Project on

KrishiWala

A Dissertation submitted towards the Partial fulfillment for
VI Semester Examination of
Master of Computer Applications
(5 Year Integrated) 2022-2027
Session Jan-May 2025

Under the guidance of:

Dr. Nitin Nagar
Senior Assistant Professor
IIPS, DAVV

Submitted By:

(IC-2K22-04) Abhishek Yaduwanshi
(IC-2K22-34) Geetanshi Jain
(IC-2K22-72) Sachin Yaduwanshi

International Institute of Professional Studies,

Devi Ahilya Vishwavidyalaya, Indore, M.P. 2025

DECLARATION

We hereby declare that the project entitled **KrishiWala** which is submitted by us for the partial fulfillment of requirement for the award of Masters of Computer Application (5 years integrated) semester VI to International Institute of Professional Studies, Devi Ahilya Vishwavidyalaya, Indore, is authentic record of my own work carried out under the supervision of **Dr. Nitin Nagar, Senior Assistant Professor,** IIPS- DAVV, Indore.

The matter embodied in this dissertation work is authenticated and is genuinely done by us and has not been submitted to this university or any other university or Institute. Thus, we solely own the responsibility for the originality of the entire content. We also declared that the content of project report does not violate the copyright, Trademarks infringes on the patent, statutory right or propriety right of others.

Signature of Student:

Abhishek Yaduwanshi : _____

Geetanshi Jain : _____

Sachin Yaduwanshi : _____

Date : 07/05/2025

Place : Indore

CERTIFICATE FROM GUIDE

It is to certify that dissertation on "**KrishiWala**", submitted by **Mr. Abhishek Yaduwanshi**, **Ms. Geetanshi Jain** and **Mr. Sachin Yaduwanshi** to the International Institute of Professional Studies, DAVV, Indore has been completed under my supervision and the work is carried out and presented in a manner required for its acceptance in partial fulfillment for the award of the degree of "Master of Computer Application (5 Years) Semester VI."

Project Guide

Signature : _____

Name. : Dr. Nitin Nagar

Date. : 07/05/2025

CERTIFICATE

It is to certify that dissertation on "**KrishiWala**", submitted by **Mr. Abhishek Yaduwanshi, Ms. Geetanshi Jain** and **Mr. Sachin Yaduwanshi** to the International Institute of Professional Studies, DAVV, Indore has been completed under my supervision and the work is carried out and presented in a manner required for its acceptance in partial fulfillment for the award of the degree of Master of Computer Application (5 Years) Semester VI.

Internal Examiner:	**External Examiner:**
Signature: _____	**Signature : _____**
Name : Dr. Nitin Nagar	**Name : _____**
Date : 07/05/2025	**Date : _____**

ACKNOWLEDGEMENT

First and foremost, we are deeply thankful to our college, **International Institute of professional Studies (IIPS), DAVV, Indore**, for providing us with the platform, resources, and encouragement to undertake this innovative project.

A special mention needs to be made for **Prof. Yamini Karmarkar,** Director, IIPS, DAVV, **Dr. Jogendra Dongre,** Program In-charge, and our Batch Mentor **Dr. Pradeep Jatav,** who have always been supportive and cooperative throughout this journey.

We acknowledge our sincere gratitude to all those who have contributed significantly to this project. We would like to express our deep sense of appreciation to our project guide, **Dr. Nitin Nagar**, who has been a constant source of inspiration throughout the project. His belief in the idea and his insightful mantra, "**No idea is useless if people use it**" gave us the confidence to develop and refine our project. Whenever we faced uncertainties or doubts, he patiently listened, offered clarity, and guided us in the right direction. We are truly grateful for his encouragement and mentorship.

We are grateful to our colleagues and classmates for their continuous support through reviews, appreciation, and constructive suggestions that significantly contributed to enhancing the quality of our work. Before concluding, we would like to express our heartfelt gratitude to our parents and friends, who have always stood by us whenever we were in need. We sincerely regret any inadvertent omissions.

With heartiest thanks to all,
(IC-2K22-04) Abhishek Yaduwanshi
(IC-2K22-34) Geetanshi Jain
(IC-2k22-72) Sachin Yaduwanshi

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

Farmers in rural areas often face trouble finding land on rent, as they don't know the rental prices in nearby villages. At the same time, landowners struggle to get a fair price and find the right tenant. Small farmers also lack proper farming equipment, and during peak seasons, it becomes hard to find tools on rent. Another major issue is the shortage of labor, which delays important farming work. To solve these problems, we introduce KrishiWala, a user-friendly digital platform that connects farmers, landowners, equipment providers, and laborers. Landowners can post land details, and interested farmers can contact them directly. Equipment owners can list their tools for rent, and farmers can also hire laborers as per their needs. KrishiWala makes farming easier by improving communication, access, and trust among all stakeholders.

CHAPTER 1: INTRODUCTION

1.1 About the project

KrishiWala is a cutting-edge online platform designed to provide farmers with the facility of land renting and agricultural equipment on rent. In today's agricultural landscape, where the availability of land and expensive equipment is a major challenge, KrishiWala offers farmers a powerful, reliable, and user-friendly solution. KrishiWala provides the following features to farmers to solve this problem.

- Landowners can post their land.
- Farmers can approach landowners.
- Laboure can register to get work.
- Equipment owners can post their equipment to get a fair price.
- Farmers can take equipment on rant.

1.2 Problem Definition

Many farmers face significant challenges in renting agricultural land due to a lack of transparency in pricing and limited awareness of rental rates in different villages. As a result, they often struggle to find suitable land within their budget. On the other hand, landowners face difficulties in securing fair prices for their land and finding reliable tenants. Additionally, small-scale farmers often lack access to essential farming equipment, forcing them to rent equipment. However, they frequently encounter issues with equipment unavailability when needed, leading to delays and inefficiencies in farming operations.

December 2019, the estimated number of agricultural households in the country was approximately 93 million (9.3 crore). The survey also found that 83.5% of rural households owned less than one hectare of land, while only 0.2% owned more than 10 hectares of land. Workforce in Agriculture (As per India's 2011 Census) - According to India's 2011 Census, the total workforce in the country was 263 million (26.3 crore), out of which 119 million (11.9 crore) people were engaged in the agricultural sector.

Among them, approximately 14.4% (38 million or 3.8 crore) worked as agricultural laborers. This data was published by the Office of the Registrar General and Census Commissioner of India. How many farmers and landowners in India?

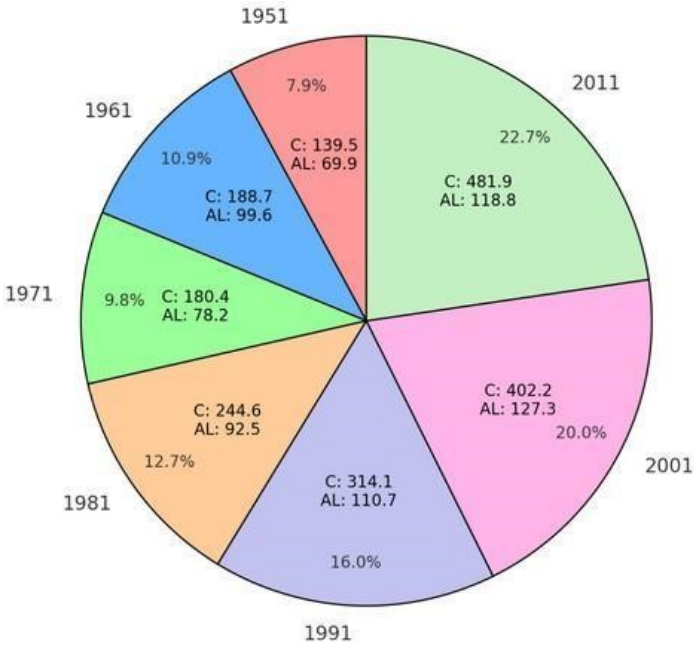


Fig 1.1: Distribution of Cultivators and Agriculture Labours

This pie chart shows the distribution of Cultivators (C) and Agricultural Laborers (AL) in India from 1951 to 2011. The outer ring represents different years, while the inner section displays their numbers (in millions) with distinct colors for clarity. It also highlights the percentage of the total agricultural workforce, revealing employment trends over time. Beyond the numbers, the pie chart also emphasizes the percentage distribution of the total agricultural workforce, offering insight into how the balance between Cultivators and Agricultural Laborers has evolved over time. The visualization thus not only provides statistical data but also serves as a powerful tool to understand long-term employment patterns and the changing face of agriculture in India. Understanding these trends is crucial for designing effective agricultural and rural development policies. Policymakers must address the growing dependence on wage labor in agriculture.Ensuring skill development and alternative livelihood options can help improve rural resilience.

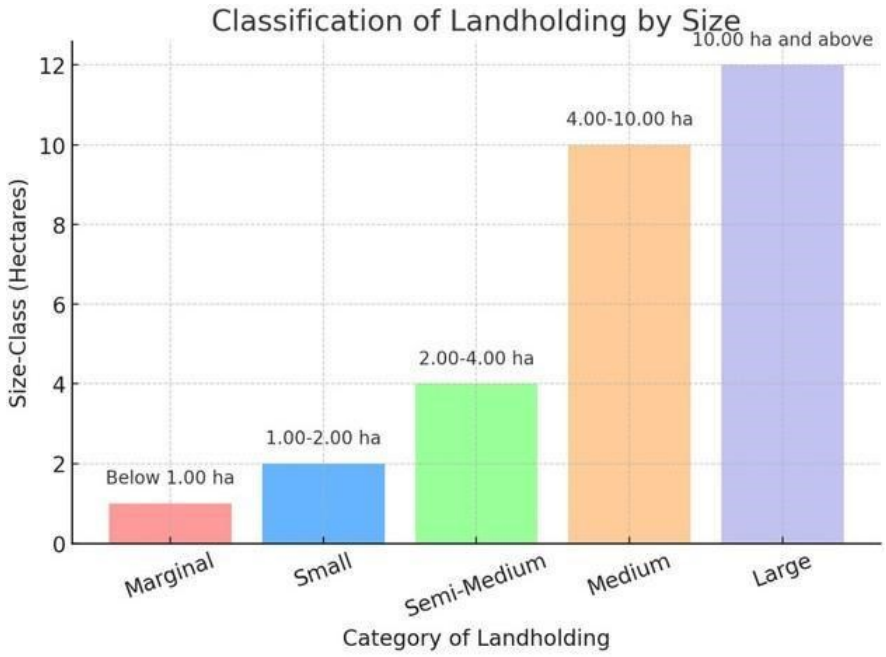


Fig 1.2: Category of Land Holding

The Fig 1.2 represents different categories of land based on their size in hectares. It categorizes landholdings into five groups:

- **Marginal**: Below 1.00 hectare
- **Small**: 1.00–2.00 hectares
- **Semi-Medium**: 2.00–4.00 hectares
- **Medium**: 4.00–10.00 hectares
- **Large**: 10.00 hectares and above

Each category is represented by a different colored bar, with the bar height showing the upper range of land size. This classification helps in analyzing land distribution patterns, ownership, and policy planning related to agriculture. The chart visually shows how landholding categories grow from marginal to large, emphasizing disparities in agricultural land access.

1.3 Aims

The aim of the KrishiWala project is to create a digital platform that connects landowners and farmers, enabling seamless land leasing and equipment rental. The system aims to:

I. Bridge the Information Gap – Provide farmers with real-time rental prices of land in different villages.
II. Ensure Fair Pricing – Help landowners get better prices for their land by increasing accessibility to potential tenants.

By implementing this system, KrishiWala aims to make land and equipment rental more transparent, efficient, and accessible for farmers. The primary goal of this project is to enhance market access by creating a direct connection between farmers, consumers. By bridging this gap, the platform aims to empower farmers to reach a broader customer base without relying heavily on intermediaries. This leads to a significant reduction in middlemen dependency, ensuring that farmers receive fair prices for their produce and enjoy higher profit margins. Additionally, the system seeks to increase transparency in agricultural trade by enabling direct communication between Owner and Tenants. This fosters trust, encourages fair trade practices, and contributes to a more efficient and ethical marketplace.

1.4 Project Goals

The primary goal of this project is to enhance market access by creating a direct connection between farmers, consumers. By bridging this gap, the platform aims to empower farmers to reach a broader customer base without relying heavily on intermediaries. This leads to a significant reduction in middlemen dependency, ensuring that farmers receive fair prices for their produce and enjoy higher profit margins. Additionally, the system seeks to increase transparency in agricultural trade by enabling direct communication between Owner and Tenants. This direct interaction not only builds trust but also enables better coordination in terms of crop planning, demand forecasting, and delivery schedules. The platform minimizes communication gaps and delays, creating a more efficient and farmer-friendly agricultural system.

1.5 Benefits

- **For Landowners:** Landowners can easily list and lease their unused land, helping them earn a regular income without much effort.
- **For Tenants:** People looking for land can quickly find the right place for farming or other work, saving time and effort.
- **For Laborers:** Agricultural workers can find job opportunities nearby, improving their chances of regular employment.
- **For Business Owners:** Those managing the platform can earn commission on each successful land deal or booking, creating a source of income.
- **For Farmers (End Users):** Farmers can access a wide range of services, including machinery, Laboure, and land, all from one platform, making farming operations more efficient and cost-effective.
- **For Service Providers:** Providers of agricultural machinery or Laboure services can expand their reach by connecting with more farmers, increasing their customer base.
- **For Local Communities:** By facilitating easier access to land, Laboure, and machinery, the platform helps boost local economies, contributing to the overall development of rural areas.
- **For Investors:** Investors in the platform can benefit from its growth, as it has the potential to scale and generate consistent revenue, making it an attractive investment opportunity.
- **For Government & Policy Makers:** The platform can provide valuable real-time data on land usage, employment trends, and service demand, helping in better policy formulation and targeted rural development initiatives.
- **For Environmental Sustainability:** By promoting efficient land use and shared resources like machinery and encourages eco-friendly farming practices, supporting sustainable agriculture.

CHAPTER 2: ANALYSIS

2.1 Feasibility Analysis

Feasibility analysis is a crucial step in determining the viability of any project. The feasibility study for this project is conducted based on the following criteria:

2.1.1 Economic Feasibility:

In our project, the development is carried out using open-source technologies such as React.js for the frontend and Django for the backend, paired with SQLite3 for lightweight database management. These technologies are completely free and have strong community support, making them ideal for a cost-sensitive project. The system benefits farmers by reducing dependency on middlemen, improving land and equipment utilization, and creating employment opportunities for laborers.

2.1.2 Technical Feasibility:

The platform can be built using Django for the backend, React.js for the frontend, and SQLlite3 for data storage. For deployment, we use Render for hosting the Django backend and Vercel for deploying the React frontend. Both platforms offer free-tier services with seamless deployment pipelines, minimizing infrastructure costs while ensuring reliable performance. We use JWT (JSON Web Token) authentication to securely manage user sessions and access control. The tokens are signed using the HS256 (HMAC with SHA-256) algorithm, ensuring integrity and confidentiality during data transmission.

The platform integrates with Google Maps for location searching. Regular updates and automated testing are performed to ensure smooth and reliable performance. The **KrishiWala** project is technically feasible, utilizing modern, open-source technologies to ensure efficient **development, scalability, security, and real-time communication**. The **frontend** will be developed using **React.js,** offering a dynamic

and modular user interface. The **backend** will be powered by **Django**, This technology stack guarantees a high-performance, scalable, and user-friendly agricultural platform.

2.1.3 Operational Feasibility

The system is designed to be user-friendly, featuring a clean and intuitive interface that enables farmers, landowners, and laborers to easily navigate the platform, post listings, and book services without requiring any technical expertise. It simplifies and automates processes such as land leasing, equipment rentals, and labor hiring, significantly reducing manual effort and improving overall efficiency through features like location-based search and direct communication.

Accessible on both mobile and desktop devices, the platform supports English language usage and integrates a chatbot for real-time assistance, enhancing user support. The system is built to operate sustainably with minimal human intervention, relying on automated workflows and real-time data updates to ensure smooth and reliable long-term functioning. With a user-centric design and minimal operational costs, **KrishiWala** is highly feasible for real-world implementation.

2.2 System Analysis

System analysis is the process of studying and understanding the requirements of the system, identifying problems, and defining the functional and non-functional aspects of the proposed solution

2.2.1 Define Functional Requirements:

A Functional Requirement describe the functions, tasks, or activities the system should be able to perform. These are typically based on user needs, and they guide the development of the system's features and interfaces. The following are some functional requirements needed for KrishiWala.

I. **User Login and Authentication:** The system must allow users to register and log in using valid credentials. Invalid login attempts should trigger appropriate error messages and prevent access.

II. **Data Entry/ Input Forms:** Users should be able to submit required information through forms such as land listing, labor registration, or machine availability. The data must be validated and stored in the database securely.

III. **Search and Filter Options;** The system must provide users with the ability to search for labor, land, or machines based on criteria such as location, category, and availability using advanced filters.

IV. **Booking or Request Feature:** Users should be able to send booking requests to landowners, laborers, or machine providers. The system should handle the request lifecycle, including status updates like pending, accepted, or rejected.

V. **User Dashboard:** A dedicated dashboard must allow users to manage and update their personal details, track past bookings, and view/edit registered assets (land, labor, machinery).

VI. **Chat Support Module:** The system must include a real-time chat feature that enables users to communicate with the support team to resolve queries or seek assistance regarding services.

VII. **Database Interaction;** The system should support efficient and secure database operations including insertion, retrieval, updating, and deletion of user and module data.

2.2.2 Define Nonfunctional requirements:

A **Non-Functional Requirement** defines **how** a system should perform rather than what it should do. These requirements describe the **quality attributes** of the system, such as performance, security, usability, and reliability. They focus on aspects such as system performance, user experience, and operational constraints to ensure the system is efficient, secure, and easy to use.

The following are some Nonfunctional requirements needed for KrishiWala.

I. **Performance:** The system must ensure fast response times for all farming-related queries. Real-time access to labour, land, and machinery data should be maintained by optimizing backend logic and SQL database queries.

II. **Security:** KrishiWala will implement JWT (JSON Web Token) authentication using the HS256 algorithm to ensure secure user login and identity verification. All sensitive data related to farmers, landowners, and service providers will be securely encrypted during both transmission (using HTTP) and storage

III. **Usability:** Design an intuitive UI, support English languages and ensure mobile responsiveness.

IV. **Scalability:** KrishiWala will use SQLite3 for efficient data storage during the initial phase, with database indexing and query optimization to handle growing data. The Django backend, deployed on Render, will be optimized for smooth scaling and high responsiveness. Future scalability may involve migrating to PostgreSQL to handle large-scale data and high concurrency.

2.3 Software Requirements:

These define the software requirements needed for developing the system. The following are the software requirements needed for KrishiWala.

I. **Operating System:**

- Windows 10 or higher

II. **Web Browsers (For End Users)**

- Google Chrome
- Mozilla Firefox
- Microsoft Edge

III. **Frontend**

- Framework: React.js (v18.2.0)
- Build Tool: Vite (v4.4.9)
- Styling Library: Tailwind CSS (v3.3.2)

IV. **Backend**

- Framework: Django (v4.2.5)
- API Framework: Django REST Framework (DRF)
- Programming Language: Python (v3.11.x)
- Web Server: Gunicorn (v21.2.0) (for backend deployment.

V. **Database**

- DBMS: SQLite3

VI. **Frontend Build Tool Dependency**

- Node.js: required for running Vite and building frontend assets.

VII. **Hosting Platforms**

- Frontend Hosting**:** Vercel
- Backend Hosting: Render

VIII. **API Format**

- Type: REST APIs (for communication between frontend and backend)

IX. **Integrated Development Environment (IDE)**

- Visual Studio Code

X. **Version Control System**

- Git:
- Repository Platform: GitHub

2.4 Hardware Requirements

Hardware Requirements refer to the **physical components** needed to develop and support a software system effectively. The following are the software requirements needed for KrishiWala.

I. **Processor**: AMD Ryzen 5.
II. **RAM**: Minimum 8 GB of higher.
III. **Storage**: At least 500 MB of free space.
IV. **Operating System**: Windows 10 or higher.

CHAPTER 3: PROJECT PLANNING

3.1 Project scope

KrishiWala is a comprehensive digital platform designed to connect farmers, landowners, and laborers, facilitating streamlined processes for land leasing, equipment rentals, and labor hiring. The platform empowers users to register, post listings, search with filters, and make bookings with ease. The platform benefits landowners, farmers, and businesses while generating revenue through commission-based bookings. An admin panel ensures verification and smooth operations. This project aims to enhance agricultural efficiency, reduce dependency on middlemen, and provide a transparent marketplace for rural communities.

3.2 Project Objective

The KrishiWala platform is designed to digitize and simplify agricultural by connecting farmers, landowners, and laborers. It enables users to lease land, rent equipment, and hire labor seamlessly through an intuitive interface. The system provides secure registration, listing management, search and filter options, direct bookings, and online payments. It ensures transparency, efficiency, and accessibility for all stakeholders, reducing dependency on intermediaries. With features like searching, posting, and chat support, the platform enhances agricultural productivity and promotes fair trade in rural areas.

3.3 TimeLine

The project timeline outlines the sequence of activities and key milestones planned during the various phases of project development. Figure 3.1 shows the detailed timeline to develop KrishiWala in a structured and systematic manner.

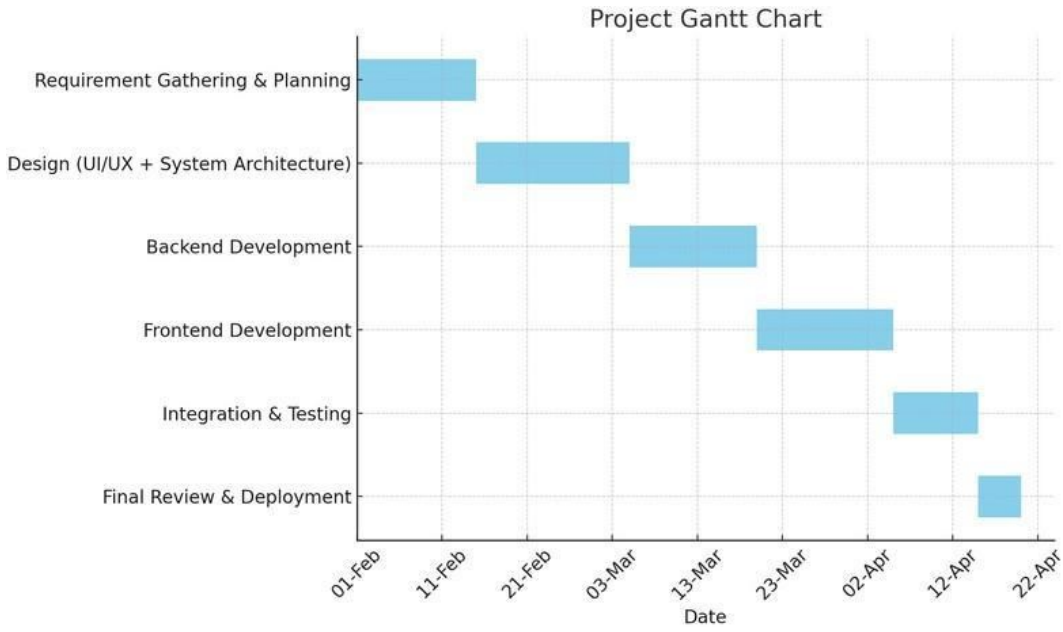


Fig 3.1: Project planning Gantt Chart

3.4 Task Breakdown (Work Breakdown Structure - WBS)

A Work Breakdown Structure (WBS) is a project management tool that breaks the project into smaller, manageable tasks. The WBS defines the project development and maintenance activities along with week-wise timelines. The detailed description of the Work Breakdown Structure (WBS) for KrishiWala is as follows.

Week 1: Planning and Documentation

The project begins with careful requirement analysis where we identify user needs and define the core functionalities of the system. This stage involves detailed documentation of all the requirements, including functional and non-functional needs. The technology stack is finalized after evaluating the most suitable tools and technologies that align with the project goals. The next step in the planning phase is to design the database structure and overall system architecture, laying the foundation for a scalable, efficient, and user- friendly application. This includes defining how various components like user data, machine rentals, and labor listings will interact with each other and be stored in the backend.

Week 2: UI/UX Design & Frontend Development

Once the core features and structure are defined, the UI/UX design phase begins. The focus is on creating a user-friendly interface with intuitive navigation that ensures a seamless experience for all types of users. The frontend development process follows where responsive web pages are created for various modules like registration, listings, and bookings. These pages need to be fully functional on desktop and mobile platforms, ensuring accessibility across devices. Additionally, a key part of this stage is testing the user flow and ensuring that every interaction within the application is easy and ready to use.

Week3: Backend Development & Database Integration

In this phase, the backend development takes place. The core system logic is implemented using frameworks such as Django, depending on project requirements. This involves setting up the authentication system, ensuring secure login and user management. Data storage mechanisms are also set up to ensure all user-generated data such as land listings, machine rentals, and labor availability are stored securely and efficiently. This is followed by the integration of the database, where the frontend communicates with the backend to retrieve and store data seamlessly. It's essential to ensure that the backend is optimized to handle high traffic and large datasets as the platform scales.

Week4: Integration, Testing & Quality Assurance

Testing begins as soon as individual modules are implemented. Unit testing ensures that each component of the system functions correctly in isolation. Integration testing follows to check if different modules work together as expected and if the data flows seamlessly between the backend and frontend. User acceptance testing is done with a group of real users to ensure that the system meets their needs and is intuitive to use. Additionally, quality assurance is conducted throughout the development process to identify and resolve any potential bugs or performance issues.

Week5: Maintenance & Future Enhancements

Once the system is deployed, ongoing maintenance is required to monitor system performance and ensure that the platform operates smoothly. Any bugs discovered after deployment are promptly fixed, and system performance is continuously optimized. Regular updates and patches are applied to improve security and functionality, ensuring that the platform stays up-to-date with the latest technologies and best practices. Additionally, user feedback is continuously gathered to identify areas for improvement, enabling the platform to evolve in line with the needs of its users. Performance metrics such as loading times, response rates, and error logs are carefully tracked to maintain a high level of reliability.

3.5 Team Structure

Team Structure refers to the way in which the roles, responsibilities, and communication flow are organized within a project team. The team structure of KrishiWala is organized as follows.

I. Abhishek Yaduwanshi

- Role: Backend Development & Deployment
- Responsibilities: API development, database and Authentication mechanism.

II. Geetanshi Jain

- Role: Frontend Development & Testing
- Responsibilities: UI/UX design, frontend and application testing.

III. Sachin Yaduwanshi

- Role: Initial Investigation
- Responsibilities: Problem analysis, requirement gathering.

CHAPTER 4: SYSTEM DESIGN

System Design of KrishiWala defines the structural layout of the software that connects farmers, landowners, laborers, and machinery providers through a unified digital platform.

4.1 System Architecture

System Design of KrishiWala defines the structural layout of the software that connects farmers, landowners, laborers, and machinery providers through a unified digital platform. It follows a three-tier architecture: frontend for user interaction, backend for logic handling, and a database for storing user and request data. Each module—land, labor, and machinery—is functionally connected through filtered search and request systems. The design ensures scalability, data security (with SHA-256 encryption), and user-friendly access to agricultural services.

I. **Presentation Layer (Frontend):** This layer is responsible for all user interactions. It includes interfaces for user registration, login, land/labour/machine listings, and request management. It allows farmers to search, apply filters, and send/receive/view requests.

II. **Application Layer (Backend Logic):** This is the core layer where all the business logic resides. It handles request processing, input validation, user authentication (with SHA-256 password encryption), and communication between the frontend and database.

III. **Database Layer (Data Storage):** This layer stores all essential data including user information, land, machine, labor, request data, and bank account details. It supports operations like insertion, retrieval, and updates securely and efficiently. It also enables seamless interaction between different modules by acting as a centralized data hub. Data validation and access control mechanisms are implemented to ensure integrity and privacy.

4.2 System Workflow Diagram

The Fig 4.1 describes the overall System Workflow, when a user opens the application, they are first directed to the **login page**. If the user is already registered, they can log in directly; otherwise, they are redirected to the **registration page**. After successful registration, the user is taken back to the login page. Upon successful login, the user is presented with six options.

(1) Land Registration
(2) Search Land
(3) Machine Registration
(4) Search Machine
(5) Labour Registration
(6) Search Labour

If the user clicks on the **Land Registration** option, a form is displayed. If the form is filled correctly without any errors, it is submitted successfully. In the **Search Land** option, an interface appears where users can filter land records based on their needs. The **Machine Registration** and **Labour Registration** options work similarly, forms are shown, and if completed correctly, they are submitted successfully. In the **Search Machine** and **Search Labour** options, users are provided with interfaces to filter the available records according to their requirements.

If a user is interested in any land, machine, or Laboure listing during the search, they can send a request to the respective owner. If the owner accepts the request, the user is granted permission to proceed with the booking. To enhance user experience and provide instant assistance, an intelligent chatbot is integrated throughout the application. This chatbot is available 24/7 and can help users with navigation, resolving common issues, answering frequently asked questions, and even guiding them step-by-step through the booking or request process.

Additionally, all actions performed by the user are tracked and stored for future reference, allowing users to view and manage their history. Notifications and alerts are also generated for important actions like request approvals, enhancing communication and response time within the system.

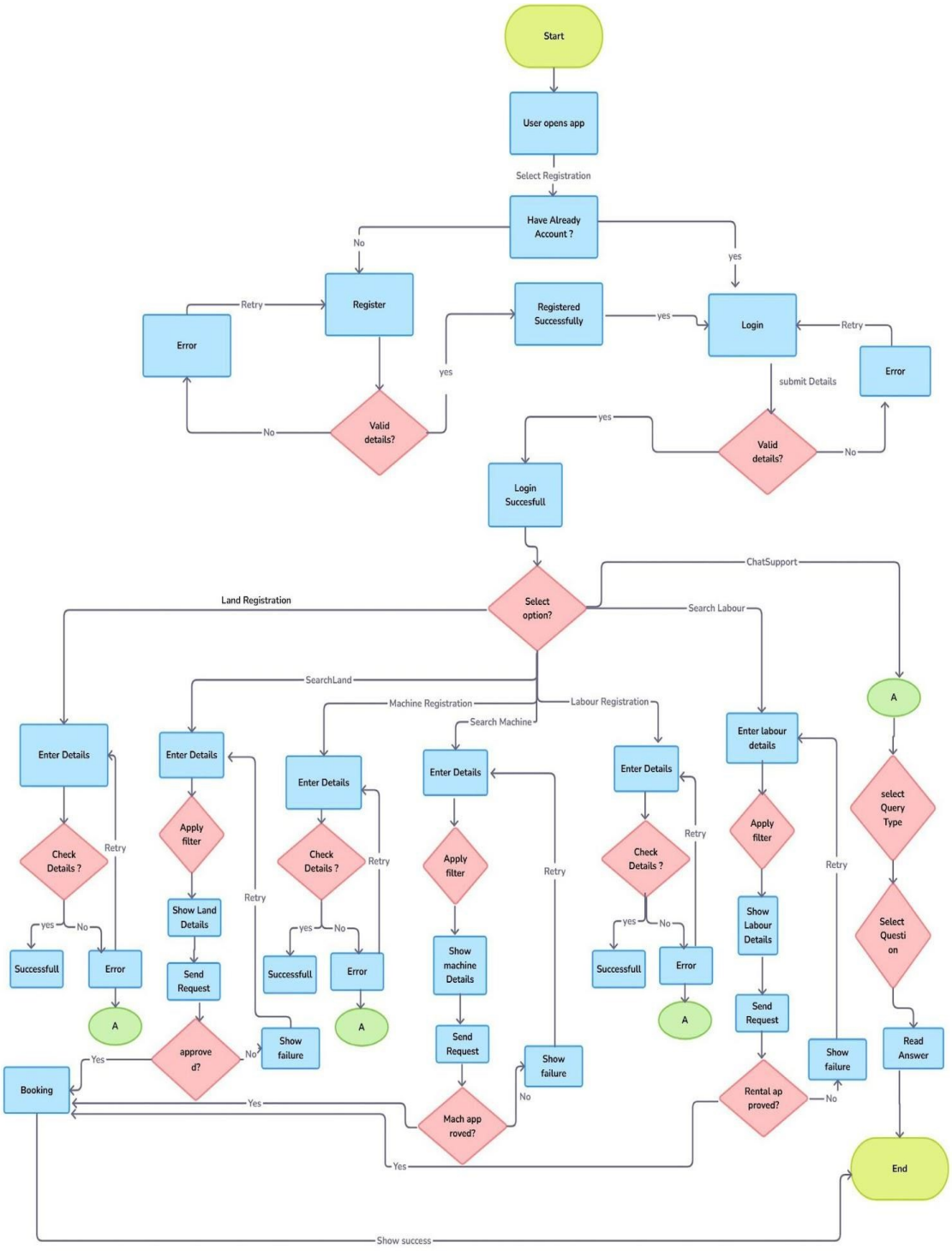


Fig 4.1: System Workflow Diagram

4.3 Module-wise Functional Flow

The Fig 4.3 describes Module-wise Functional Flow, each section outlines the specific operations and interactions within each module of the KrishiWala system. It describes the step-by-step processes for key modules such as land rental, machinery rental, and labor hiring, detailing how each user action triggers the corresponding system response.

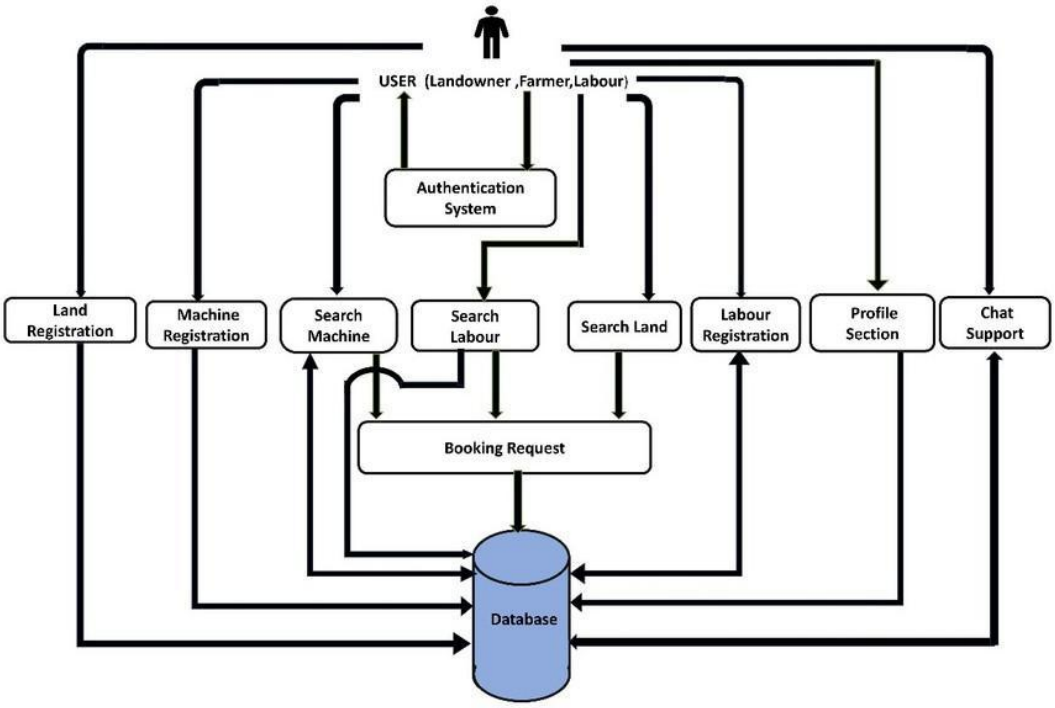


Fig 4.2: Module-wise Functional Flow

4.3.1. Description of Each Module

This section provides a detailed explanation of each core module developed for **KrishiWala.** The description of each module in KrishiWala is as follows:

I. User Authentication Module

- When a user first opens the application, they are directed to the Login Interface.
- If the user is already registered, they can log in directly. Otherwise, they are redirected to the Registration Page.

- Upon successful registration, the user is navigated to the login page, and after successful login, all core features of the application become accessible.

II. Land Registration Module

- When the user clicks on "Land Registration", a form is displayed where they can enter land-related information such as location, size, price, rental period, and available facilities.
- If all inputs are valid, the form is submitted successfully, and the data is securely stored in the backend database

III. Land Search & Request Module

- On selecting "Search Land", the user is presented with a filter-based interface where they can search for land based on various criteria like location, size, price, and duration.
- If suitable land is found, the user can send a request to the landowner.
- If the request is accepted by the owner, the user is granted permission to proceed with the booking.

IV. Machine Registration Module

- The "Machine Registration" option allows users to enter the details of agricultural machinery they wish to rent out, such as machine type, condition, pricing, etc.
- If the data passes validation, the form is successfully submitted and the machine is listed in the system.

V. Machine Search & Request Module

- By selecting "Search Machine", the user is presented with a filtering interface to search for machinery as per their requirements.
- Upon finding a suitable machine, the user can send a request to the machine owner. Once the owner accepts the request, the user gains access to book the machine.

VI. Labour Registration Module

- The "Labour Registration" module enables laborers to register their details including name, location, skills, and availability.
- If all inputs are valid, the data is securely submitted and stored in the backend database.

VII. Labour Search & Hiring Request Module

- The "Search Labour" option allows farmers to search for available laborers using filters such as location and skill type.
- Once a suitable laborer is found, the farmer can send a hiring request. •Upon acceptance by the laborer, the hiring is confirmed.

VIII. Request Management System

- All types of requests (Land, Machine, Labour) are stored in a centralized Request Table.
- In the User Profile Section, requests are categorized as:
    - Sent Requests – Requests sent by the user.
    - Received Requests – Requests received by the user.
    - Previewed Requests – Requests that have been viewed but are awaiting action

IX. Profile Management Module

The Profile Management Module acts as a centralized dashboard where users can view and manage their activities. It displays the user's name and mobile number, which are auto-filled and locked for security. Users can track sent, received, and previewed requests for land, machinery, and labor. It also shows assets registered by the user, including land, machines, and labor details. It also shows assets registered by the user, including land, machines, and labor details. This helps users keep a clear record of their ongoing engagements. The streamlined layout ensures easy navigation and better control over their agricultural resources.

X. Chatbot Assistance Module

- A Chatbot Integration is provided within the system to assist users at any stage of the application.
- The chatbot is capable of understanding user queries and help section, enhancing the overall user experience.

4.4 Detailed Component Design

The Detailed Component Design outlines the internal structure and functioning of each module within the system. It focuses on how different components like backend logic, database schema, and user interface elements interact to deliver the desired functionality.

4.4.1 Backend Logic

The backend of the **KrishiWala** platform is developed using **Django** and the **Django REST Framework (DRF)**. It is designed to solve a real-world problem faced by farmers in renting agricultural land, hiring labour, and renting machinery. The system is structured to handle multiple user interactions and ensure secure, seamless communication between frontend and backend components through RESTful APIs.

Authentication and User Management

- The application includes secure **user registration and login functionality**.
- During registration, users provide their **name, email, mobile number, and password**. Passwords are securely stored using **SHA-256 hashing** for encryption.
- After login, users receive a **JWT (JSON Web Token)** to manage session-based access control.
- The system restricts changes to the **mobile number and name** after registration to maintain consistent identity and data integrity across all features.

Core Database Models

The backend is structured around the following key models:

- **User**: Stores the primary user data including name, email, mobile number, and hashed password.
- **Land**: Contains land-related details such as location, size, price, time period, and facilities.
- **Machine**: Stores details of agricultural machinery registered for rent.
- **Labour**: Holds information about registered labourers available for hire.
- **Requests**: Three separate tables manage the request system:
    - LandRequest: Handles rental requests for land.
    - MachineRequest: Manages requests for renting machinery.
    - LabourRequest: Manages labour hiring requests.
- **BankDetails**: Stores optional bank account details for labourers and machine owners, enabling secure payment transactions.

Workflow Logic

I. **User Access**:
    a. Users must log in using their **registered mobile number and password**.
    b. If not registered, users must first complete the **registration** process.

II. **Search and Filter**:
    a. Users can search for **land, machinery, or labour** using dynamic filters such as location, price, size, duration, and type of work.

III. **Registration of Assets**:
    a. Farmers can register their **land, machinery, or themselves as labour**.
    b. During registration, the system **auto-fills and locks** the name and mobile number based on the logged-in user to prevent data tampering.

IV. **Request Management**:
    a. Farmers can send requests to:
        i. **Land owners** (to rent land),
        ii. **Machine owners** (to rent equipment),
        iii. **Labourers** (to hire for agricultural work).
    b. All requests are stored and tracked in their respective request tables.

This modular and RESTful architecture ensures a **scalable, secure, and user-friendly** experience, allowing seamless interactions for farmers seeking to rent or offer agricultural resources.

4.4.2 Database Design

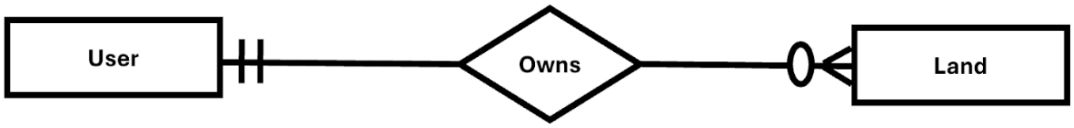
Database design is a concept that shows how the system stores and accesses data. It defines the overall architecture of data storage in the system, ensuring efficient data retrieval and manipulation. A well-structured database design is essential for ensuring data integrity, security, and scalability. This section details the KrishiWala database, highlighting the key tables and their relationships, which enable seamless management of land rentals, machine rentals, and labor hiring. By structuring the data around entities like users, land, machines, labor, and requests, the system ensures that information is stored in an organized manner that supports easy access, updates, and querying. The database uses normalization techniques to reduce data redundancy, optimize performance, and enhance the user experience. Additionally, we have incorporated security features, such as encrypting passwords with the SHA-256 hashing algorithm, ensuring that sensitive information is protected.

4.4.2.1 Relationship and Mapping Cardinalities

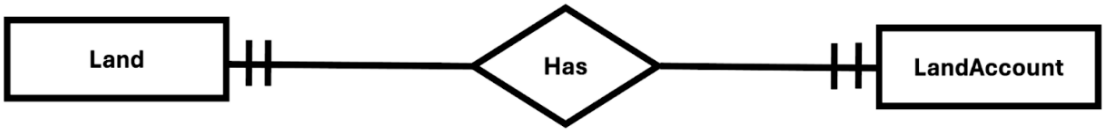
In database design, **relationship and mapping cardinalities** define how entities (tables) in the system are related to each other and how many instances of one entity can be associated with instances of another. In the context of **KrishiWala**, understanding these relationships is crucial for designing an efficient and logical database schema. These relationships include one-to-one, one-to-many, and many-to-many associations between entities like users, land, machines, and labour. Proper mapping ensures data integrity and supports smooth operations across the platform.

i) Binary relationship examples:

Binary relationships define how two entities relate to each other within a database structure. For **KrishiWala**, here are some examples of binary relationships that are used for developing the system.

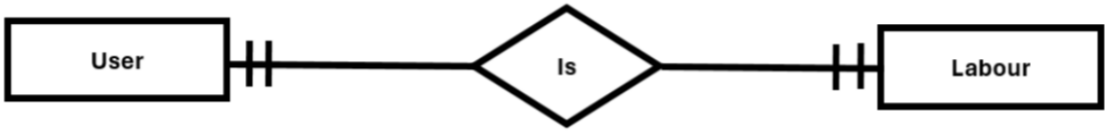


Description: Each User can own zero or more Lands, but every Land must be owned by exactly one User.



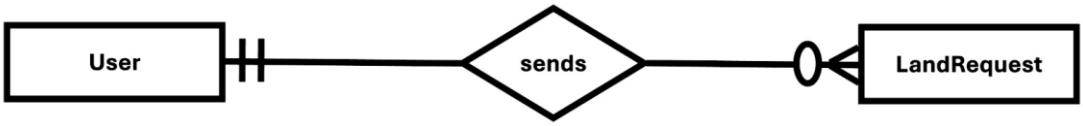
Description: Each Land must have exactly one LandAccount, and each LandAccount is linked to exactly one Land.

Description: Each Land can have zero or more associated LandPhotos, but each LandPhoto is connected to exactly one Land.



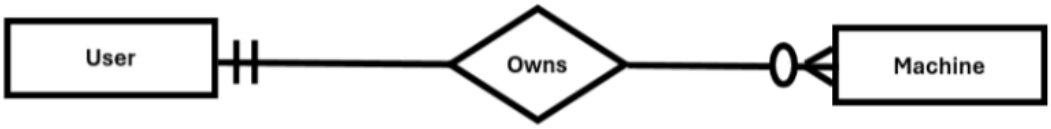
Description: Each User in the system is exactly one Labour, and each Labour is associated with exactly one User.

Description: Every Machine must have at least one associated Machine Image, while each Machine Image is optionally associated with exactly one Machine.

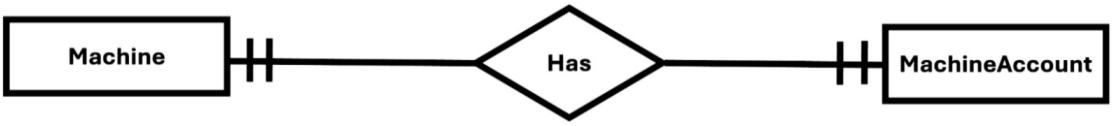


Description: Each User can send zero or more Land Requests, but each Land Request must be sent by exactly one User.

Description: Every Labour must receive at least one Labour Request, while each Labour Request may or may not be assigned to a Labour.

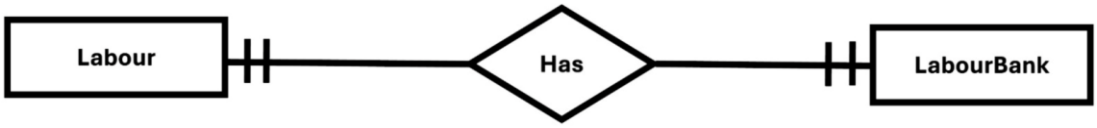


Description: Each User must own at least one Machine, while a Machine may or may not be owned by a User.



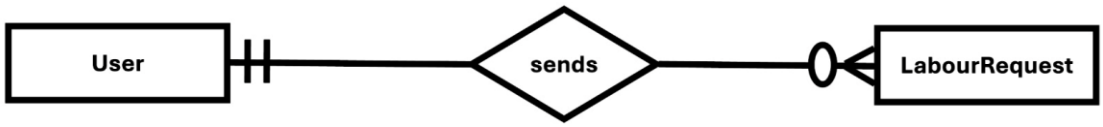
Description: Every Machine must have exactly one Machine Account, and every Machine Account must be linked to exactly one Machine.

Description: Every Land must receive at least one Land Request, while each Land Request may or may not be associated with a Land.

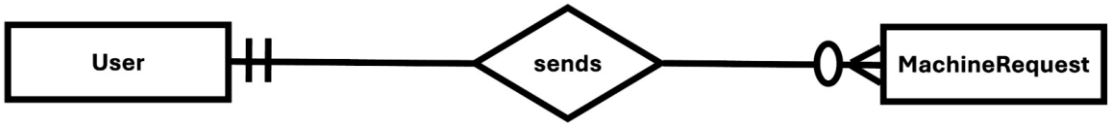


Description: Every LabourBank must be associated with at least one Labour, while each Labour may or may not be associated with a LabourBank.

Description: Every Machine must receive at least one MachineRequest, while each MachineRequest may or may not be associated with a Machine.



Description: Each User can send zero or more LabourRequests, but each LabourRequest must be sent by exactly one User.



Description: Each User can send zero or more MachineRequests, but each MachineRequest must be sent by exactly one User..

4.4.2.2 ER diagram

An E-R (Entity-Relationship) Diagram visually represents the entities, attributes, and relationships within a system. It helps in understanding the database structure and how different components interact. ER diagrams are essential for planning, designing, and optimizing database systems effectively.

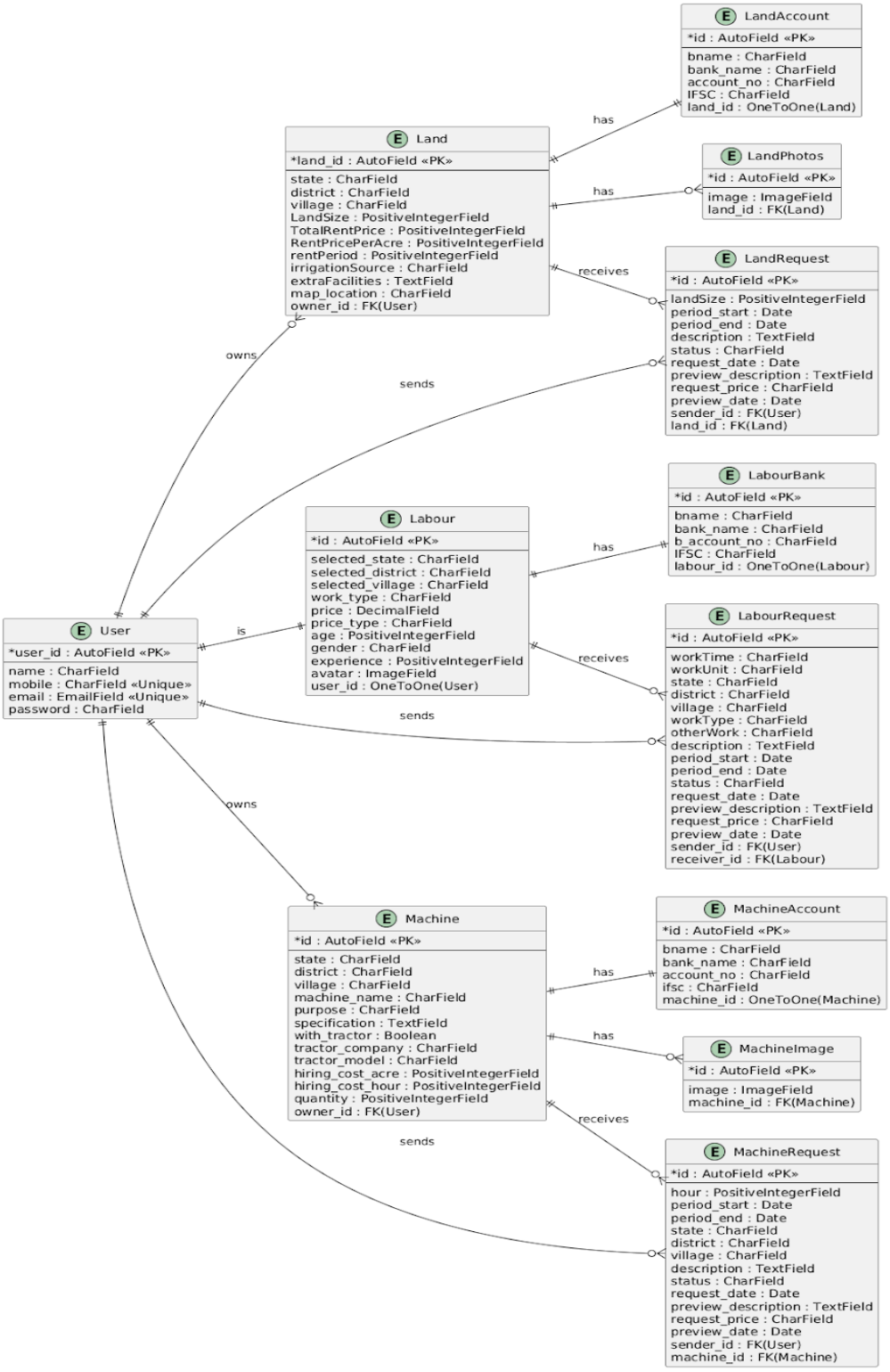


Fig 4.3: ER Diagram of KrishiWala

4.4.2.3 ER Diagram Description:

The ER (Entity-Relationship) diagram(Fig 4.3) of the **KrishiWala** software represents the overall structure of how different entities in the system are connected and interact with each other. At the core of the system is the **User** entity, which represents farmers or individuals who can either register their land, machines, or themselves as labourers. Each user has a unique identity with personal information like name, mobile number, email, and password. A user can register one or more **lands**, where each land entry stores information such as its location (state, district, village), size, rent price, period, irrigation source, extra facilities, and map location. Photos of the land are stored separately in the **LandPhotos** table, while banking details for rental transactions are managed in the **LandAccount** table.

Similarly, users can also register **machines** available for rent. Each machine includes details like machine name, purpose, specifications, location, pricing (per hour or per acre), and whether it comes with a tractor. Machine-related photos are saved in the **MachineImage** table, and associated bank account details are stored in the **MachineAccount** table. A user can also act as a **labourer**, with information such as preferred location, work type, price, age, gender, and experience, all stored in the **Labour** table. Labourers also have their own banking details recorded in the **LabourBank** table.

The system supports a structured **request system** where users can send and receive requests related to land rental (**LandRequest**), machine rental (**MachineRequest**), or labour hiring (**LabourRequest**). Each request contains relevant details such as time period, location, price, status, and descriptions. These requests help connect the right people based on their needs, and each user can view their sent, received, or previewed requests in their profile.

Overall, the ER diagram showcases how the KrishiWala platform integrates different roles of farmers (as landowners, labourers, or machine providers) and enable

4.4.2.4 Normalization

Normalization is a systematic approach of organizing data in a database to reduce data redundancy and improve data integrity. The goal of normalization is to decompose larger tables into smaller, well-structured tables without losing information or introducing anomalies. The process typically follows several normal forms (NFs), each with its own set of rules.

In this project, the database schema has been carefully designed to follow the First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF).

(i) First Normal Form (1NF): A relation is said to be in First Normal Form when all attributes contain only atomic (indivisible) values and there are no repeating groups or arrays in any table. In the proposed system, all the tables are designed with atomic fields such as name, mobile, email, state, district, etc. There are no multi-valued, nested, or composite fields in any of the entities. As a result, the database schema satisfies the conditions of the First Normal Form (1NF).

(ii) Second Normal Form (2NF): A relation is in Second Normal Form if it is already in 1NF and all non-key attributes are fully functionally dependent on the entire primary key, eliminating any partial dependencies. In this project, all tables have single-column primary keys such as user_id, land_id, and id. Moreover, every non-key attribute in these tables is fully functionally dependent on its respective primary key. For example, in the Land table, attributes like state, village, and landSize depend solely on land_id. Hence, the database design is in accordance with the Second Normal Form (2NF).

(iii) Third Normal Form (3NF): A relation is in Third Normal Form if it is in 2NF and there are no transitive dependencies between non-prime attributes. The system avoids such transitive dependencies by separating related information into different tables. For instance, banking details for land, labour, and machines are stored in LandAccount, LabourBank, and MachineAccount tables respectively. Each of these

tables contains attributes like bank_name, account_no, and IFSC, which are directly dependent only on their primary key (id). Furthermore, foreign keys like land_id are used to establish one-to-one relationships with their respective entities. Therefore, the database schema fully adheres to the Third Normal Form (3NF).

Normalization Summary

Table	**1NF**	**2NF**	**3NF**
User	✔	✔	✔
Land	✔	✔	✔
LandPhotos	✔	✔	✔
LandAccount	✔	✔	✔
Labour	✔	✔	✔
Labourbank	✔	✔	✔
Machine	✔	✔	✔
MachineImage	✔	✔	✔
MachineAccount	✔	✔	✔
LandRequest	✔	✔	✔
LabourRequest	✔	✔	✔
MachineRequest	✔	✔	✔

Table 1: Detailed Summary of Normalization

4.5 Security Design

Security design involves creating measures to protect a system from unauthorized access, data breaches, and other threats. For **KrishiWala**, it ensures the safety of user data, and platform operations. Key aspects include safeguarding sensitive information like user credentials, land or machine bookings, and implementing authentication, encryption, and access control to prevent unauthorized actions. This design ensures the platform operates securely, providing a safe experience for all users. By prioritizing security from the start, KrishiWala builds user trust and ensures long-term platform reliability.

Session Management: The application ensures that user sessions are securely managed with token expiration and automatic logout after inactivity. Session tokens are stored securely in the browser's local storage.

Authentication Design: The Security Design of the application ensures data protection, authorized access, and secure interactions throughout the system, aiming to protect both user data and platform resources from unauthorized access or malicious activities.

Authentication Mechanism: The application uses a secure login system with encrypted passwords hashed using the SHA-256 algorithm. JWT (JSON Web Token) is implemented for session management, ensuring that only authenticated users can access protected resources and perform key operations like sending requests or registering assets.

Input Validation: To prevent injection attacks and ensure data integrity, all user inputs are thoroughly validated on both client and server sides. Fields are checked for proper format, length, and type before processing, ensuring the application handles only clean and expected data.

Chapter 5: Software Design Methodology

Software design is the process of planning how the software will work before building it. It helps in breaking down the big software into smaller parts that are easy to develop, test, and improve. A good design makes sure that all features are clear, the flow of data is smooth, and each part of the system works properly together. It also helps developers understand what to do, when to do it, and how each part connects with the others. A clear design saves time and reduces errors in development.

5.1 Overview of Agile Methodology

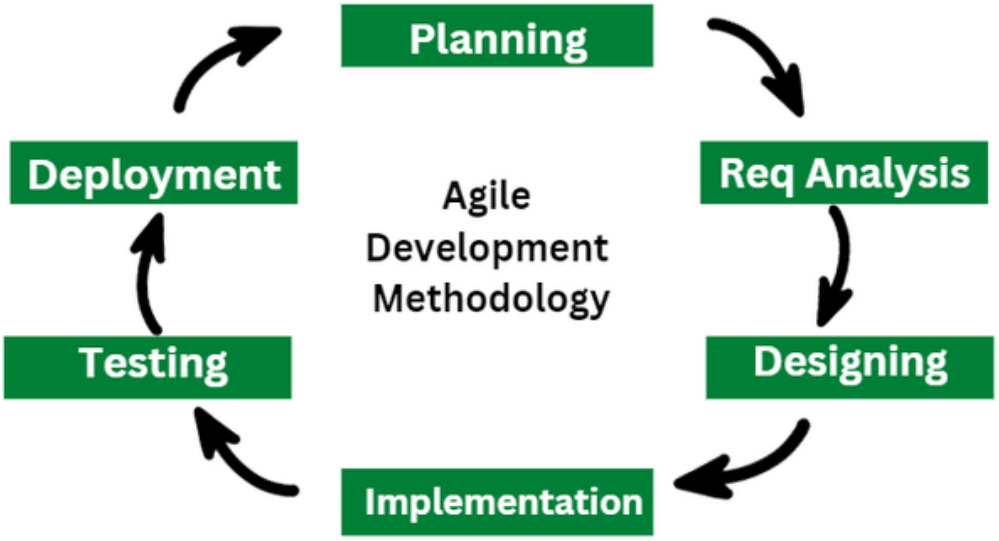


Fig 5.1: Agile Development Methodology

Agile is a flexible software development method where work is done in small steps called “sprints.” Each sprint focuses on one small part of the project and is completed in a short time (like 1 or 2 weeks). After every sprint, the team reviews the work and makes improvements. Agile helps to get feedback early and allows us to change the software easily if needed. Instead of building everything at once, Agile builds the

software slowly but with better quality. It keeps the team focused and gives value to the users quickly. It's very useful for real-world problems where requirements can change over time.

5.2 Agile Process for the Project

For the **KrishiWala** project, we followed the Agile process step-by-step. First, we tried to understand the real problems farmers face while renting land, hiring labour, and finding machinery. After that, we listed the required features like login, land renting, machine renting, labour hiring, and request management. We divided these features into smaller modules. This helped us to focus on one feature at a time instead of building everything together. Each feature became one sprint in our Agile cycle.This kept our code organized and helped avoid errors caused by working on the same files. It also made team collaboration much easier.

We also used **CI/CD (Continuous Integration and Continuous Deployment)** in a simple way. Whenever we pushed code to GitHub, our project got updated automatically. The frontend was deployed on **Vercel**, and the backend was deployed on **Render**. Both platforms pulled the latest code and deployed the updated version. This allowed us to check our project live after every small change.Using Agile and CI/CD helped us manage the project easily. We could build and test our software in steps, fix bugs quickly, and release updates without any trouble. It also helped us work as a team smoothly, even while handling different parts of the project. This method saved time and improved the quality of our final software.

5.3 Key Phases in the Development Life Cycle

The Development Life Cycle refers to a step-by-step process followed during the creation of any software project. For KrishiWala, the following are the key phases: requirement analysis, system design, implementation, testing, deployment, and maintenance. Each phase ensures that the project progresses in a structured manner, minimizing errors and improving overall quality.

i) Requirement Gathering

In this phase, we understood the real problems that farmers face in their daily life. Many farmers struggle to rent land, find machines for work, or hire labour at the right time. We talked to people and thought about how we can solve these problems using a digital solution. We also noted what features are most important, like filters, request system, user profiles, and secure login. We decided to create a platform where farmers can register their land, machines, or themselves (as labour) and connect with others. Our main goal was to save farmers' time and make the process easy. We also kept in mind that the platform should be simple to use even for people from rural areas.

ii) Planning

After collecting all requirements, we started planning how to build the project step-by-step. We broke the whole software into small modules like login system, request system, land listing, machine renting, labour hiring, and profile section. We discussed what each module should do and in what order we will build them. We also planned the database structure — what tables we need for users, lands, machines, labours, and requests. We decided on using Django for backend and React with Vite for frontend. The planning also included assigning work to each team member and setting timelines for completing each part.

iii) Designing

In the design phase, we created a clear structure for our system. We designed how users will register and login, how land and machines will be listed, and how requests will be sent. We drew rough sketches (wireframes) of each page like login page, home page, land listing, machine filter, and profile section. For backend, we designed our database schema including tables for users, lands, machines, labours, and request types. We also thought about how different parts of the system will connect — for example, how a user sends a request and how that data is shown in the profile. We kept the design simple and user-friendly.

iv) Development / Integration

This phase was about writing the actual code. We followed the Agile approach and worked on one module at a time. We first built the login and registration system using Django and created encrypted passwords using SHA256. Then we developed the land, labour, and machine modules where users can register and search using filters. We created request functionality where a farmer can send requests to landowners, machine owners, or labourers. The frontend was built using React + Vite and connected with backend APIs. We pushed the code to GitHub regularly and worked in branches to avoid conflicts.

v) Testing

Testing was an important part of our process. After completing each module, we tested it before moving to the next one. We tested the login system by checking correct and incorrect logins. We tested the land, machine, and labour modules by registering data and seeing if it showed correctly. We also tested the filter system — for example, searching land by size or price. The request system was tested to see if the right requests were sent and shown in the profile section. We fixed any bugs we found during testing. This helped us to make sure each feature worked properly before final deployment.

vi) Deployment

Once development and testing were done, we deployed our project to live platforms. The frontend (React + Vite) was deployed on Vercel, which made it easy to update and test quickly. The backend (Django + SQLite) was deployed on Render, which allowed our APIs to be used by the frontend. We connected both platforms so the whole app worked together smoothly. We also used GitHub for version control, and each push triggered updates on deployment. This made our deployment process fast and automatic. Now, users can access the project online and use all features easily.

vii) Maintenance and Updates

After deploying the KrishiWala software, we entered the maintenance phase. In this phase, we made sure that the platform continues to work smoothly without any issues.

We regularly monitored the performance of the system, fixed bugs, and improved speed and security whenever needed. We also took feedback from real users like farmers and improved the features based on their suggestions. For example, if users found any problem in the request system or had trouble filtering land or machines.

5.4 Tools and Technologies Used

In the development of this project using Agile methodology, various modern tools and technologies were utilized. For the frontend, React.js with Vite was employed to build a fast and responsive user interface, along with HTML, CSS, and JavaScript for structure and styling. On the backend, Django (Python) was used to manage server-side logic and APIs, while Django REST Framework (DRF) helped create and manage RESTful APIs. Data related to users, land, machinery, labour, and service requests was stored using SQLite. For secure authentication, JWT (JSON Web Token) was implemented for session management and SHA-256 hashing was used to encrypt user passwords. GitHub was used as the version control platform for source code management, team collaboration, and tracking changes. The React frontend was deployed using Vercel, and the Django backend was hosted on Render.

5.5 Benefits of Using Agile Methodology

Using Agile methodology in our KrishiWala project gave us many advantages. The biggest benefit was that we worked in small steps, which made it easier to manage and track progress. Each feature like login, request system, or land renting was treated as a separate part and completed one by one. Agile allowed us to test each part after building it. This helped in finding and fixing bugs early. We also collected feedback after every phase, so we could improve features based on what users needed. Our team worked together smoothly using GitHub.

Chapter 6: System Implementation

The implementation phase involves deploying the system in a real-world environment, ensuring smooth functionality, and making it accessible to users.

6.1 Frontend Implementation:

The frontend of the KrishiWala application was developed using React with Vite. The user interface was designed to be clean and simple so that farmers from rural areas can use it easily. On the home page, users can see options to search land, machines, and Labour. There are filter options like location, land size, price, period, and available facilities to make the search better. The design is responsive, which means it works well on both mobile and desktop devices. Features like **login, register, profile page, request system, Machine Registration, Land Search, Labour Search and Labour Registration** were added to improve user experience.

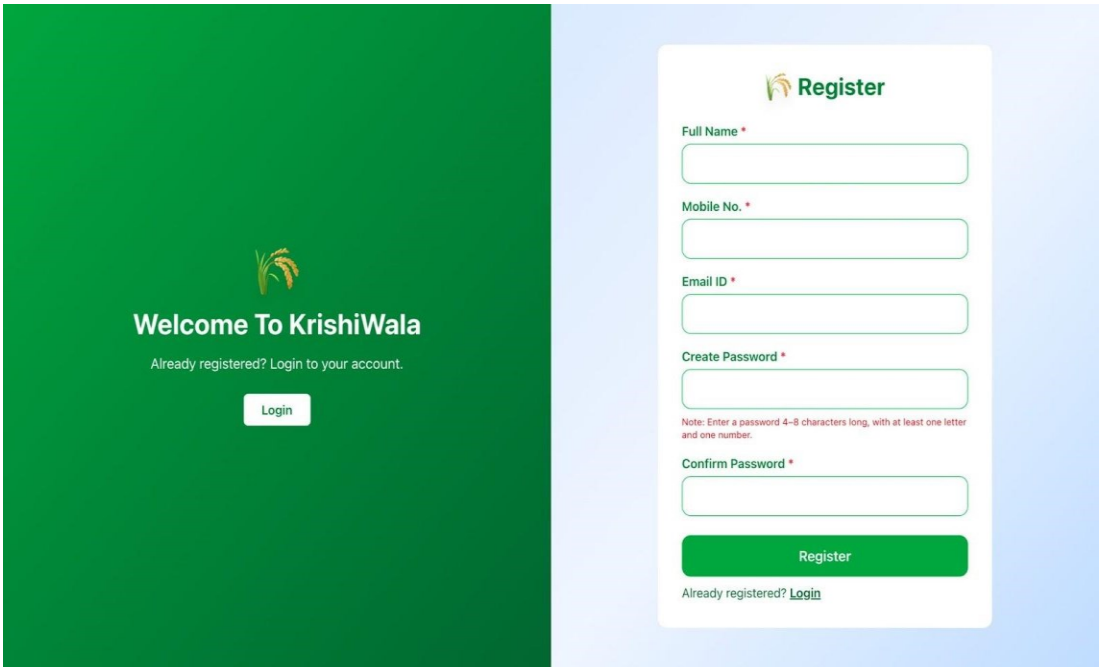


Fig 6.1: User Registration page

The **User Registration Page** (Fig 6.1) is designed to collect necessary user details for account creation on the KrishiWala platform. It includes input fields for the **Full**

Name, **Mobile Number**, **Email ID**, **Create Password**, and **Confirm Password**. Users must fill in these fields and click the **Register** button to complete the registration process. The system ensures proper validation, such as matching passwords and checking for valid input formats.

On the **left side** of the page (fig 6.1), there is a **Login section** that allows users who have already registered to sign in directly. After a successful registration, the system prompts the user to proceed to this login section to access their account and continue using the platform.

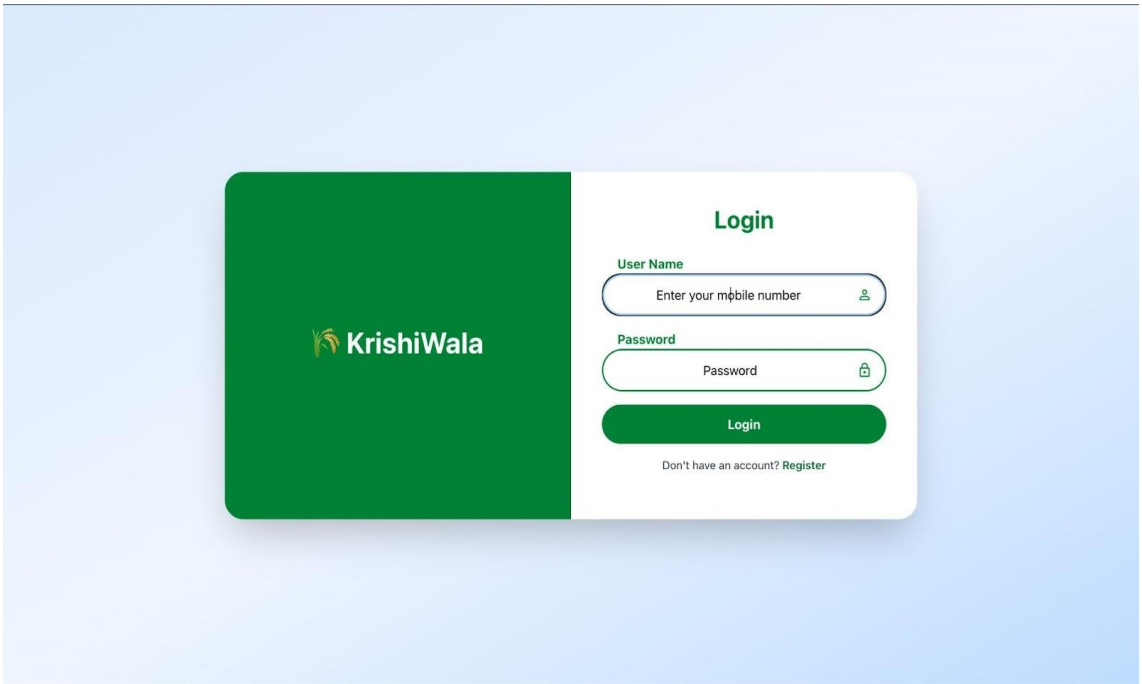


Fig 6.2: Login Page

The **Login Page** (Fig 6.2) serves as the entry point for users to access the KrishiWala platform. It includes input fields for the **user's mobile number** and **password**, ensuring a simple and secure login process. Upon entering valid credentials, the system authenticates the user. If the login is successful, the user is automatically **redirected to the Home Page.**

Fig 6.3: Home page before Login

Fig 6.4: Home page After login

Fig 6.3: Home Page (Before Login) – This page introduces users to KrishiWala, highlighting its key services like land leasing, machinery rental, and labour support. It provides general information aimed at landowners, farmers, and service providers, encouraging them to register or log in.

Fig 6.4: Home Page (After Login) – Once logged in, users see a personalized dashboard where they can browse listings, send requests, check booking status, access their profile, and interact with the chatbot. The page updates dynamically with the user's name and role for a tailored experience.

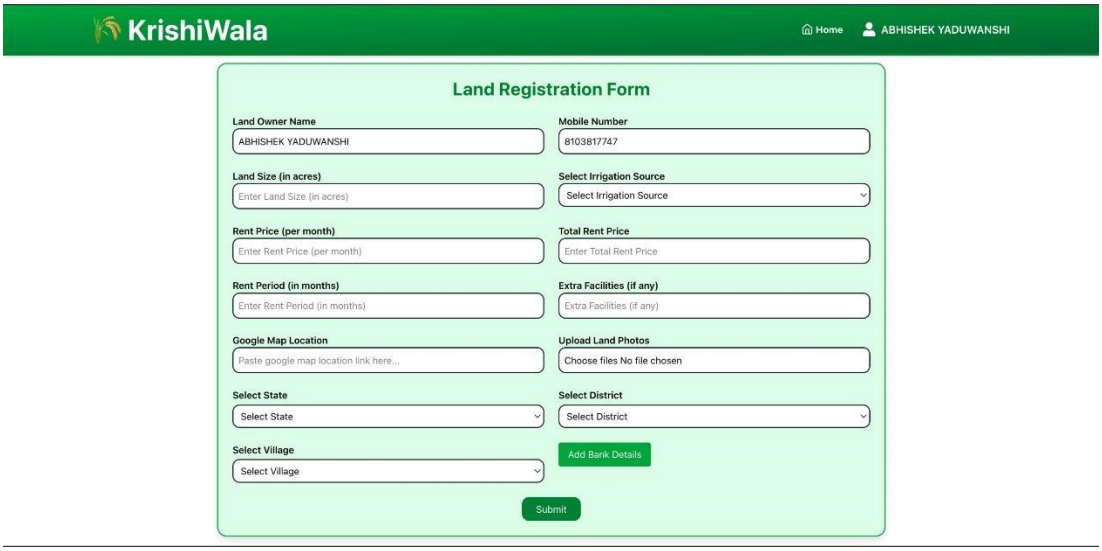


Fig 6.5: Land Registration Page

The Fig 6.5 describes the submission of a **land registration form**. It serves as a user interface for landowners to provide essential details related to their land and bank information. The form includes various fields such as landowner's name, mobile number, rent details, land location, irrigation sources, and bank details. This form also handles file uploads (land photos) and integrates multiple dynamic components, such as Select Address and Bank Details. when we talk about **dynamic components** like the **Account Number** and **Address**, we are referring to fields whose options or validation can change based on the user's input or selections.

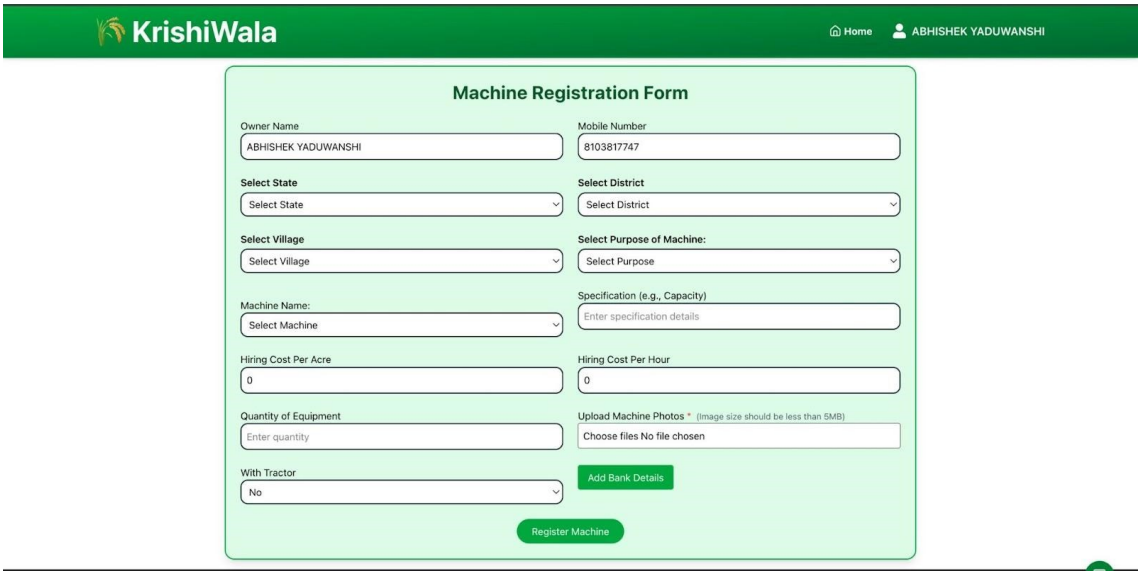


Fig 6.6: Machine Registration Form

The **Machine Registration Form** (Fig-6.6) is a user-friendly form designed for landowners to register their machinery for hire. It captures essential details such as the owner's name, mobile number, machine specifications, location, and hiring costs. The form includes fields like **Owner Name** and **Mobile Number**, where the mobile number is validated to ensure it follows a specific format. The **Location Selection** component allows users to select the state, district, and village dynamically through the **SelectAddress** component.

The **Machine Name & Purpose** are chosen from dropdowns, while the **Specification** field captures machine-specific details. If the machine is registered with a tractor, additional fields for **Tractor Company** and **Model** are displayed. Users also enter the **Hiring Cost Per Acre** and **Hiring Cost Per Hour**, both of which are validated for correct input. The **Quantity** field ensures only valid positive integers are entered. **Machine Photo Upload** allows users to upload images with size restrictions. The form dynamically adjusts based on user selections, showing additional fields where necessary. Once all fields are completed, the **Save Details** button submits the form after validating the data. If successful, a confirmation message appears, and the form data is logged; otherwise, users are alerted to any errors.

Fig 6.7: Labour Registration Form

The **Labour Registration Form (Fig 6.7)** component allows users to register their personal, work, and payment details. It collects the user's name, mobile number, age, and gender, with validation ensuring proper formatting, such as a valid 10-digit mobile number and age between 18 and 60 years. The user is required to select a location (state, district, and village), ensuring all fields are filled correctly. For work details, users can choose a work type from a list or specify "Other," and provide their years of experience, with validation for values between 0 and 40 years.

The Add Bank Details section includes input for expected wages and a choice between daily or hourly payment types. Bank details are also captured, including the bank name, account number, and IFSC code, with corresponding validation for correct formats. The user can upload a profile picture. Form validation ensures all fields are correctly filled before submission. Upon successful validation, and the user receives a success message and is redirected to the home page.

Upon successful validation, the data is securely stored in the database and the user receives a success message. The form is designed to be responsive, providing a seamless experience on both desktop and mobile devices. Error messages are clearly displayed for any invalid inputs to assist the user in correcting mistakes.

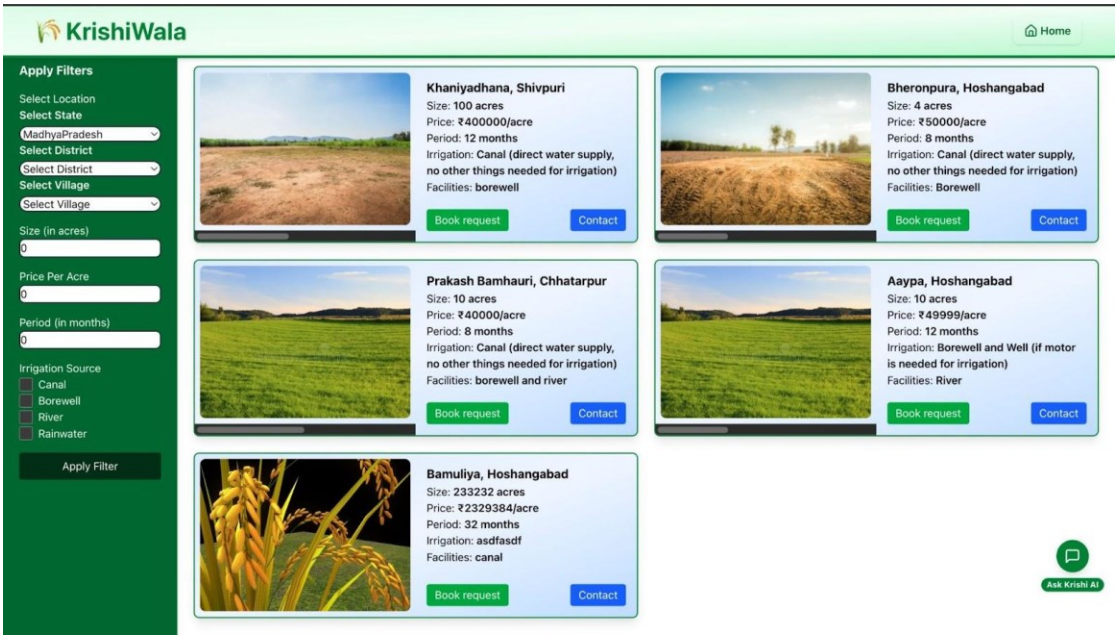


Fig 6.8: Search land Filter

The Fig 6.8 describes the **Search Land Filter,** a list of lands available for rent and provides users with advanced filtering options. Users can filter listings based on location (state, district, village), land size, rent price per acre, rental period, and irrigation source. It uses React state hooks to manage these inputs and dynamically update results. The layout includes a reusable header component for filter controls and a scrollable main section to show listings.

Each land card contains a horizontal image slider for browsing land photos. Alongside the images, it presents key details such as village name, district, land size, price, rental period, irrigation sources, and additional facilities. Two action buttons Book and Contact allow users to take immediate steps. The design is responsive and optimized for both mobile and desktop views, making it a user-friendly feature for a land rental platform. After clicking the 'Book' button, a **Booking Request for the Landowner page (Fig-6.9)** opens. It contains the Center Name, Mobile Number, Land Size, and Renting Period, along with two buttons: Cancel and Send Request. Once submitted, the request is stored in the database and sent to the respective landowner. Users receive a confirmation message. This process ensures transparency and smooth interaction between landowners and interested farmers.

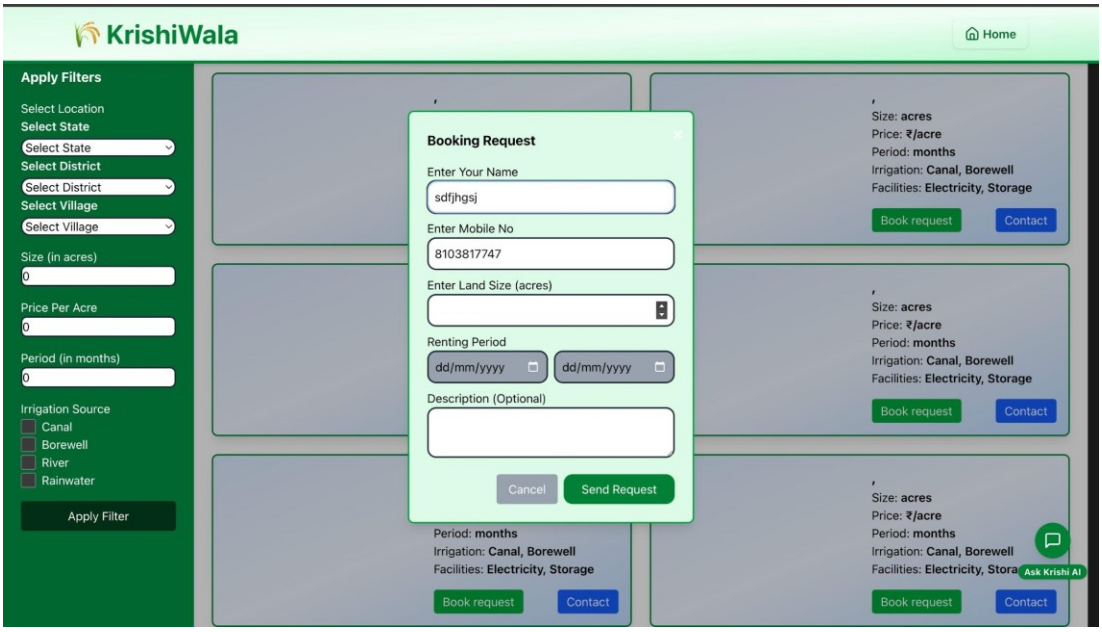


Fig 6.9 Booking Request send for Landowner

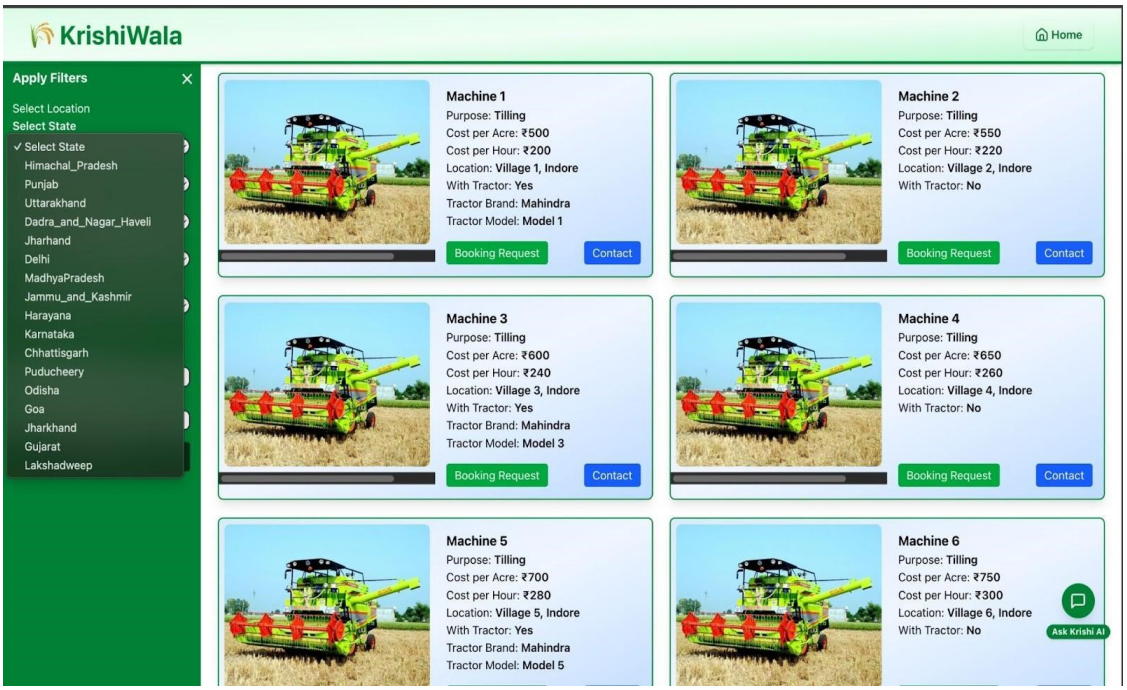


Fig 6.10: Search Machine Filter

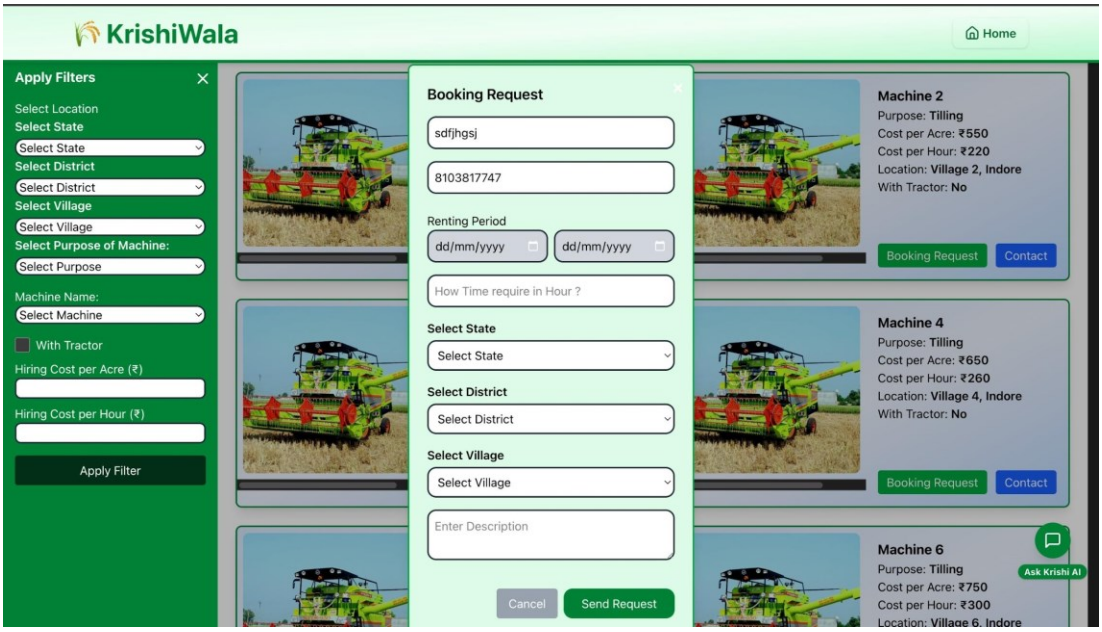


Fig 6.11: Booking Request send to Machine Owner

The **Search Machinery Filter** (Fig. 6.10) component displays a list of agricultural machinery available for hire, each with detailed information. Every machinery listing includes the **Machine Name**, **Purpose** (such as tilling), whether it comes **with a tractor**, **Tractor Brand**, **Tractor Model**, **Hiring Cost per Acre**, and **Hiring Cost per Hour**. Additionally, each entry shows the **Location**, including **State**, **District**, and **Village**, along with a horizontal image slider showcasing **Machine Photos**. Two buttons **Book** and **Contact** are provided with each listing for user interaction.

After clicking the **Book** button, a **Send Request to Machine Owner** (Fig. 6.11) page appears. This page includes input fields for the **User's Name**, **Mobile Number**, and **Renting Period**, along with dropdowns to **Select State**, **Select District**, and **Village** to specify the location. At the bottom, there are two buttons: **Send Request**, which submits the booking request, and **Cancel**, which closes the form without submitting. The form ensures that users provide all necessary details to proceed with the booking. Basic validation is implemented to prevent submission of incomplete forms. Basic validation is implemented to prevent submission of incomplete forms. All submitted requests are stored in the database for tracking and further processing.

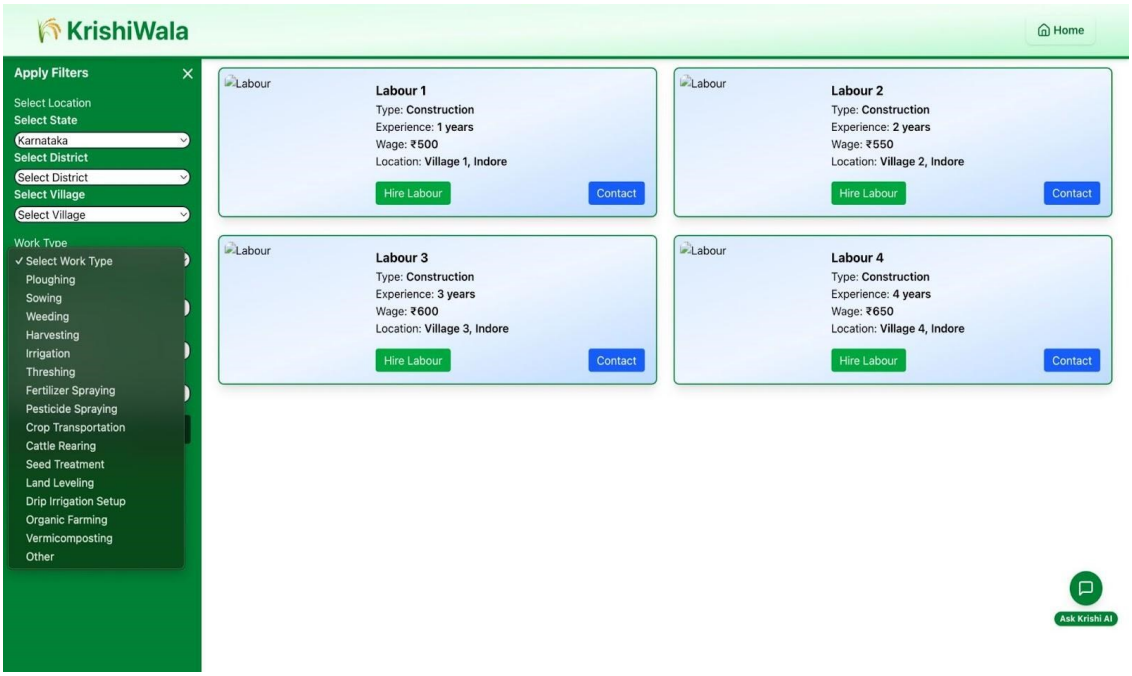


Fig 6.12: Search Labour Filter

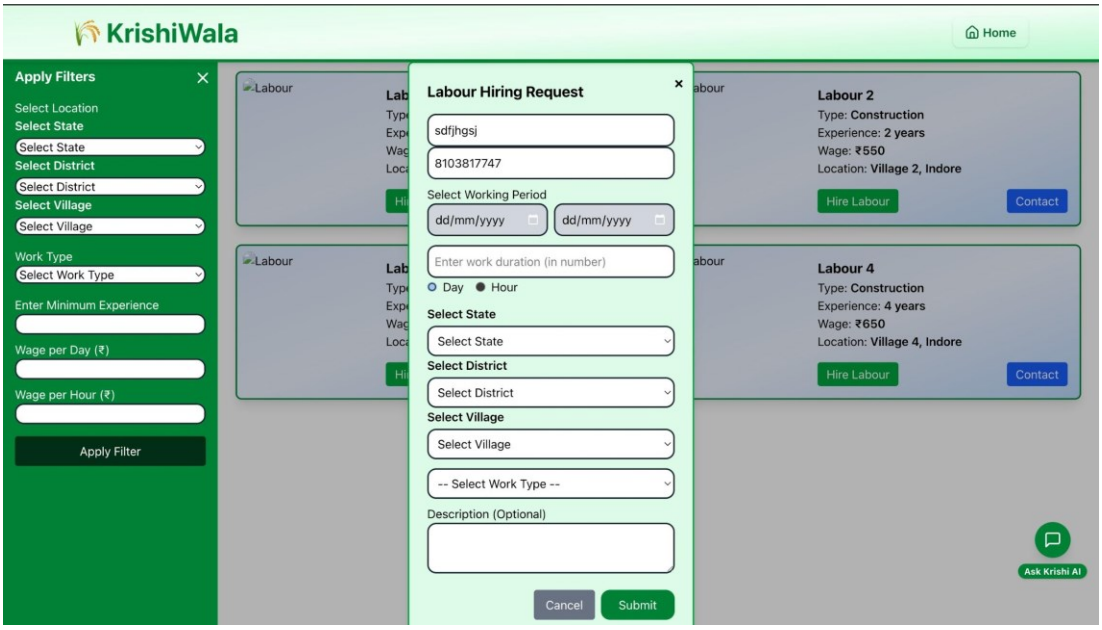


Fig 6.13: Send Request to Labour

The **Search Labour Filter (6.12)** feature allows users to find and hire labourers for various agricultural or construction tasks. Users can apply filters such as labour type, experience, daily wage, availability, and location (State, District, and Village) to narrow down the results. The page displays a list of available labourers with their photo, name, type of work, years of experience, wage rate, and location.

When a person clicks on the **Hire** button, the **Search Request to Labour** form (Fig. 6.13) appears. This form collects essential information needed to request a labourer's service. It includes fields for the user's **Name** and **Mobile Number**, followed by options to **Select Working Period** and choose whether the labour is needed **per Day or per Hour**. Users must also specify the **Address** where the work is to be done and the **Type of Work** required. At the bottom of the form, there are two buttons: **Submit**, which sends the hiring request to the selected labourer, and **Cancel**, which closes the form without taking any action.

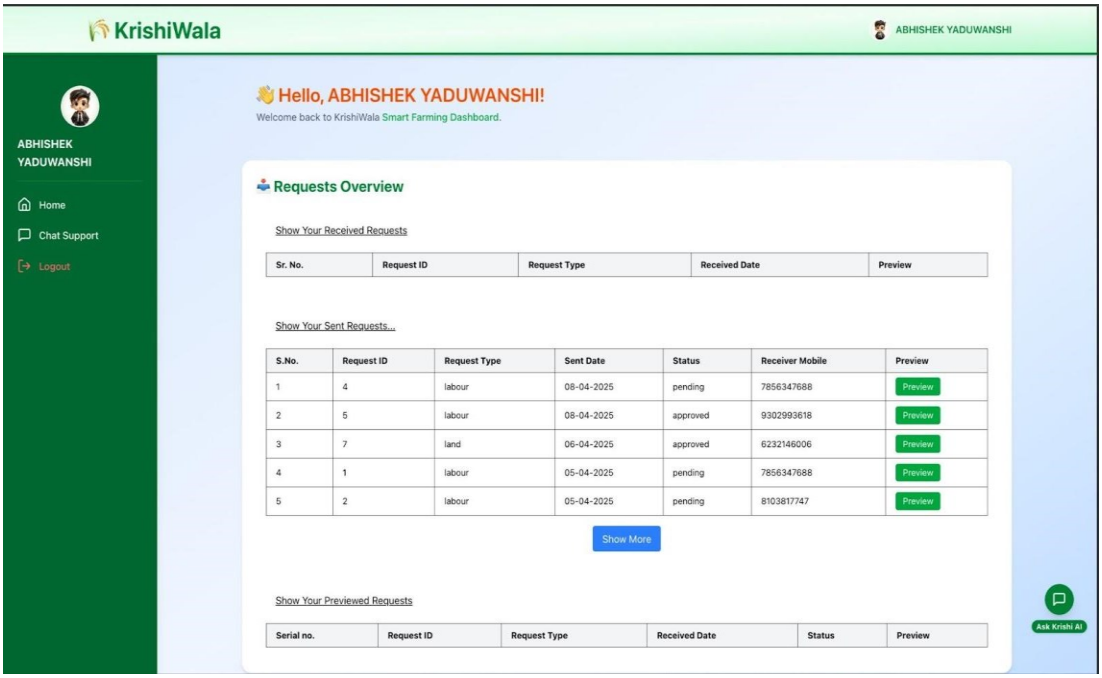


Fig 6.14: Profile Page

The Profile Page (Fig 6.14) serves as a personalized dashboard for users to manage their farming-related activities. It displays a welcome message with the user's name and offers easy navigation through features like Labour Profiles, Machinery, and Land Records. Users can view and manage their requests under tabs like Sent, Received, and Previewed. It also provides sections for Bookings, Transactions, and quick access to Chat Support, ensuring smooth interaction and tracking of services.

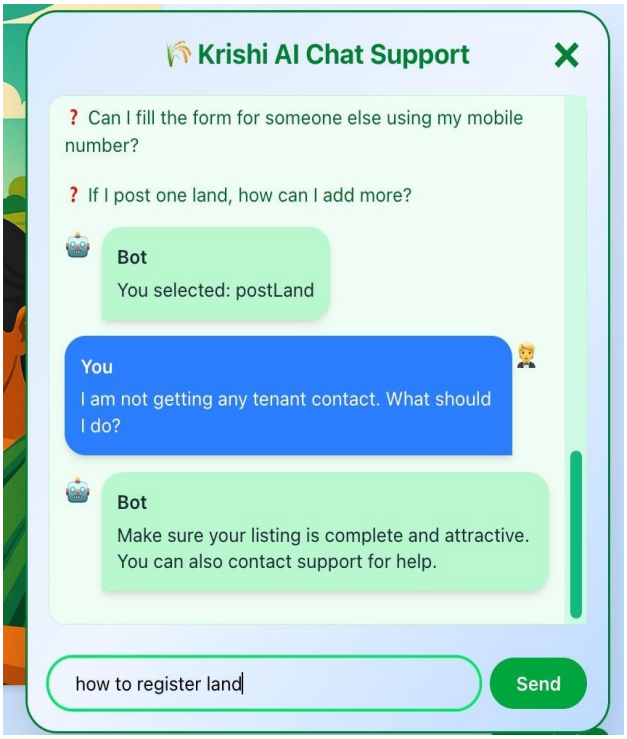


Fig 6.15: KrishiWala Chat Support

The Fig 6.15 Krishi AI Chat Support is an interactive chatbot feature designed to provide farmers with instant solutions to their queries and concerns. It appears as a chat icon at the bottom of the screen, which opens a chat window when clicked. Users are presented with categories such as Land Records, Machinery, and Labour, each containing a list of relevant questions. Clicking on a question immediately displays the answer in the chat. If the user's query is not listed, they can select the "Other" option and type their question manually. It ensures a smooth user experience with instant replies, helping users navigate and resolve issues efficiently.

6.2 Backend Implementation

The backend was developed using **Django REST Framework (DRF)**. It handles all the main operations like user registration, login, storing land, machine, and labour data, and managing request actions. After a user log in using their mobile number and password, a **JWT token** is sent for session management. This token helps the user stay logged in. All-important actions like sending or receiving a request, registering land or machine, and updating profile data are done through secure backend APIs. The mobile number and name of the user are fixed once they register. These cannot be changed later to avoid confusion and keep data secure.

6.3 Frontend Backend Integration

For the frontend, I have used **React with Vite** and styled the application using **Tailwind CSS**. The backend is developed using **Django** along with the Django REST Framework. Integration between the frontend and backend has been achieved through RESTful APIs. In the frontend, API calls are made using the fetch method to interact with the backend. On the backend side, dedicated endpoints have been created to handle each type of request appropriately. This approach ensures that both the frontend and backend remain decoupled and function independently, allowing for better scalability and maintainability of the application.

6.4 Database Implementation

The project uses **SQLite** as the database, which is the default database provided by Django. SQLite is lightweight, file-based, and requires no separate server setup, making it ideal for small to medium-scale applications and development phases. **Fig 6.13** shows the structure of the database tables used in the KrishiWala application. Each table represents a key module of the system, such as User, Land, Machine, Labour, BookingRequest, and ChatMessages. These tables are interconnected through foreign key relationships to ensure data integrity.

Fig 6.16: Database Tables

Integration with Django

SQLite is automatically integrated when creating a Django project. The default database settings in settings.py already point to an SQLite database file:

```
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': BASE_DIR / 'db.sqlite3',
  }
}
```

Model Creation using models.py

In Django, tables are created using Python classes inside the models.py file of each app. Each class represents a table, and each attribute inside the class represents a column in that table. Django's **ORM (Object Relational Mapping)** automatically converts these classes into actual SQL tables.**Fig 6.14** illustrates the content inside the database tables of the KrishiWala application. It provides a clear view of the actual data stored in each table after various user interactions such as registrations, bookings, and chatbot conversations.

user_id	name	email	password	mobile
1	ABHISHEK YADUWANSHI	abhishekyaduwanshi4852@gmail.com	pbkdf2_sha256$870000$oEGvfEKFFesQ...	8103817747
2	sachin	abhishekyaduwanshi4852@gmail.com	pbkdf2_sha256$870000$e57O1KXqSDdN...	9302993618
3	Abhi	abhishekyaduwanshi4852@gmail.com	pbkdf2_sha256$870000$8h5GrZqik5JbJ...	9926872588
6	Lalit Lowanshi	lalitlowanshi4852@gmail.com	pbkdf2_sha256$870000$hocl55qzpRJA2...	9753772831
7	ram	abhishekyaduwanshi4852@gmail.com	pbkdf2_sha256$870000$ilPmgVYVzmy8...	9876543210

Fig 6.17: Datainside Tables

Migrations and ORM Integration

Once the models are defined:

- Django generates SQL migration files using the command:

python manage.py makemigrations

- The tables are created in the SQLite database using:
  python manage.py migrate

6.5 Deployment Strategy

The deployment of the **KrishiWala** platform was carefully planned to ensure smooth functioning and easy access for users. The frontend was developed using **React with Vite**, which offers fast build times and a modern development experience. It was deployed on **Vercel**, a cloud platform that is well-suited for hosting frontend applications, offering features like continuous deployment and custom domains. The backend was built using **Django Rest Framework (DRF)**, which provides a powerful and flexible API structure. For the backend deployment, **Render** was used as it supports automatic deployments from GitHub and handles backend services well. The database used was **SQLite**, which is lightweight and comes integrated with Django, making it easy to set up and manage during development and deployment. The entire project was hosted through a **GitHub organization** to allow collaboration among team members. This strategy ensured that the system was live, accessible, and maintained properly with smooth coordination between the frontend, backend, and database layers.

6.6 Implementation Challenges and Solutions

Some of the main challenges faced during development:

- **Handling different types of filters** for land, machine, and labour: Solved by creating step-by-step filter logic.
- **Session management and data security:** Solved by using **JWT tokens** and password hashing.
- **Preventing mobile number changes after login:** Solved by disabling the update option for mobile number and name fields.
- **Profile management with multiple requests:** Solved by creating three separate request tables and showing.

6.7 Results of Implementation

After the successful implementation of the project **KrishiWala** the system effectively solved the real-world problems faced by farmers in renting land, hiring labour, and accessing agricultural machinery. The platform allowed users to register themselves and perform different actions based on their needs. Farmers were able to register their land, list their machines for rent, and offer their labour services. Other farmers could then search for available land, machines, or labour using multiple filters such as location, size, price, duration, and facilities, which made the search process easy and efficient.

The login and registration system worked smoothly using a mobile number, email, and encrypted password, with user information securely stored in the database. Once logged in, users could send rental or hire requests, and each user had a personalized profile displaying all their requests. The notification system kept users updated on request status and approvals. Admins could manage listings, verify user data, and ensure platform safety. The intuitive user interface made the system accessible even to those with limited digital literacy. Overall, the system significantly improved connectivity, efficiency, and transparency.The system also featured real-time chat support, offering instant assistance to users for any queries or issues.

Chapter 7: Testing

System testing aims to ensure that all components of the web application, including the user interface, database, and backend functionality, work together as expected and meet the project requirements.

7.1 Testing Objectives

The primary objective of testing in the KrishiWala system is to ensure that all functionalities operate smoothly and meet user expectations. Testing aims to identify and resolve bugs, performance issues, and security vulnerabilities before deployment. It ensures that user authentication, land and equipment listings, hiring processes, and payment transactions work as expected. Additionally, testing helps validate system reliability, responsiveness, and scalability, ensuring that the platform can handle multiple users simultaneously without crashes or slowdowns.

7.2 Testing Scope

The testing scope covers all critical functionalities of KrishiWala, including user registration, login, posting and renting land or equipment, hiring labor. Functional testing ensures that each module works independently and while performance testing assesses the system's ability to handle high traffic. Security testing is performed to prevent unauthorized access and data breaches. Additionally, usability and compatibility testing ensure that the platform is easy to navigate and accessible across different devices, including mobile phones and desktops.

7.3 Testing Principles

Testing principles help make the software better and more reliable. Testing can find mistakes, but it cannot prove that there are no mistakes at all. Because testing everything is not possible, testers check the most important and risky parts first.

Testing early helps catch problems sooner, which saves time, money, and effort, as mistakes usually happen in a few important areas.

7.4 Functional Testing

Functional testing verifies that each function of the software works according to the requirement specification. It focuses on user commands, data input, business processes, and expected outputs. Here are some example of Functional Testing for KrishiWala.

7.4.1 Unit Testing

Each component or module (e.g., login, land registration) is tested independently. This ensures that individual blocks of code are working correctly in isolation.

Frontend Components: React components such as login form, registration form, and dashboard sections were unit tested that ensure correct rendering, input handling, and state updates.

Database Operations: Ensured that individual operations like insert, update, fetch, and delete work as expected using Django's built-in testing features.

Backend APIs: Tested and to verify correct responses, status codes (200, 404, 500), and error handling for user login, registration, land/machine/Laboure submissions, and request APIs.

7.4.2 Integration Testing

Integration Testing aims to test the functionality of different modules when integrated together, ensuring that these modules work properly with each other. During this process, the data exchange, interaction, and dependencies between modules are tested.

1. **User Registration and Authentication Integration Testing:**

- Verify if the user registration information is stored correctly in the database.

- Ensure that once a user registers, the login system retrieves the correct credentials.

2.. Search and Filter Integration Testing:

- Ensure that the search query pulls relevant product data from the database.
- Test the filter options (e.g., category, location) to ensure they are applied correctly to search results.
- Check if search results are updated live based on selected filters.

7.4.3 System Testing

System testing for KrishiWala ensures that the entire platform, including all integrated modules like user registration, and profile management, chatSupport functions as expected. During this phase, the complete system is tested for overall performance, security, and usability. The goal is to verify that all the modules work together seamlessly and that the platform can handle real-world scenarios effectively, ensuring the system meets the project requirements and provides a smooth user experience.

7.4.4 **Alpha Testing**

Alpha testing for KrishiWala is carried out internally by the development team before the platform is released to external users. During alpha testing, the team tests all core features like registration, searching, filter, Request sending, and management to identify any significant bugs or issues. The goal is to ensure that the platform is functional, secure, and stable before moving to the next testing phase, and to address major defects or usability problems early on.

7.4.5 **Beta Testing**

Beta testing for KrishiWala involves releasing the platform to a select group of external users (farmers, buyers, and sellers) to gather feedback on real-world usage. Beta testers interact with the platform in different environments, providing valuable insights into usability.

7.5 Non-Functional Testing

Non-functional testing focuses on evaluating the non-functional aspects of the system, such as its performance, usability, security, and compliance with various standards. Unlike functional testing, which checks the system's functionality against requirements, non-functional testing ensures that the system performs well under different conditions and meets certain non-functional criteria, ensuring a seamless user experience.

7.5.1 Security Testing

In the security testing for KrishiWala, we focused on ensuring the platform's protection against potential vulnerabilities. We implemented the HS256 algorithm for secure token generation and validation, which ensures that the communication between the client and server is secure. In addition, encryption mechanisms were applied to sensitive user data, such as passwords and payment details, ensuring they are stored and transmitted securely.

7.5.2 Usability Testing

In the usability testing for KrishiWala, the focus was on evaluating the platform's ease of use and overall user experience. We tested the platform with actual users, including farmers, buyers, and sellers, to ensure that the interface was intuitive and easy to navigate. Key areas of testing included product listing, order placement, search functionality.

7.6 Featured Tested

Featured Tested refers to the process of thoroughly testing key features of a product or system to ensure they function as intended. This includes evaluating the performance, usability, and security of the features under different conditions. It helps identify potential issues or bugs that may affect the user experience or functionality. Here are some example of Featured Tested of KrishiWala, it ensure that the

highlighted or most important features are reliable and meet the expected standards before release.

Feature To Be Tested	Test Description
Logging to the system	Tests whether the user can log in with valid credentials and is redirected correctly.
Land Registration Functionality	Checks if land details submitted by the user are successfully stored in the database.
Land Search Functionality	Tests the ability to search for land records based on various filters or queries.
Form Validation	Verifies that all required fields are validated and appropriate error messages appear.
Authentication Token Storage	Checks if JWT token is received and stored in local Storage upon successful login.
Secure Password Handling	Confirms that user passwords are encrypted using SHA-256 before being stored.
Mobile Responsive UI	Ensures that the login and registration pages render properly on mobile devices.
Navigation Links	Tests if navigation (e.g., Register → Login) is functional and smooth.

7.7 Test cases:

It Describes the detailed Testing of some important modules of KrishiWala.

7.7.1 Test Case 1: Logging to the system

This case will test the login system. The test must be conducted to see if access is allowed only to the authenticated users. On Successful login, the main interface must be visible to the user.

Test case Identifier	Test Items	Input Specifications	Output Specifications	Special Procedural
TC1	Login Number field and password field.	1)Login mobile number is incorrect. 2) Login mobile is correct, but password is incorrect. 3) Login number or password is blank or both are blank. 4) Login number and password both are correct.	1)The Input specifications 1, 2 & 3 must generate an exception and ask the user to input the credentials again. The input specification 4 must show the user Main Interface.	Enter the mobile no. and password and press the login button.

Preliminary test results for test case

Test case 1	Completed/Not Completed	Result Summary
TC1	Completed.	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.

7.7.2 Test Case 2: Land Registration Functionality

This test case checks if the user can successfully register land details through the system. It makes sure the entered information is saved correctly in the backend database. It also checks if the system shows the right success or error messages.

Test Case Identifier	Test Items	Input Specifications	Output Specifications	Special Procedural Requirem-ents
TC2-1	Mobile Number Validation	Enter invalid mobile number (e.g., `12345`)	Alert: "Please enter a valid mobile number (10 digits, starting from 6-9)."	None
TC2-2	Rent Price Validation	Submit form with rent Price = `0` or negative	Alert: "Please enter a valid rent price (a positive number)."	None
TC2-3	Account Number Validation	Enter alphabetic characters (e.g., `ABC123XYZ`) in account number field	Alert: "Please enter a valid account number (9–18 digits)."	None
TC2-4	IFSC Code Validation	Enter IFSC code with invalid symbols (e.g., `ABC@#123`)	Alert: "Please enter a valid IFSC code..."	None

TC2-5	File Upload Optional Check	Submit form with all valid details but without selecting file.	Form should submit successfully if file is optional.	File upload is optional
TC2-6	Name Field Validation	Enter names with numbers or special characters (e.g. `Rahul123`).	Alert: "Please enter a valid name"	None
TC2-7	Address Fields Validation	Leave State, District, and Village fields empty.	Alert: "Please select State, District, and Village."	None
TC2-8	Irrigation Source Selection	Select default "Select" option for irrigation source.	Alert: "Please select an irrigation source."	None
TC2-9	All Fields Blank Validation	Leave all fields blank and click "Submit".	Alert: "Please Enter all details".	None
TC2-10	Successful Form Submissions	Fill all fields correctly with valid data.	Land should be registered, successfully and navigation to home ("/").	None

Preliminary test results for test case 2

Test Case ID	Description	Status	Result Summary

Test Case 2	Land Registration Functionality	Completed	The results for all input specifications have passed. No discrepancies were found between expected and actual outcomes. Data was stored and validated correctly.

7.7.3 Test Case 3: Land Search Functionality

Test Case ID	**Test Cases Description**	**Input Specifications**	**Expected Result**	**Special Procedural Requirement**
TC3- 1	Validate that header filter states update correctly.	Change state, district, village, size, priceperAcre, period, irrigation options.	Respective states should be updated and visible in UI.	None
TC3-2	Validate that header filter states update correctly.	Simulate valid API response to setresponseData().	Simulate valid API response to setresponseData() . Respective states should be updated and visible in UI.	None
TC3-3	Verify image slider renders properly.	properly Simulate land data with multiple image strings	All images render in horizontal scrollable layout within the land card.	None

Preliminary test results for test case 3

Test Case ID	Description	Status	Result Summary
Test Case 3	Land Search Functionality	Completed	The results for all input specifications have passed. No discrepancies were found between expected and actual outcomes. Data was filtered, stored, and displayed correctly based on selected search criteria (state, district, village, size, price, period, irrigation source). UI dynamically updated the filtered land listings without error.

Chapter 8: Limitations

Expert Availability:
The effectiveness of KrishiWala depends on the availability and responsiveness of agricultural experts. If expert availability is limited, farmers may experience delays in resolving their issues, which could impact their satisfaction.

Accuracy of Recommendations
The platform provides personalized agricultural recommendations, but their accuracy depends on the quality of user input and the expertise of agricultural specialists. Farmers are advised to consult agricultural scientists or local agricultural departments before making critical decisions.

Limited Scope of Advice
KrishiWala offers valuable agricultural guidance, but it may not address all the needs of farmers. Complex agricultural issues, climate change, soil quality, or specialized crops may require additional support beyond the platform's scope.

Technical Constraints
The platform's performance and usability may be affected by technical limitations such as low internet bandwidth, device compatibility issues, or server downtime. These constraints could impact the user experience and the overall effectiveness of the platform. The effectiveness of KrishiWala is also influenced by the availability of digital infrastructure in rural areas, where many users may lack access to smartphones or stable internet connections. Additionally, frequent app updates or technical bugs can disrupt user activity and reduce trust in the system. Security concerns, such as data breaches or unauthorized access, may also pose risks if robust cybersecurity measures are not maintained.

Chapter 9: Conclusion

9.1 Conclusion

KrishiWala addresses critical challenges faced by farmers, landowners, and agricultural laborers by providing a transparent and efficient platform for land and equipment rental. By bridging the information gap, ensuring fair pricing, simplifying rental agreements, and improving equipment accessibility, the system empowers farmers with reliable solutions that enhance their productivity and financial stability.

Through a user-friendly interface, KrishiWala connects landowners directly with tenants, reducing the complexities of traditional agreements while ensuring secure transactions.

9.2 Future Development

Mobile Application – Expanding the platform to mobile devices will make it more accessible to farmers, enabling them to get real-time updates, market prices, and advisory services anytime, anywhere. A mobile-friendly interface will enhance user experience and engagement.

AI-based Pricing – Integrating AI-driven pricing mechanisms will help farmers receive fair market prices by analyzing real-time market trends, demand-supply dynamics, and historical data.

Government and NGO Partnerships – Government-NGO partnerships can boost digital farming adoption by offering farmers subsidies, training, and access to beneficial schemes.

Payment Integration - users can book land, machinery and hire labours directly from KrishiWala and are able to make payment. Future updates may include wallet systems and EMI options to support small and marginal farmers in managing their finances more efficiently.

Bibliography

1. Cultivators or Agricultural Labour data in range: https://agriwelfare.gov.in/en/Agricultural_Statistics_at_a_Glance
2. React for Devopment Guide - https://react.dev/learn
3. Django Documentaion - https://docs.djangoproject.com/en/stable/
4. REST framework guide - https://www.django-rest-framework.org/tutorial/quickstart/
5. Tailwind CSS Documentaion - https://tailwindcss.com/docs
6. Office of the Registrar General & Census Commissioner, India. (2011). Complete Villages Directory - India (State/District/Sub-District Level) - Census 2011. Open Government Data (OGD) Platform India. https://data.gov.in/catalog/complete-villages-directory-indiastatedistrictsub-district-level-census-2011
7. JSON Data Structure Reference :ECMA International. (2017). The JSON Data Interchange Syntax (ECMA-404). Retrieved from: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/
8. Categories of Farmaers based on Land holdings: https://pib.gov.in/Pressreleaseshare.aspx?PRID=1562687