instances.

A use-case diagram may include the following elements:

- Actors.
- Use cases.

Relationships in use cases

Active relationships, also known as behavioral relationships, are a type of interaction that is frequently shown in use case diagrams. The four main types of behavioral relationships are inclusion, communication, generalization and extension.

1) Communicates

The behavioral relationship communicates connects an actor to a use case. Remember that the purpose of the use case is to provide some sort of benefit to the system's actor. As a result, it is critical to document these interactions between actors and use cases. A line with no arrowheads connects an actor to a use case.

2) Includes

The includes relationship (also known as the uses relationship) describes a situation in which a use case contains behavior that is shared by multiple use cases. To put it another way, the common use case is included in the other use cases. The included relationship is indicated by a dotted arrow pointing to the common use case.

3) Extends

The extended connection describes when one use case contains behavior that allows a new use case to handle a variant or exception to the basic use case. A distinct use case handles exceptions to the basic use case. The arrow connects the basic and extended use cases.

4) Generalizes

The generalized relationship indicates that one thing is more prevalent than another. This link could be between two actors or between two use cases. The arrow points to a "thing" in UML that is more general than another "thing."

In this system Use Case diagram:

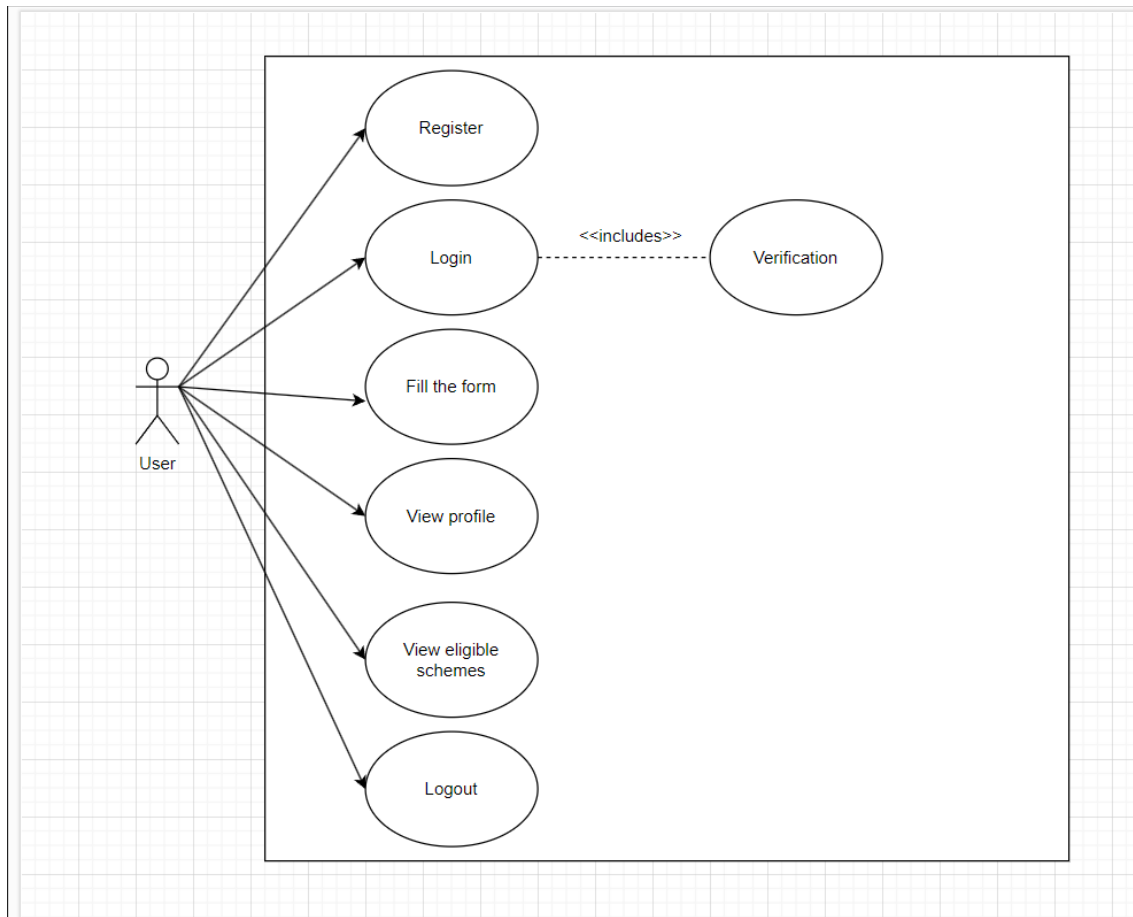


Fig 4.1.1.1: Use Case diagram for the application

Actors:

- Admin
- User

Use Cases:

- Gather data from user
- Processing the statement
- Naïve Bayes Classifier algorithm applied to predict the result
- Display of result

Connections:

- Admin must gather data required to choose which algorithm to be used
- Then the admin will be processing the data and choose the algorithm based on accuracy
- User will be using the efficient model for prediction on the data

4.1.2 SEQUENCE DIAGRAM

Because it illustrates how a group of items interact with one another, a sequence diagram is a form of interaction diagram. These diagrams are used by software engineers and business people to comprehend the requirements for a new system or to document a current process. Sequence diagrams are sometimes known as event diagrams or event scenarios. Sequence diagrams can be useful as a reference for businesses and other organizations. Make the diagram to show:

- Describe the specifics of a UML use case.
- Create a model of the logic of a complex procedure, function, or operation.
- Examine how objects and components interact with one another in order to complete a process.
- Plan and comprehend the specific functionality of a current or future scenario. The following scenarios lend themselves well to the use of a sequence diagram:

A usage scenario is a diagram that shows how your technology might be utilized in the future. It's an excellent approach to make sure you've thought through every possible system usage situation.

Method logic:

A UML sequence diagram can be used to study the logic of any function, method, or complex process, just as it can be used to examine the rationale of a use case. If you view a service to be a high-level method utilized by several customers, a sequence diagram is a fantastic approach to map out service logic.

Object:

An object has a state, a lead, and a personality. The structure and direction of objects that are, for all intents and purposes, indistinguishable are depicted in their fundamental class. Each object in a diagram represents a specific instance of a class. An order case is an object that is not named.

Message:

A message is the exchange of information between two articles that causes an event to occur. A message transmits information from the source point of control convergence to the objective point of control convergence.

Link:

An existing association between two objects, including class, implying that there is an association between their opposing classes. If an object associates with itself, use the image's hover adjustment.

Lifeline:

It reflects the passage of time as it goes downward. The events that occur consecutively to an object during the monitored process are depicted by this dashed vertical line. A designated rectangle shape or an actor symbol could be the starting point for a lifeline.

Actor:

Entities that interact with the system or are external to it are shown.

Synchronous message:

This is represented by a solid line with a solid arrowhead. This symbol is used when a sender must wait for a response to queries before proceeding. Both the call and the response should be depicted in the diagram.

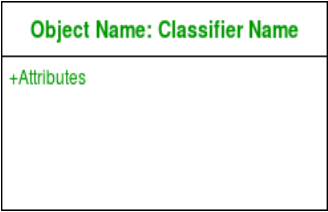


Asynchronous message:

A solid line with a lined arrowhead is used to represent this. Asynchronous messages do not necessitate a response before the sender can proceed. The diagram should only include the call.

Delete message:

An X follows a solid line with a solid arrowhead. This message has the effect of causing an object to be destroyed.

Sequence diagram components:

Name	Description	Symbol
Object symbol	Represents a class or object in UML. The object symbol demonstrates how an object will behave in the context of the system. Class attributes should not be listed in this shape.	
The activation box	Represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes.	
Actor symbol	Shows entities that interact with or are external to the system.	




Lifeline symbol	Represents the passage of time as it extends downward. This dashed vertical line shows the sequential events that occur to an object during the charted process. Lifelines may begin with a labelled rectangle shape or an actor symbol.	
Alternative symbol	Symbolises a choice (that is usually mutually exclusive) between two or more message sequences. To represent alternatives, use the labelled rectangle shape with a dashed line inside.	
Message symbol	This symbol is used when a sender needs to send a message.	

Table 4.1.2.1: Sequence Diagrams Symbols

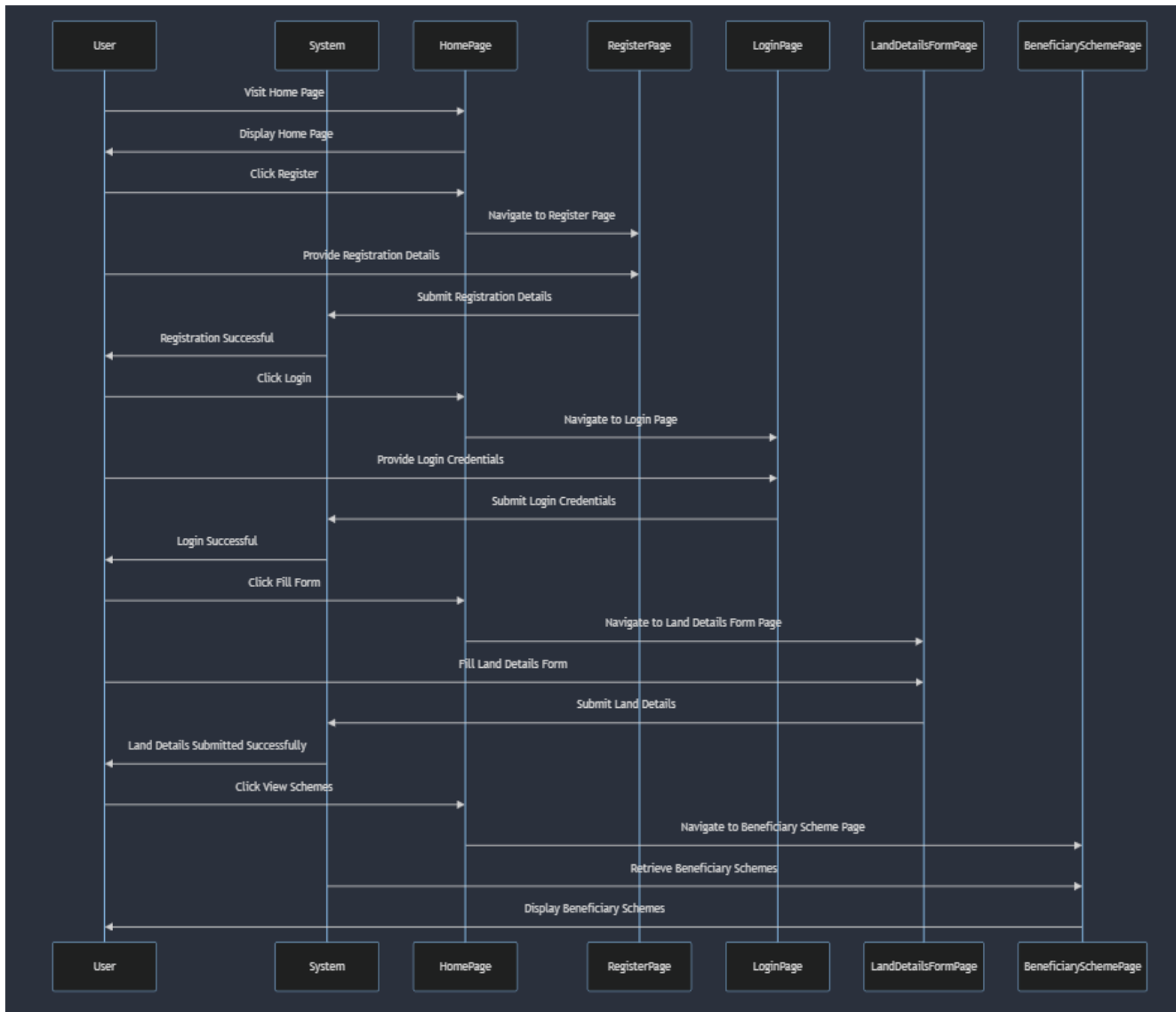


Fig 4.1.2.1: Sequence Diagram for the application

4.1.3 ACTIVITY DIAGRAM

An activity diagram is a flowchart that displays the movement of information from one action to the next. A system operation can be used to describe the activity.

From one operation to the next, the control flow is guided. In nature, this flow might be sequential, branching, or concurrent. By employing numerous parts such as join, fork and so on, activity diagrams cope with all sorts of flow control.

Activity diagrams provide the same basic functions as the other four diagrams. It captures the dynamic behavior of the system. The other four diagrams depict message flow from one item to the next, whereas the activity diagram depicts a specific system operation is referred to as an activity. It doesn't show any communication flow from one activity to the next. The phrases activity diagrams and flowcharts are often used interchangeably. Although the diagrams resemble flowcharts, they are not.

Notations

Initial point or start point

A small, filled circle, followed by an arrow, represents the beginning action state or starting point for any activity diagram. Make sure the start point of an activity diagram with swimlanes is in the top left corner of the first column.

Activity or Action state

An action state is a representation of an object's non-interruptible action. You can make an action state in SmartDraw by sketching a rectangle with rounded corners.

Action flow

Transitions from one action state to another are depicted by action flows, also known as edges and routes. An arrowed line is commonly used to depict them.

Decisions and branching

A diamond signifies a multiple-choice decision. Place a diamond between the two activities when one requires a decision before moving on to the next. A condition or guard expression should be used to label the outgoing alternates. One of the paths can also be labeled "else."



Fig 4.1.3.1: Activity Diagram for the system

This activity diagram shows the whole activity of the system. The Activity starts with the client submitting the data required for logging in and entering his land details. Results are then displayed.

5. PROPOSED SYSTEM

5.1 METHODOLOGY

Project Planning and Requirements Analysis:

- Conducted a thorough analysis of the other websites which provide benefits to farmers in letting them know about their schemes they are benefitted by.
- Defined the scope, objectives, and functionalities of the project.

System Architecture Design:

- Designed the overall architecture of the system, including the database schema, user interfaces, and interactions between different input forms.
- Selected appropriate technologies and frameworks to meet project requirements.

Frontend and Backend Development:

- Implemented the frontend components for the welcome page, home page, forms page, result page, profile page and available schemes page.
- Developed the backend logic using [Backend Language/Framework], ensuring proper data flow and authentication mechanisms.

Database Design and Implementation:

- Designed a database to store information about farmers details such as username and password.
- Implemented the database using [Database Management System].

User Authentication and Authorization:

- Implemented secure user authentication and authorization mechanisms for the farmer [Authentication Framework].
- Ensured that sensitive information is protected and access is granted based on user roles.

5.2 FUNCTIONALITIES

5.2.1 Secure login for the farmers:

- Secure login through authentication mechanisms.

5.2.2 Evaluating Data

- Based upon the farmers input data the website evaluates the schemes the farmer can avail based upon his state, land area and type of crop he is growing.

5.2.3 Displaying result:

- Result is then displayed based on the given inputs.

5.2.4 Availing:

- Furthermore, proper links and information is being provided so that the farmers can directly access the schemes website provided by the government.

5.3 ADVANTAGES OF PROPOSED SYSTEM:

- It helped us in analyzing and summing up in terms of creating a website in which there is an ease to farmer access to it and properly entering details and getting the result in no time.
- It also helped us know the lagging and complex features of the current websites.

6. CODING AND IMPLEMENTATIONS

6.1 Frontend Implementation:

Registration Page :

- The registration page allows farmers to create an account.
This page includes a form with fields for the username and password.
- When the farmer submits the form, the data is sent to the server for registration.

Login Page :

- Farmers use their registered username and password to log in.
- Upon successful login, the system stores the farmer's session information.

Scheme Selection Page:

- After logging in, farmers are directed to a page where they can select the schemes they want to avail. This page may display a list of available schemes with checkboxes for selection.
- On submission, the selected schemes are stored in the session or sent to the server for processing.

Details Entry Pages :

- The farmer is directed to three consecutive pages where they enter personal and land details.
- Each page has a form with relevant fields (e.g., personal details, land size, soil type).
- The entered data is stored in the session or sent to the server.

Results Page :

- After filling in the details, the farmer is redirected to the results page.
- The backend processes the data from the previous pages to determine the schemes the farmer is eligible for.
- The results, i.e., the list of benefited schemes, are displayed on this page.

Database Integration:

- Use a database to store user credentials, academic data, complaints, messages.

7. TESTING

In machine learning, testing is mainly used to validate raw data and check the ML model's performance. The main objectives of testing machine learning models are:

- Quality Assurance
- Detect bugs and flaws

Once your machine learning model is built (with your training data), you need unseen data test your model. This data is called testing data, and you can use it to evaluate the performance and progress of your algorithms' training and adjust or optimize it for improved results.

Testing data has two main criteria. It should:

- Represent the actual dataset
- Be large enough to generate meaningful predictions

7.1 TYPES OF TESTING

7.1.1 MANUAL TESTING

Manual Testing is a type of software testing in which test cases are executed manually by a tester without using any automated tools. The purpose of Manual Testing is to identify the bugs, issues, and defects in the software application. Manual software testing is the most primitive technique of all testing types and it helps to find critical bugs in the software application.

Any new application must be manually tested before its testing can be automated. Manual Software Testing requires more effort but is necessary to check automation feasibility. Manual Testing concepts does not require knowledge of any testing tool. One of the Software Testing Fundamental is “**100% Automation is not possible**“. This makes Manual Testing imperative.

7.1.2 AUTOMATED TESTING

Automation Testing is a software testing technique that performs using special automated testing software tools to execute a test case suite. On the contrary, Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps.

The automation testing software can also enter test data into the System Under Test, compare expected and actual results and generate detailed test reports. Software Test Automation demands considerable investments of money and resources.

7.2 SOFTWARE TESTING METHODS

7.2.1 BLACK BOX TESTING

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer. In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

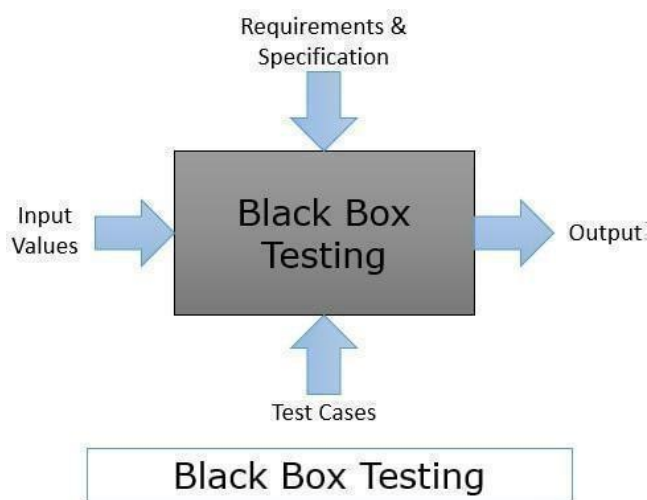


Fig. 7.2.1.1 Black Box Testing

7.2.2 GRAY BOX TESTING

Gray box testing is a software testing method to test the software application with partial knowledge of the internal working structure. It is a **combination of black box and white box testing** because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing.

Gray box testing commonly identifies context-specific errors that belong to web systems. For example; while testing, if tester encounters any defect then he makes changes in code to resolve the defect and then test it again in real time. It concentrates on all the layers of any complex software system to increase testing coverage. It gives the ability to test both presentation layer as well as internal coding structure. It is primarily used in integration testing and penetration testing.

This testing technique is a combination of Black box testing and White box testing. In Black box testing, the tester does not have any knowledge about the code. They have information for what will be the output for the given input. In White box testing, the tester has complete knowledge about the code. Grey box testers have knowledge of the code, but not completely.



Fig. 7.2.2.1 Grey Box Testing

7.2.3 WHITE BOX TESTING

White box testing is an approach that allows testers to inspect and verify the inner workings of a software system—its code, infrastructure, and integrations with external systems. White box testing is an essential part of automated build processes in a modern Continuous Integration/Continuous Delivery (CI/CD) development pipeline. White box testing is often referenced in the context of Static Application Security Testing (SAST), an approach that checks source code or binaries automatically and provides feedback on bugs and possible vulnerabilities. White box testing is a testing technique, that examines the program structure and derives test data from the program logic/code. The other names of glass box testing are clear box testing, open box testing, logic driven testing or path driven testing or structural testing.

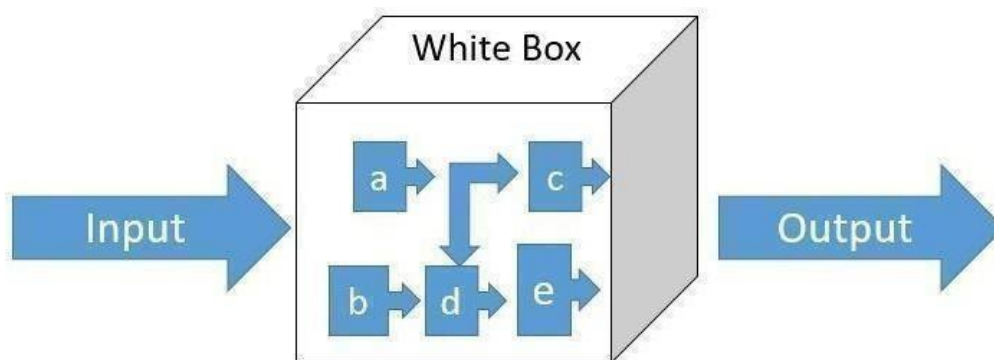


Fig. 7.2.3.1 White Box Testing

7.3 TESTING LEVELS

7.3.1 NON-FUNCTIONAL TESTING

Non-functional testing is a type of software testing to test non-functional parameters such as reliability, load test, performance and accountability of the software. The primary purpose of non-functional testing is to test the reading speed of the software system as per non-functional parameters. The parameters of non-functional testing are never tested before the functional testing. Non-functional testing is also very important as functional testing because it plays a crucial role in customer satisfaction.

7.3.1.1 PERFORMANCE TESTING

Performance testing is a form of software testing that focuses on how a system running the system performs under a particular load. This is not about finding software bugs or defects. Different performance testing types measure according to benchmarks and standards. Performance testing gives developers the diagnostic information they need to eliminate bottlenecks.

7.3.1.2 STRESS TESTING

Stress Testing is a type of software testing that verifies stability & reliability of software application. The goal of Stress testing is measuring software on its robustness and error handling capabilities under extremely heavy load conditions and ensuring that software doesn't crash under crunch situations. It even tests beyond normal operating points and evaluates how software works under extreme conditions.

7.3.1.3 SECURITY TESTING

Security Testing is a type of Software Testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders. It ensures that the software system and application are free from any threats or risks that can cause a loss. Security testing of any system is focused on finding all possible loopholes and weaknesses of the system which might result in the loss of information or reputation of the organization.

7.3.1.4 PORTABILITY TESTING

Portability Testing is one of Software Testing which is carried out to determine the degree of ease or difficulty to which a software application can be effectively and efficiently transferred from one hardware, software or environment to another one. The results of portability testing are measurements of how easily the software component or application will be integrated into the environment and then these results will be compared to the non-functional requirement of portability of the software system.

7.3.1.5 USABILITY TESTING

Usability Testing, also known as User Experience (UX) Testing, is a testing method for measuring how easy and user-friendly a software application is. A small set of target end-users, use software applications to expose usability defects. Usability testing mainly focuses on the user's ease of using application, flexibility of application to handle controls and ability of application to meet its objectives. This testing is recommended during the initial design phase of SDLC, which gives more visibility on the expectations of the users.

7.3.2 FUNCTIONAL TESTING

It is a type of software testing which is used to verify the functionality of the software application, whether the function is working according to the requirement specification. In functional testing, each function is tested by giving the value, determining the output, and verifying the actual output with the expected value. Functional testing performed as black-box testing which is presented to confirm that the functionality of an application or system behaves as we are expecting. It is done to verify the functionality of the application. Functional testing is also called black-box testing.

7.3.2.1 INTEGRATION TESTING

Integration testing is done to test the modules/components when integrated to verify that they work as expected i.e. to test the modules which are working fine individually and do not have issues when integrated. The main function or goal of this testing is to test the interfaces between the units/modules. The individual modules are first tested in isolation. Once the modules are unit tested, they are integrated one by one, till all the modules are integrated, to check the combinational behavior, and validate whether the requirements are implemented correctly.

7.3.2.2 REGRESSION TESTING

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. Regression Testing is nothing but a full or partial selection of already executed test cases that are re-executed to ensure existing functionalities work fine. This testing is done to ensure that new code changes do not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

7.3.2.3 UNIT TESTING

Unit Testing is a software testing technique by means of which individual units of software. i.e. a group of computer program modules, usage procedures, and operating procedures are tested to determine whether they are suitable for use or not. Unit Testing is defined as a type of software testing where individual components of a software are tested. Unit Testing of the software product is carried out during the development of an application. An individual component may be either an individual function or a procedure.

7.3.2.4 ALPHA TESTING

Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the **user acceptance testing**. This is referred to as alpha testing only because it is done early on, near the end of the development of the software. Alpha testing is commonly performed by homestead software engineers or quality assurance staff. It is the last testing stage before the software is released into the real world.

7.3.2.5 BETA TESTING

Beta Testing is performed by real users of the software application in a real environment. Beta testing is one of the types of **User Acceptance Testing**. A Beta version of the software, whose feedback is needed, is released to a limited number of end-users of the product to obtain feedback on the product quality. Beta testing helps in minimization of product failure risks, and it provides increased quality of the product through customer validation. It is the last test before shipping a product to the customers. One of the major advantages of beta testing is direct feedback from customers.

7.4 TEST CASES

Sl no.	Testcase	Expected Result	Actual Result	PASS/FAIL
1	Login and accessing into the website	Correct username and password	Correct username and password	PASS
2	Login and accessing into the website	Correct username and password	Incorrect username and password	FAIL
3	Submitting the form	Successful submission	Successful submission	PASS
3	Submitting the form	Successful submission	Please enter valid details	FAIL

Table 7.4.1: Testcases

8. RESULTS

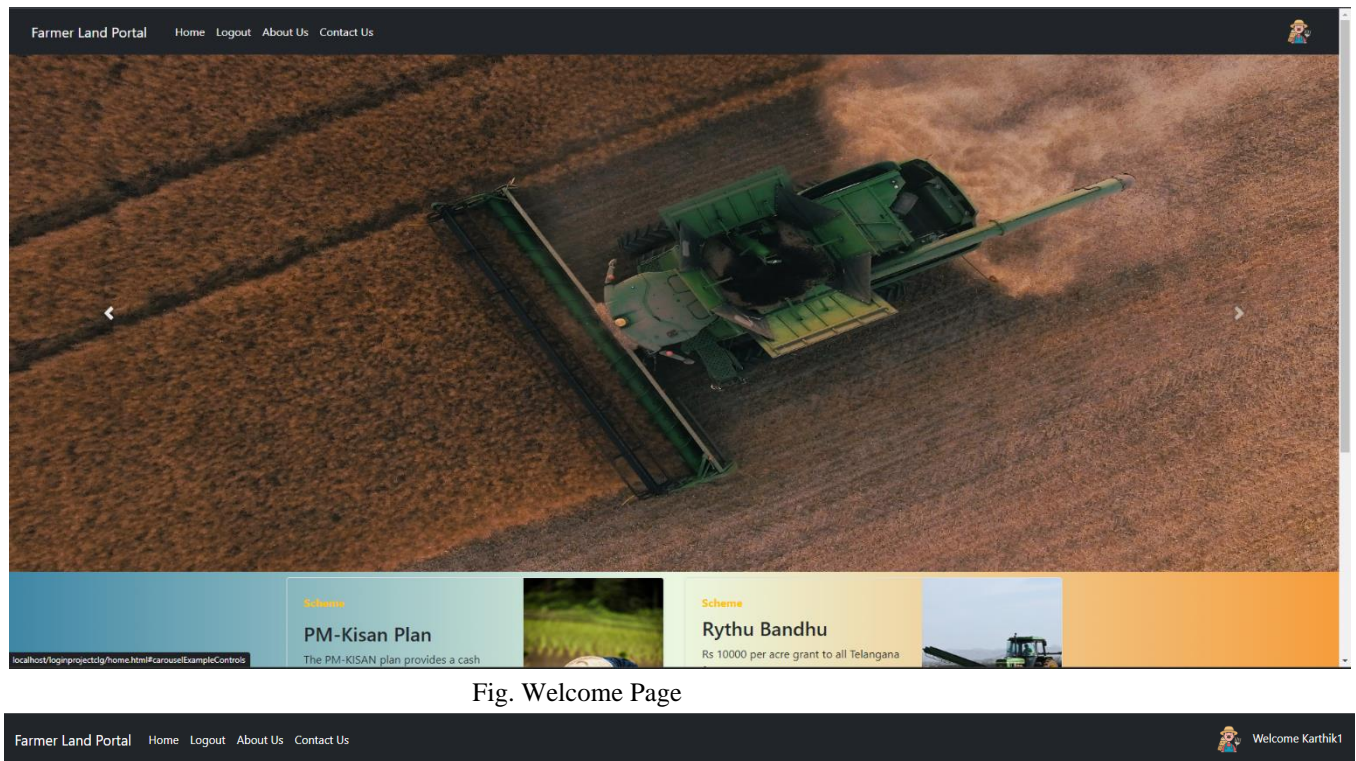


Fig. Welcome Page

Welcome Karthik1!

Please choose an option below:

[Enter land Details to claim benefits](#)

[Know more about all the Schemes provided](#)

Fig. Home Page

Enter Land and Bank Details:

Land District	Sub Division	Village	
<input type="text" value="Guntur"/>	<input type="text" value="guntur"/>	<input type="text" value="Bapatla"/>	
Type of Crop	Land Area (acres)	Organic farming ¹	Insurance ¹
<input type="text" value="Food Crop"/>	<input type="text" value="6"/>	<input type="text" value="Yes"/>	<input type="text" value="Yes"/>
<input type="button" value="Submit Details"/>			

Fig. Form 2/2

You are eligible for the following schemes:

1. Providing financial support to farmers suffering crop loss/damage arising out of unforeseen events from Pradhan Mantri Fasal Bima Yojana and Weather Based Crop insurance Scheme
2. Eligible for:
 - > Natural Machine on natural farming scheme
 - > paramparagat Krishi Vikas Yojana
 - > National Food Security Mission
 - > Capital Investment Subsidy Scheme under Soil Health Management Scheme

Fig. Result Page

9. CONCLUSION AND FURTHER WORK

Extending the project to support many languages, building more user-friendly interfaces. Time to time updates of any new introduced schemes. Delivering error free mechanism.

10. REFERENCES

- [1] <https://farmer.gov.in/>
- [2] <https://vikaspedia.in/schemesall/schemes-for-farmers>
- [3] <https://agricoop.gov.in/>