Dr. Sudha P

Dr. Sudha P - report2.pdf

Quick Submit

Quick Submit

Presidency University

Document Details

Submission ID
trn:oid:::1:3428894046

Submission Date
Dec 1, 2025, 9:24 AM GMT+5:30

Download Date
Dec 1, 2025, 9:31 AM GMT+5:30

File Name
report2.pdf

File Size
1.5 MB

80 Pages

16,909 Words

100,292 Characters

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

Chapter 1
INTRODUCTION

While agriculture remains the backbone of many developing countries, it continues to be the source of struggle for farmers who have to settle for prices that are not commensurate with the quality of their produce. The issue of market access has remained the major challenge for these farmers, as well as the involvement of multiple middlemen who have been siphoning their profits. By the time digital and mobile-based agricultural solutions come to the rescue, farmers are still grappling with issues of communication, transparency, and access to real-time market information. This is because most of the existing platforms have not fully integrated the necessary features such as live market pricing, AI-based crop recommendations, weather analytics, and multilingual accessibility. The objective of the project «Namma Raitha» is to develop a mobile and web-based application that links farmers directly with retailers so that they can get fair and transparent pricing. The app offers features such as real-time market price data, AI-driven crop and disease prediction, and multilingual support to enable farmers make the right decisions. The necessity of such a platform cannot be overemphasized as it goes beyond the simple removal of middlemen to the empowerment of farmers through data-driven insights that will lead to higher productivity and profit. In contrast to the former works that concentrated on separate issues such as market access or crop monitoring, Namma Raitha is a single platform that merges all these functionalities, thus, facilitating the digital transformation of the agricultural sector, fairness, and efficiency.

1.1 Background

Agriculture is still the leading source of livelihood for a big part of the population in developing countries. But, farmers do not get good prices for their products most of the time because they have a limited access to markets and there are many intermediaries who take a large share of the profits [1].

The latest innovations in digital technology have come up with mobile-based agricultural solutions that facilitate transparency, communication, and access to real-time market data. These instruments enable farmers to make right decisions and to have direct dealings with

buyers [2], [5]. However, the majority of current platforms do not have such advanced features as AI-powered crop suggestions, up-to-the-minute pricing, and language flexibility.

In order to solve these problems, the "Namma Raitha" project was initiated, a mobile and web platform that connects farmers directly with retailers, thus giving them the right price and transparency. It uses AI-based disease detection [10], live market data from government sources [13], and multilingual accessibility to not only facilitate the digital transformation of agriculture but also to make farmers the recipients of data-driven insights.

1.2 Statistics

Around 43% of the population in India is engaged in agriculture, which also contributes about 18% to the GDP [1]. Despite this, a majority of farmers still use the services of middlemen to sell their farm products and thus lose 15–30% of their profits due to unjust pricing and lack of direct marketplace access [2]. Research indicates that almost 55% of farmers do not know the daily market prices, which results in them making poor income choices [3].

Such issues give rise to the necessity of the “Namma Raitha” project. Farmers need a digital solution that can provide them direct connectivity with retailers, real-time market data, and AI-based crop insights. This platform makes sure that pricing is fair, the farmers' dependence on intermediaries is lessened, and data-driven decision-making is encouraged to enhance the farmers' income and sustainability.

1.3 Prior Existing Technologies

Several mobile applications are introduced as tools that could possibly open the door for farmers to have direct access to markets and make their work more efficient. The projects of Kishan et al. [1], Bhavani et al. [2], and Tati et al. [3] are among such works that have provided platforms for crop listing, transport booking, and fertilizer access. In addition to these, solutions like AgroTIC [5] and eKichabi v2 [6] have introduced features such as crop monitoring and analytics. Apps made with Flutter, Firebase, and local languages [7] have been instrumental in ensuring that more people are reached, whereas the use of machine learning models like Lightweight CNNs and PDSE-Lite [12] has been made for crop recommendation and disease detection.

Nevertheless, none of these platforms have combined local pricing, AI analytics, weather data, and language support into one application — a space that "Namma Raitha" is occupying.

1.4 Proposed Approach

Aim of the Project:

The main goal of the "Namma Raitha" project is to create a web and mobile platform that connects farmers directly with retailers without any intermediaries thus providing fair and transparent pricing. The system further intends to equip farmers with real-time market data, AI-driven crop and disease prediction, and multilingual access for their convenience and profitability.

Motivation:

Quite often, farmers incur financial losses as a result of unjust pricing, limited market access, and being unaware of the prevailing rates of their commodities. These problems are what led to the idea of creating a digital platform which would allow farmers to be empowered in selling their products directly, monitor prices, and gain access to intelligent agriculture insights. The motivation is to increase farmers' income, promote digital inclusion, and modernize the agricultural supply chain.

Proposed Approach:

Their proposed scheme amalgamates instant market data from verified governmental APIs, AI-supported plant disease identification via image processing, and soil information-based crop recommendation. The platform created with React Native at the front end and FastAPI at the backend makes sure there is smooth interaction between farmers and retailers. Besides this, the system facilitates negotiations, order tracking, and multilingual support (English, Kannada, Tamil, Telugu, Hindi) for ease of use by the majority.

Applications of the project

• **Direct Market Access:** Facilitates farmers to sell their products directly to retailers thus cutting the middlemen.

• **Smart Agriculture:** Delivers weather, soil, and crop suggestions through AI-based analytics.

• **Disease Detection:** The image-based prediction technology assists farmers in the early identification of crop pathogens.

• **Fair Pricing System:** Provides real-time and transparent price determinations.

• **Multilingual Platform:** Farmers from different linguistic backgrounds benefit more from the platform

Limitations of the Proposed Approach:

Needs internet access that might not always be available in quite remote and rura areas.There could be costs associated with the initial installation of smartphones and the use of data which might be a challenge to some farmers. The accuracy of AI-based predictions will be limited by the quality and the volume of input data (for example, soil or plant images). Can be less scalable if the government does not continuously update the data or if there is no integration with real payment gateways.

1.5 Objectives

1.Behavior:

Develop mobile and web interfaces with a friendly user experience enabling farmers and retailers to interact, list products, bargain for prices, and finalize transactions instantly.

2.Analysis:

To build and integrate AI-driven models that interpret soil nutrient data (N, P, K, pH), weather, and crop images for providing the right crop recommendations and accurate plant disease identification.

3.System Management:

Design a centralized backend system with FastAPI and Supabase for effective data storage, fetching of market data, and syncing of farmer-retailer conversations across various gadgets.

4.Security:

Install safe authentication and role-based access control to secure the user data, guarantee the privacy of transactions, and uphold the trustworthiness of all the communications between the users and the server.

5.Deployment:

Place the system on cloud infrastructure to maintain its ability to handle increased users or work over different times, be accessible through multiple platforms (Android, iOS, and web), and have a stable performance under real-time data loads.

1.6 SDGs

The "Namma Raitha" initiative is very much in harmony with the United Nations Sustainable Development Goals (SDGs), as it encourages inclusive growth, sustainable agriculture, and digital innovation. By increasing farmers' income, the system being fair, making agriculture more efficient, and giving access to technology- driven solutions, the system is a means to accomplish several SDGs.

Fig 1.1 Sustainable development goals

1.SDG 1 – No Poverty

By eliminating intermediaries, the project facilitates farmers to access the market directly, thus, by increasing their income, rural poverty levels are reduced. UNDP (2024) states that fair pricing and inclusive digital access are the most important factors in achieving the poverty reduction targets [1].

2.SDG 2 – Zero Hunger

The platform with the help of AI-based recommendations and disease detection enhances the productivity to achieve the goal of zero hunger and also decreases crop loss. This is in line with food security and sustainable agriculture, which is a direct contribution to SDG Target 2.3 - to double the agricultural productivity and incomes of small-scale food producers [1], [2].

3.SDG 8 – Decent Work and Economic Growth

The project opens up new digitally-driven market possibilities for farmers and retailers, thus, leading to the promotion of entrepreneurship as well as the economic growth of the agricultural sector. With the help of transparency and efficiency, the project is in compliance with SDG 8.2, which refers to innovation-driven productivity [1].

4.SDG 9 – Industry, Innovation and Infrastructure

"Namma Raitha" employs cloud-based deployment, FastAPI backend, and AI/ML technologies which are the elements of digital innovation and infrastructure development in the rural areas as mentioned under SDG 9.c - technology is more accessible and available [1], [3].

5.SDG 12 – Responsible Consumption and Production

The platform through direct connections between farmers and retailers reduces food wastes, facilitates local trade, and encourages the use of efficient supply chains, which is in line with SDG 12.3 that is concerned with the reduction of food loss all over the world from production and supply chains [1].

1.7 Overview of project report

Chapter 1 of the "Namma Raitha" project report gives an overview of the project, detailing its background, objectives, motivation, the reason for the need, and even how the project aligns with the United Nations Sustainable Development Goals (UN SDGs).

Chapter 2 offers an extensive review of the related literature, summarizing the

previous researches and the existing technologies concerning the digital agriculture, direct market access, and AI-based crop prediction systems.

Chapter 3 depicts the system architecture and methodology with the help of the application design, the modules, and the workflow of the proposed app.

Chapter 4 deals with the implementation and the technologies used, as the technology stack for frontend and backend integration is described by mentioning tools such as React Native, FastAPI, and Supabase.

Chapter 5 is about the results and evaluation where it is covered the performance of the model, the user interface outcomes, and the system efficiency analysis.

Chapter 6 contains a lot of information about the project budgeting and the resources required. It includes software, hardware, and maintenance costs.

Chapter 7 is about the system applications, limitations, and future modifications, and Chapter 8 is the last one. It summarizes the report and suggests ways of extending the project for a wider agricultural digital transformation.

Chapter 2

LITERATURE REVIEW

2.1 E. S. Kishan, H. B. Tarun, and M. S. Prasad, "Mobile App for Direct Market Access for Farmers," *International Journal of Innovative Research in Technology*, 2025.

This paper introduces the concept for a mobile application to connect farmers and retailers directly, eliminating middlemen in the agricultural supply chain. The system provides the farmer with options to list their crops and allows the retailer to view and directly purchase them. The main emphasis is placed on making the process more transparent, maintaining more equitable prices, and generally easing farmer–retailer transactions. It lays a platform for many later works on digital agricultural platforms and motivates the need for features like real-time pricing and communication.

2.2 A. D. Bhavani, M. S. Varshini, P. P. Reddy, and V. C. Varshith, "Mobile Application for Direct Market Access for Farmers," *International Journal of Engineering and Management Sciences*, 2025.

This work proposes another mobile platform with similar goals, which is to help farmers access real-time market information and sell their produce directly. It emphasizes crop listing, the visibility of market prices, and interaction in a user-friendly manner. The paper underlines how digital access to price information enables farmers to make better selling decisions. Still, advanced functionalities of the system, such as machine learning mechanisms or automated recommendations, are not proposed.

2.3 T. S. Tati, O. K. Landage, and R. K. Sangle, "Mobile Application for Farmer to Get Direct Access to Market, Transport, Fertilizer Shops," *International Journal for Research in Applied Science and Engineering Technology*, 2025.

The paper extends the basic model of market access by incorporating other services such as transport information and locations of fertilizer shops, addressing the major logistical challenges that farmers face. The broader scope of the system helps farmers not only to sell produce but also to manage the other essential pre-harvest and post-harvest activities. It still misses the features which are predictive or intelligent.

2.4 K. Sughasini, R. S. Gowda, and P. M. Reddy, "Mobile App for Direct Market Access for Farmers," *Journal of Emerging Technologies and Innovative Research*, 2024.

This paper proposes a mobile application that extends existing market-access systems to add crop-monitoring and data-analysis capabilities. It supports farmers in monitoring their crops and making simple analytics-based decisions. While this paper does include more analytical functionality than many of the earlier works, it does not include any advanced machine learning models or integrated environmental data, such as soil or weather information.

2.5 J. P. Gomez, A. M. Lozano, and C. A. Lopez, "AgroTIC: A Mobile Application for Crop Monitoring and Direct Market Access," *arXiv preprint*, 2023.

AgroTIC combines crop monitoring with a marketplace platform, thus reducing the fragmentation of agricultural tools. It facilitates farmers in assessing the status of their crops and selling their produce on the same platform. The paper acknowledges that unified agricultural applications are much-needed; however, it lacks ML-based disease detection and dynamic pricing.

2.6 R. C. Mwakalinga, A. C. Kaijage, and P. E. Mushi, "eKichabi v2: A Dual-Platform Mobile Agricultural Directory for Farmers in Tanzania," *arXiv preprint*, 2024.

This paper describes a dual-platform system, presumably web and mobile, for the farmers of Tanzania. In general, this is an agricultural directory that assists farmers in finding the most useful services and information for them. It emphasizes accessibility and reach, particularly to those areas that are remote. The work does not incorporate predictive analytics nor crop-specific decision-making.

2.7 R. Vyas, R. Rameja, V. Arora, and V. Sahu, "A Mobile Platform for Direct Market Access to Farmers Using Flutter and Firebase," *Geetanjali Institute of Technical Studies*, India, 2024.

The following work is dedicated to the development of a market-access application using contemporary development technologies: Flutter-frontend and Firebase-backend. It emphasizes ease of development,

platform independence, and cloud-based connectivity. The functionality of the system is still confined within marketplace functions and does not include ML or environmental intelligence.

2.8 K. T. G. Kumar, G. K. Abhishek, and P. G. K. Karthikeya, "Android App for Farmers to Sell Crops in Regional Language," *International Research Journal of Engineering and Technology*, 2020.

This paper proposes a regional language-based agricultural marketplace app targeted at improving accessibility for farmers unaware of English. The focus of this application is on usability and local-

language support. While advantageous for inclusivity, the system provides only basic features and lacks any analytical or intelligent components.

2.9 S. M., R. G. S., S. M. Holla, P. S., S. Prabhanjan, and S. C. Sumana, "Android Application on Agricultural Marketing," *International Research Journal of Engineering and Technology*, 2020.

This work focuses on the design of an Android-based agricultural marketing platform to assist farmers in marketing their crops digitally. It allows for communication between buyers and sellers, although real-time data, market intelligence, or machine learning models are not integrated.

2.10 Summary of Literature Reviewed

Table 2.1 presents a comprehensive summary of the key findings, methodologies, and limitations identified in the reviewed literature.

Table 2.1 Summary of Literature Reviews

Reference	Focus Area	Key Findings	Accuracy/ Performance	Limitations

Kishan et al. (2025)	Direct Market Access for Farmers	Designed a mobile app enabling farmers to directly list crops and connect with retailers, reducing middlemen.	Improves transparency and direct transactions.	No ML features; lacks weather/soil/disease analysis
Bhavani et al. (2025)	Market Price Visibility & Crop Listing	Provided real-time market price insights and simple crop listing features to help farmers make informed decisions.	Effective for visibility and awareness.	No predictive intelligence; basic functionality only.
Sughasini et al. (2024)	Crop Monitoring + Market Access	Added crop-monitoring and analytical features to a market-access app.	Supports basic analytics for agricultural decisions.	Does not use ML for disease detection or recommendations.
Tati et al. (2025)	Market + Transport + Fertilizer Shop Access	Integrated transport and fertilizer shop info along with market access, supporting logistics needs.	Enhances information access for logistics planning.	No decision-support; does not include ML or automated pricing.
Gomez et al. (2023)	Integrated Crop Monitoring & Market Access (AgroTIC)	Combined crop monitoring with direct market access in a single platform.	User convenience and integrated management.	No ML integration; no dynamic market prediction.
Mwakalinga et al. (2024)	Agricultural Directory (Tanzania)	Dual-platform directory enabling farmers to find agricultural services.	Enhances accessibility in rural regions.	Enhances accessibility in rural
Vyas et al. (2024)	Flutter-Firebase Farmer Market Access	Developed using modern frameworks for smooth UI and scalable backend.	Easy deployment and cross-platform support.	Lacks ML/AI features; limited to marketplace tasks.
Kumar et al. (2020)	Regional-Language Farming App	Enabled crop selling in regional languages to improve accessibility for non-English-speaking farmers	Improves usability for diverse users.	Limited features; no data intelligence.

S. M. et al. (2020)	Agricultural Marketing App	Basic Android application for enabling digital crop marketing and buyer–seller communication.	Helps users communicate and trade digitally.	No real-time data, prediction, or analytics.

2.11 Identified Gaps and Research Opportunities

A survey of literature regarding farmer-retailer mobile platforms exposes several significant gaps that influence their effectiveness in actual agricultural markets negatively. In most cases, the systems that have been developed are merely marketplace functions that allow for crop listing, buyer-seller communication, and market price visibility, yet there is no support integrated for data-driven decision-

making. Out of all the applications that have been reviewed, not a single one has an essential components combination such as dynamic price recommendation, weather-based insights, soil-nutrient-based crop suggestions, and image-based disease detection in one single platform. So this gap is significant, farmers have to use different apps or external sources of knowledge to access the market.

In addition, a few applications do not have language options and hence can only access farmers who speak the languages of their choice. Predictive intelligence is a further significant gap noted in the research of mobile farmer-retailer platforms. Systems for open markets and logistics have been proposed in the studies; however, they do not conceive any machine learning-based models for disease diagnosis, yield estimation, or personal recommendations. Furthermore, many of the proposed systems do not offer real-time API integration for weather and market data, which leads to outdated or static information that is not very helpful for decision-making. From a technological perspective, some research works highlight user interface functions and do not talk about backend scalability, performance testing, or advanced analytics pipelines that are necessary for a large-scale deployment.

These constraints provide a wide array of research possibilities. Through these means, upcoming systems can integrate various sources of data — market datasets, weather APIs, soil nutrients, image-

based diagnostics — into a single decision-support ecosystem. On-device inference, lightweight CNNs, and ensemble models could potentially open the way for the crop diseases to be detected in real-time. The borders of existing research are extended further by improving models for pricing prediction, negotiating workflows, and introducing blockchain-based traceability to build up trust in agricultural supply chains. Lastly, the demand for multilingual and region-adaptive platforms that cater to the farmers' linguistic and socio-economic backgrounds is quite high. By responding to these issues, one can come up with intelligent, scalable, and holistic farming applications, such as the Namma Raitha system, that are described in the project.

Chapter 3

METHODOLOGY

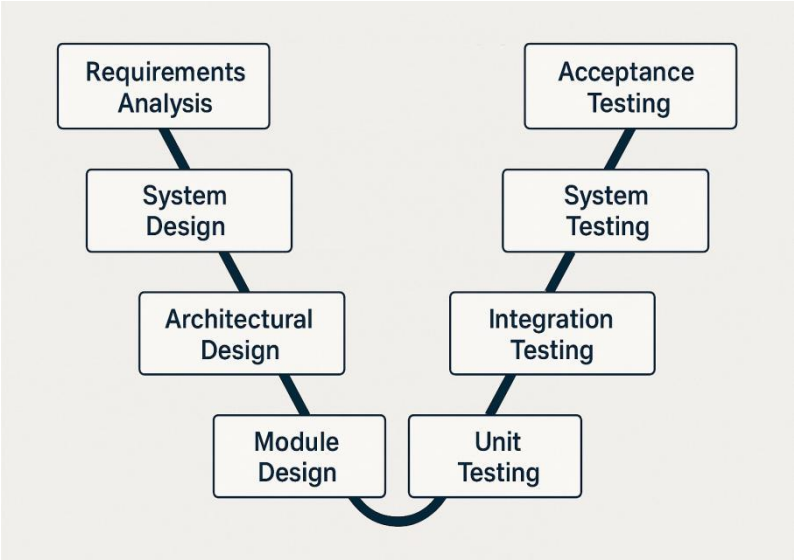


Fig 3.1 V-Model Phases

The methodology that is used to develop the Namma Raitha app is the V-Model, which is a very structured way of doing things, always checking what has been done, and it is also a very systematic way of validation for each step of the project. It is a very fitting model for this project as the system comprises many modules that are interrelated—market price integration, weather APIs, soil-based recommendations, and CNN-based plant disease

detection—hence each of them requires having the test planned beforehand and validation to be done in parallel.

1. Requirements Analysis

During the time of the first phase, analysis of functional and non-functional requirements was done based on the problem statement, literature review, and analysis of already present agricultural app. Some of the most important requirements were direct farmer-retailer connectivity, dynamic pricing using market data from data.gov.in, weather-based decision support, soil-nutrient-based crop recommendations,

plant disease detection using image inputs, multilingual support, and an order-negotiation workflow. These requirements become the base for system, software, and module design.

2. System Design

The system design phase was about the overall architecture of the app being defined. The PDF explains that the solution is divided into three main modules:

- Farmer Module: listing of crops, price recommendation, disease prediction, weather insights, soil-based recommendations, negotiation.
- Retailer Module: crop browsing, cart management, price negotiation, order placement, tracking.
- Backend & Analytics Module: API integration, database management, authentication, and machine learning inference.
- This architecture is giving a guarantee of the separation of the modules, their scalability, and also the integration with no problem that can be penetrated between the frontend, backend, and ML components.

3. High-Level Architecture Design

The focus of high-level design is mainly on the technologies and components that will be used:

- Frontend: React Native (Expo) for cross-platform mobile development
- Backend: FastAPI for API routing and data processing
- Database & Storage: Supabase for authentication, crop data, and media handling
- ML Model: CNN-based plant disease detection model trained on the Kaggle Plant Disease Dataset
- External APIs:data.gov.in for daily market commodity prices

4. Detailed Functional and Module Design

The design elaborated the internal functionalities of each module:

- Farmer Module Design
- Crop upload and image input
- Dynamic price calculation (+₹5 margin)
- Soil recommendation logic based on N, P, K, pH
- Weather advisory integration

5. Coding Phase

The coding phase was about implementing all parts of the program by using the chosen technologies. The mobile user interface was created with the help of React Native, backend operations were done with FastAPI, and database plus storage functions were performed by Supabase. A CNN disease identification model was set up as an API endpoint and was made efficient enough for mobile inference (≈0.8 seconds per prediction).

6. Unit Testing

Testing was done for each module separately:

- Farmer module: price recommendation, weather data retrieval, disease detection accuracy, soil-based suggestions
- Retailer module: item search, cart functions, payment simulation
- Backend: API response times (<250ms), model inference speed, authentication

- Unit testing was a point in the process where the accuracy of every single system element was confirmed prior to bringing them together.

7. Integration Testing

Integration testing confirmed the interaction without a hitch between the various parts:

- Farmer → Backend → Retailer workflows
- ML model → API → Frontend data rendering
- Weather and market data → pricing workflow
- Order management → notification updates

8. Testing

Tests of the full system were done to measure performance, usability, and trustworthiness. The team used model testing (98% accuracy) along with API performance checks, and UI/UX testing for the support of various languages to assess the system.

9. Acceptance Testing

By acceptance tests, the system was confirmed to have met the initial demands such as direct farmer-retailer connectivity, reasonable pricing, disease detection, weather and soil

integration, and complete order workflows. Hence, the platform was considered ready for deployment in the real world.

Fig 3.2 Summary of various methodology

The Figure 3.7 illuminates six significant project management strategies.

Waterfall is a rigid, sequential method where each stage—analysis, design, development, testing, and deployment—occurs one after another, thus it is a method that can be used for projects with fixed requirements.

Agile is an adaptive method, working in small, repeated cycles where teams plan, develop, test, and improve continuously.

Scrum, a form of Agile, breaks down the work into short sprints with regular planning, reviews, and retrospectives to get feedback and keep improving.

Kanban employs a visual board to monitor tasks through the stages like “To Do,” “In Progress,” and “Done,” thus facilitating the workflow management for the teams to be more efficient.

Extreme Programming (XP) is centered on continuous testing, user stories, teamwork, and frequent small releases as a way to keep the product of high quality.

Lean is a method for getting rid of waste, facilitating flow, and delivering value quickly through concentrating solely on activities that benefit the user.

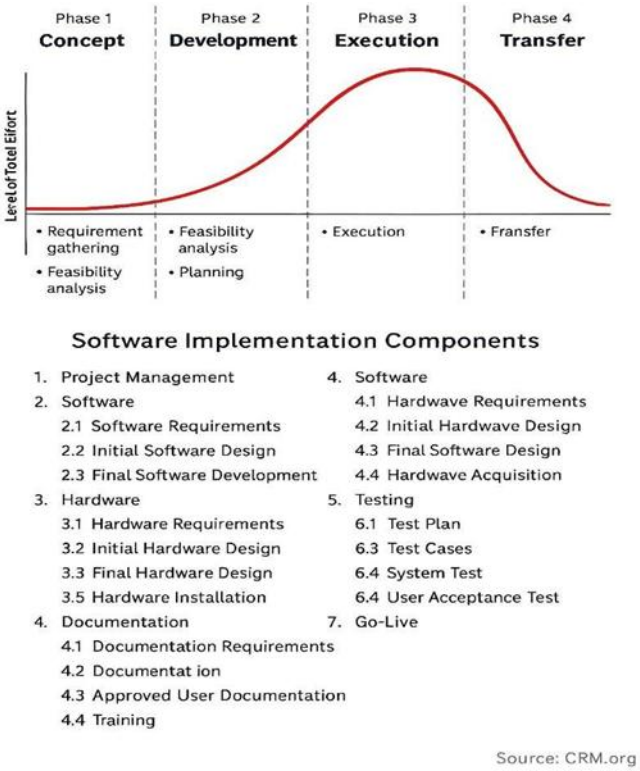


Fig 3.9 Summary of project breakdown to task

The Fig 3.9 details the software implementation journey through a series of four different phases.

The description of the path through the software endeavor starts with Phase 1: Concept. As part of this team gathers requirements and performs feasibility analysis. The purpose and the practicality of the project are comprehended.

In Phase 2: Development stages of the project get more precise. The team works on feasibility, design ideas, and project structure to have a clear understanding of the way they should follow. The amount of work is increased here.

Phase 3: Execution is the stage where the most work is done. The software and hardware development, integration, and testing take place here. The work done at this time is at the highest peak on the graph, as indicated by the highest level of engagement.

At Phase 4: Transfer, the system that has been completed is given to users. Apart from deployment, user training, and documentation, final acceptance is also part of this stage. The work is gradually decreasing as the project comes to an end.

The bottom list illustrates the major implementation components that span across project management, software and hardware design, documentation, testing activities, and the final go-live step.

Chapter 4

PROJECT MANAGEMENT

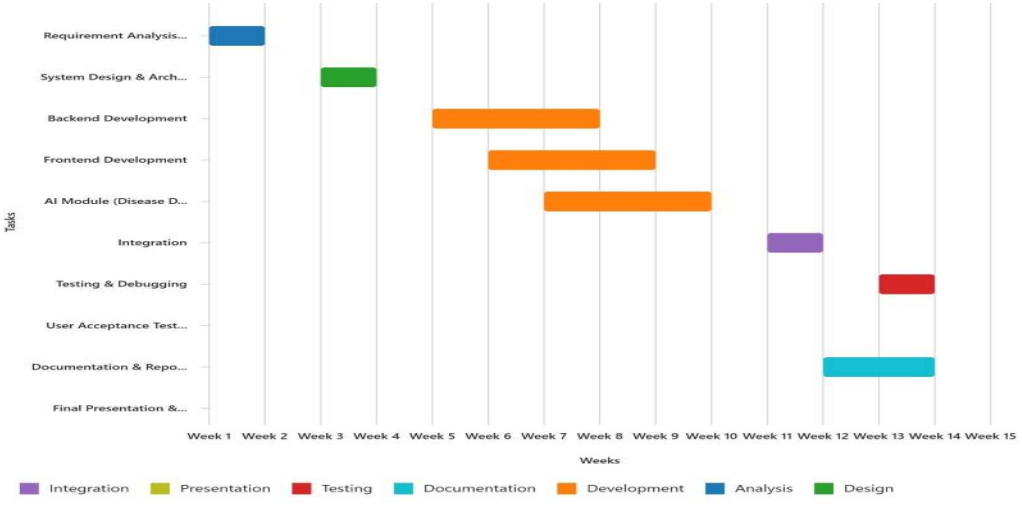
4.1 Project timeline

The Gantt chart in the uploaded picture is a comprehensive visual timeline of the whole project that effectively depicts each task along the week-wise schedule. It lays out the sequence of activities from Requirement Analysis in Week 1 to Final Presentation in Week 15. Every task is depicted as a horizontal bar indicating its start and end weeks, along with colour coding for different categories such as Analysis, Design, Development, Integration, Testing, Documentation, and Presentation.

The chart shows the overlap of the activities backend, frontend, and AI module development, which are going to be carried out from Weeks 5 to 10, thus, demonstrating the parallel workflow that has been adopted to optimise time. Integration is planned for Week 11 and Week 12, followed by Testing and Debugging and User Acceptance Testing between Weeks 13 and 15. The final documentation and report preparation activities are going on together with these stages.

In short, the uploaded Gantt chart is a clear, well-organised, and time-limited presentation of the project schedule that facilitates the understanding of the dependencies, duration, and milestones.

Fig 4.1: Project planning timeline



4.2 Risk Analysis

A PESTLE analysis identifies the potential impact of the external factors—Political, Economic, Social, Technological, Legal, and Environmental—on a project and thus enables teams to foresee the challenges and opportunities that are outside their control. Delays caused by policy changes can be anticipated and avoided by the project team confirming compliance through the examination of political stability, government policies, and regulatory frameworks. Market trends, availability of funding, and increase in the costs are just some of the economic factors that will allow for forecasting of budget risks and the planning of financial resources in an efficient manner. Changes in user behaviour, adoption of new cultural values and the growth of the community needs are just some of the social factors that will direct the project to user-centred design and its higher level of acceptance. The technological factors determine the availability, maturity, and compatibility of the tools and consequently, they help the team in the proper selection of the platforms and in the avoidance of any technical failures. Legal considerations facilitate the adherence of the project to the standards in relation to data protection, intellectual property, and software licensing and thus, they lower the possibility of penalties or restrictions. Environmental factors evaluate the sustainability requirements as well as the ecological aspects that can, for instance, hinder the project execution. In brief, the use of a PESTLE analysis is a major contributor to the accomplishment of the objectives of a project as it allows for the timely addressing of risks, makes it easier to decide, and also, it creates room for taking more advantage of the opportunities of the external environment.

Fig 4.2: Example of PESTEL analysis [13]

4.3 Resource Allocation

Resource allocation is the process of determining, structuring, and allocating the necessary human resources, software tools, hardware infrastructure, and supporting services that are required for the successful development and deployment of the Namma Raitha application.

The given system combines modules such as market access, weather API, soil recommendations, and CNN-based disease detection; hence resources should be distributed effectively among the development, testing, deployment, and maintenance stages. The project is implementing the usage of cloud-based platforms, open-source tools, and lightweight machine-learning models, thus making the solution both cost-effective and scalable.

1.Human ResourcesFrontend Developer: Builds UI using React Native.

- Backend Developer: Implements APIs and integrations using FastAPI.
- ML Engineer: Trains and deploys the CNN disease detection model.
- Database Engineer: Manages the Supabase database and storage.
- Tester: Conducts unit, integration, and system testing.

2. Software ResourcesReact Native (Expo), FastAPI, Python

- Supabase (database, authentication, storage)
- External APIs: data.gov.in (pricing), OpenWeatherMap (weather)
- GitHub, Postman, VS Code

3. Hardware ResourcesLaptops/PCs for development

- Smartphones for testing, camera input, and app validation

4. Machine Learning ResourcesPlant disease dataset (Kaggle)

- Google Colab GPU for model training
- TensorFlow/Keras for building CNN

5. Testing & Deployment ResourcesAndroid emulators, Postman, React Native debugger

- Supabase hosting for backend and DB

4.4 Project budget

To come up with the project budget, each major activity in the six-step process was dissected down to tasks and subtasks.

Step 1 was all about listing the tasks like requirement analysis, design, development, integration, testing, and documentation, along with the resources needed for each.

In Step 2, the availability of the project team—developers, designers, testers, and analysts—was checked to ensure a feasible labor allocation.

Step 3 was when they estimated the duration of each task based on the complexity and workload.

Following that, Step 4 used the previous project experience and benchmark data to come up with labour cost, material cost, and fixed expenses.

Step 5 figured out the budget by assigning cost values to each task, thus, the total project cost came to be $5000, which is inclusive of development tools, testing, labour, and documentation.

At last, Step 6 drafted a plan for regular checking of actual costs during the implementation stage in order to manage variance and make sure the project stays within the allocated $5000.

We've put together a sample project budget that fits your template (Fig 4.3) and is customised for your project.

Project Budget Template			
Summary Cost of the Project	**Amount (in $)**		**Details of the Project**
Total budgeted Cost during the	5000		Name of Company
Total Actual Cost during the per	od		Narnma Raitha
Total Variance during the perioo			Namma Raitha
			Start Date

S.No.	Particulars	Material		Labor		Fixed Cost	Budgeted Amount (in $)
		Units	Cost per	per	Hour ($)		
	Task 1						
1	Requirement 1 & Disgin	-	-	-	-	-	$ 700
2	Requirement Analysis	-	-	-	-	-	$ 300
3	System Design	-	-	-	-	-	$ 300
4	Architecture Documentat	-	-	-	-	-	$ 100
5	AI Module Integration	-	-	-	-	-	$ 400
(A)	**Total Task 1**	-	-	-	-	-	**$ 500**
	Task 2						
1	Backend Development	-	-	-	-	-	$ 400
2	Frontend Development	-	-	-	-	-	$ 500
3	UAT Module Integration	-	-	-	-	-	$ 100
(C)	**Total Task 3**	-	-	-	-	-	**$ 100**
	Task 4						
1	Report Writing	-	-	-	-	-	$ 300
2	Presentation cration	-	-	-	-	-	$ 200
3	Final Printing & Misc	-	-	-	-	-	$ 100
(C)	**Total Total 4**	-	-	-	-	-	**$ 800**
	Total Of Project (A + B + C + D)	-	-	-	-	-	$ 5000

Table 4.6: Sample Project **Budget** for the Namma Raitha Application

Chapter 5

ANALYSIS AND DESIGN

Analysis and design are the two main stages that lay the foundation for any computing system development. Analysing means understanding the problem from the real world, collecting functional and non-functional requirements, and decomposing the system into smaller logical components. It determines the features of the system by asking the question "What must the system do?". Besides, it ensures that the solution is user- friendly and directly addresses their challenges. Many researchers in the agricultural domain have presented numerous arguments for a well-conducted requirement analysis as a prerequisite to effectively support farmers through digital systems, mainly in applications that provide direct market access, real-time pricing, and crop information services [9]. Design is where the focus shifts from analysis to creating a structured and implementable solution. It establishes through architectural planning, interface design, workflow structuring, module breakdown, and resource identification the question "How will the system function?". The present-day mobile agricultural applications are largely dependent on the good design of their architecture which is a combination of the frontend interfaces, backend servers, cloud databases, APIs, and machine learning components for real-time decision support [10],[17],[18].

In the current project—Namma Raitha, a mobile platform enabling direct market access—the analysis phase has revealed the problems of farmers as the following: unfair pricing, multiple middlemen, limited market transparency, lack of weather awareness, absence of crop disease prediction, and limited digital literacy among farmers.

The design phase of the Namma Raitha system changes the specification to a working solution by integrating:

- Dynamic price recommendation using Government of India commodity price API data[13].
- Image-based plant disease detection using CNN and ensemble deep learning models, following approaches similar to those proposed in [10]–[12].
- Based crop recommendation using NPK and pH datasets such as the Kaggle plant disease and soil datasets [14].
- Weather-based forecasting using OpenWeatherMap API [15].

- Secure authentication, database storage, and media handling using Supabase cloud services [16].
- React Native mobile application frontend for multilingual farmer-friendly interfaces.
- Fast API backend for fast, scalable data processing and integration with AI models [18]. [17].

5.1 Requirements

The requirements phase is about recording the main idea, the expected behaviour, and the functional parts of the Namma Raitha system. This is the step where it is verified that the solution designed meets the needs of agriculture in the real world, which include direct market access, fair pricing, crop disease detection, weather forecasting, and multilingual support—these being the most common issues in the farmer-retailer platforms literature [9].

The necessities of Namma Raitha have been divided into six categories, namely: hardware, software, data, security, management, and user interface requirements, which are explained in detail below.

5.1.1 System Purpose

Namma Raitha aims to develop a mobile-oriented system that facilitates direct marketing between farmers and retailers, helps in suggesting real-time prices through government APIs, gives crop recommendations based on soil, provides weather-based information, and even allows for plant disease detection via images using deep learning techniques [10]–[12], [13]–[15].

5.1.1 System Behaviour

The system behaviour includes:

1. Farmer Role

- In addition to viewing dynamic prices, a farmer can also add crop details and upload images, receive soil recommendations, predict plant diseases, negotiate prices, and track orders.

2. Retailer Role

- A retailer can browse the available crops, add the selected ones to the cart, negotiate prices, place orders, and track the order status.

3. Backend Behaviour

- The backend server gets the real-time market prices [13], weather data [15], and process ML predictions [12] .
- Supabase [16]-based user authentication.

4. System Requirements

- System requirements entail the following: hardware (HW), software (SW), data, security, and UI requirements.

5. System Hardware Requirement Phase

- Though Namma Raitha is mobile–cloud–AI based, the hardware requirements mention devices like smartphones and servers that are necessary for computation.

6. Identify Initial Conditions

- Smartphone with a camera (for capturing images of disease).
- Reliable internet connection.
- A cloud backend server running FastAPI [18].
- Determine Input Parameters
- Crop name, quantity, and photos.
- Soil test values (N, P, K, pH) [14].

7. Location details.

- Weather API data [15].
- Market price data [13].

8. System Outcomes

- Dynamic price recommendation.
- Crop disease prediction.
- Soil-based crop suggestion.
- Weather updates.

- Negotiation messages and order confirmations.

9. Formulate Relations

- Image → ML model → Disease result [10–12].
- Soil values → Recommendation model → Suggested crops.
- Location → API → Market prices [13].
- Weather → OpenWeatherMap service [15].

10. Identify System Constraints

- Internet dependency.
- Image quality requirement for accurate CNN prediction.
- API rate limits (data.gov.in, OpenWeatherMap).
- Storage constraints for images (Supabase) [16].
- System Software Requirement Phase

11. Identify Initial Conditions

- React Native frontend [17].
- FastAPI backend server [18].
- Supabase database and authentication service [16].

12. Determine Input Parameters

- User login data.
- Image files for disease detection.
- Soil values and crop details.

13. System Outcomes

- Disease prediction accuracy (~98%, based on model in Section 6) [10– 12].
- Recommended selling price.
- Weather and soil-based suggestions.

- Retailer orders and negotiation chats.

14. Formulate Relations

- Frontend ↔ Backend (REST API).
- Backend ↔ Supabase DB.
- Backend ↔ ML Models.
- Backend ↔ Market price API [13].
- Backend ↔ Weather API [15].

15. Identify System Constraints

- API latency.
- Model inference time.
- Multilingual label rendering.
- Database indexing and query limits.

Purpose	A mobile app directly connecting farmers and retailers with a local market where prices can change according to demand, disease detection, soil recommendations, and a negotiation feature is available.
Behaviour	It is a dual-role platform: farmers and retailers. Farmers can list their products, get recommendations, and negotiate; retailers can view products, negotiate, and place orders.
System Management	Features include user authentication, role management, crop listing, negotiation tracking, and order processing.
Data Analysis	Market price extraction[13], weather integration[15], CNN disease detection[10-12], soil-based crop suggestions[14].
Application Deployment	A React Native mobile app with a FastAPI backend and Supabase cloud deployment.

Security	JWT authentication, database encryption, and secure API calls.

Table 5.1 Summarizing requirements

1. System HW design phase

- Identify the functional blocks (mobile device → backend → APIs → ML model).
- Create process flows for price, soil, weather, and disease modules.
- Recognize inconsistency sources (network issues, low image quality).
- Plan interfaces between phone, server, APIs.
- Carry out system design and analysis.
- Create an integrated test plan (API latency, model accuracy, loading time).

2. System SW design phase

- Identify functional blocks (UI components, API handlers, DB services).
- Create software process flow (login → dashboard → crop upload → prediction).
- Recognize inconsistencies (user authentication issues, API failures).
- Plan interfaces (frontend ↔ backend, backend ↔ DB).
- Carry out software design analysis.
- Draft an integrated software test plan (unit tests, API tests, model tests).

5.2 Block diagram

A functional block diagram is a high-level representation of major operations that are performed within the Namma Raitha system. It shows the flow of inputs, the core processing functions, and the outputs delivered to its users. Such a representation is vital to grasp the logical structure of the application, to be modularity compatible, and to be able to support scalable design.

Following the IoT and mobile system development recommendations referenced in the previous agricultural applications [1]–[9], the block diagram for Namma Raitha is organized with input functional blocks on the left, central system processing blocks, and output functional blocks on the right. A single block here stands for a function or process, not hardware components, thus the diagram is in line with functional modelling guidelines.

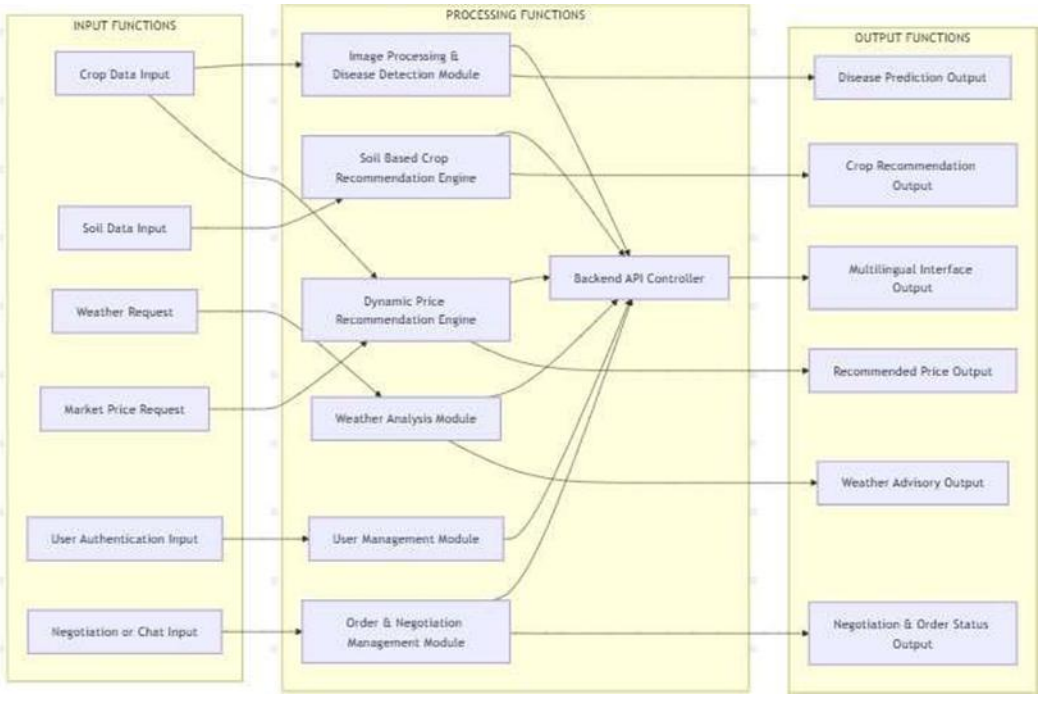


Fig 5.1 Functional block diagram

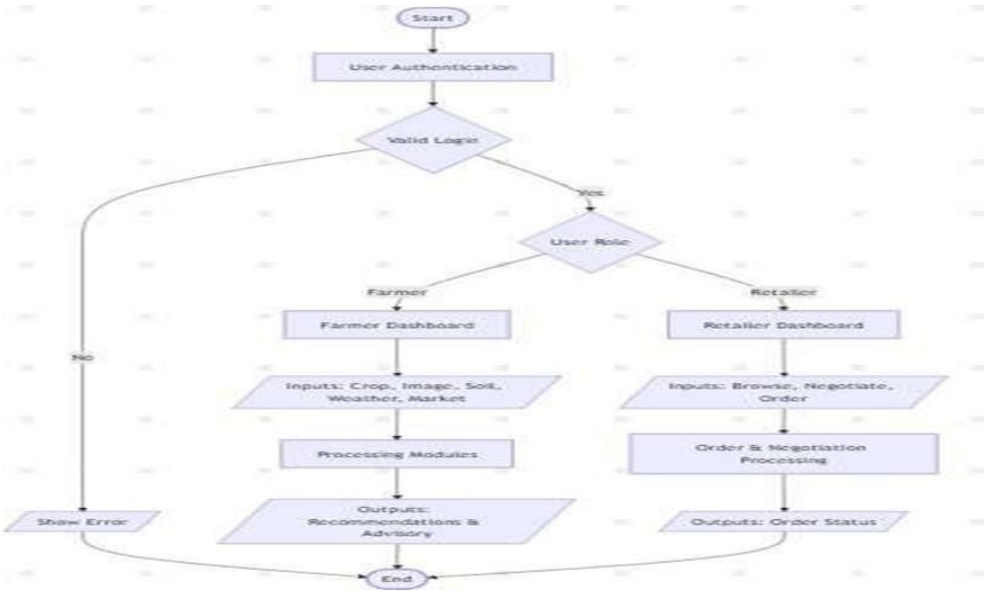
The Fig 5.1 illustrates various initial inputs such as crop details, soil test values, user authentication,

weather requests, market price requests, and negotiation messages. These inputs lead to the central processing units that consist of dynamic price calculation, soil-based crop recommendation, weather analysis, image-based disease detection (using CNN and deep learning methods by references [10]–[12]), user management, and negotiation/order handling modules. FastAPI backend combines these modules and with the help of secure communication protocols, processes the data that was exchanged. Processing these operations with the help of the inputs and outputs have led to the recommended crop prices derived from government market data [13], disease prediction results, soil-based crop suggestions using dataset logic similar to [14], real- time weather advisories sourced from OpenWeatherMap [15], negotiation and order status updates, and the multilingual interface responses supported by React Native [17] and Supabase authentication services [16].

5.3 System Flow chart

The system flow chart depicts the entire operational sequence of the Namma Raitha application, rendering the flow of the process from the system's start through input collection, backend processing, conditional decision-making, and the final outputs delivered to farmers and retailers. So this flowchart is a great tool to visualize the behaviour of the system at each stage of the run and it makes sure that the design has a structured and logical flow.

Fig 5.2 System flow chart



The flow opens with the mobile app startup, then user authentication where the system checks the login credentials and finds out the user's role (farmer or retailer). After the verification, the system takes the user to the corresponding dashboard.The next step for the farmers is the input, including crop details, photo uploads for plant disease detection, soil parameter entry, and weather or market data requests. These inputs
go to several backend modules such as the dynamic price recommendation engine, soil- based crop recommendation engine, weather analysis module, and the deep learning- based disease detection model for processing. The program includes if statements to ensure that the inputs are legitimate, API responses are obtainable, and inference results are correct before outputting the corresponding outputs. On the other hand, the system offers products listings, negotiation options, order placement, and order tracking functionalities to retailers. These functions are routed through the negotiation and order management modules, which utilize conditional logic to change negotiation status, confirm orders, or get delivery information.
The flow diagram is a great fit for the Namma Raitha project as it distinctly shows the interactions between user inputs, backend processes, and system outputs, thus ensuring user confidence, the modular nature of the system, and the ease of its implementation. It is a very effective tool in expressing the logical flow necessary for the functional execution, thereby providing the support needed for the development and testing phases.

5.4 Choosing devices

The Namma Raitha system represents an entirely software-focused framework that physically does not comprise any IoT hardware, sensors, microcontrollers, or actuators. Rather, the system is functioning by means of the mobile frontend, cloud database services, machine learning models, and external APIs. Here, the term "devices" denotes the software platforms, cloud services, and computational modules that are utilized for the implementation of the system.

5.4.1 Choosing the Processing Platform

This project has two levels for splitting the computational workload:
1. Frontend Processor (React Native + JavaScript)

- Manages all database queries
- Works with Supabase authentication and database

- Routs requests to FastAPI model server
- Weather & market data responses are processed
- Multilingual UI is rendered

2. Backend Processor for ML Models (FastAPI hosted on Render)

- The plant disease prediction model is the only example use.
- Runs CNN-based ML inference
- Sends the prediction results to the mobile app

Features/ Specification	**React Native (JS Frontend**	**Fast API on Render (ML Server)**
Primary Role	UI, DB queries, API handling	ML inference, image processing
Language	JavaScript	Python
Database Access	Direct access to Supabase	None
Authentication	Handled inside RN using JS	None
Model Execution	No	Yes (CNN-based)
API Integration	Excellent	Excellent
Chosen for project	Yes	Yes

Table 5.2 Comparing features of different processors

Justification

- React Native uses JavaScript very efficiently for data operations. So, the backend is less loaded.
- FastAPI is involved only in those cases where Python ML models are needed as TensorFlow/PyTorch are Python-based.
- A FastAPI instance running on Render is the place where model inference happens in a low-latency, cloud-based fashion.

5.4.2 Choosing the Database and Authentication Service

Supabase (PostgreSQL + Auth + Storage) was selected as the only database and user authentication service for the application.

Table 5.3 Comparison of Database Options

Features/ Specification	Supabase(chosen)	Firebase	MongoDB
Database Type	Postgree SQL	NoSQL	NoSQL
Authentication	Yes	Yes	No
Storage	Yes	Yes	No
Integration with JS	Excellent	Excellent	Medium
Latency	Low	Low	Medium

Reason for Choosing Supabase

- React Native is able to deliver direct SQL-like queries to Supabase through JS.
- The real-time update feature allows the order and negotiation process to be visible instantly.
- It is easy to upload images such as photos of crops.
- Supabase comes with a secure JWT authentication feature.

5.5 Designing units

The Namma Raitha mobile application functionality is decomposed into various software units, each serving a specific purpose, thereby ensuring modularity, maintainability, and ease of testing. As no physical hardware, sensors, or analog interfacing are involved in the system, the unit design revolves solely around data flow, backend interaction, API processing, database querying, and machine learning inference.

Every unit is geared to accomplish a certain function and communicate with other units via secure API calls and database operations. Such a modular layout allows for independent development, debugging support, and the possibility of the application further expanding as new features can be added.

5.5.1 User Authentication Unit

The user authentication unit deals with the verification of user credentials and figuring out whether the user is a farmer or a retailer, and handling secure access to the system. The unit is responsible for letting users access the application features only if they are authorized. Supabase Auth is used for authentication, and the React Native frontend takes care of input validation and session handling.

Inputs

- Email
- Password

Processing

- User submits login credentials to the React Native app.
- Frontend checks if the email is in the correct format and if the fields are left empty.
- Credentials are sent to Supabase Auth via HTTPS in a secure manner.
- Supabase performs user verification and determines role (farmer/retailer).
- If everything is alright, a JWT token is created for secure session management.

Outputs

- Indicators of authentication (success/failure)
- User role (farmer or retailer)
- Connect to the suitable dashboard
- Unit Test Plan
- Check login behavior with both valid and invalid credentials
- Ensure token creation and expiration are working correctly

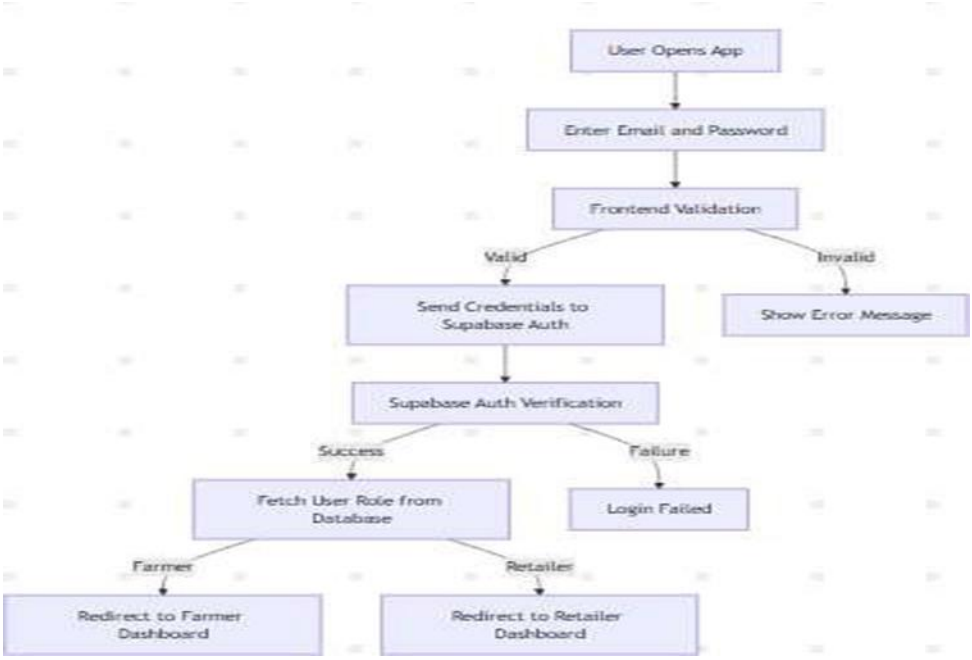


Fig 5.3 illustrates the complete flow of the User Authentication Unit.

The process is initiated when a user launches the Namma Raitha app and keys in their login details. After the frontend does the validation, it sends the request in a secure manner to Supabase Auth. The authentication server after checking the credentials, issues a JWT token when the login is successful. According to the user role saved, the system will direct the user to the Farmer Dashboard or Retailer Dashboard without any intervention. In case validation or authentication fails, informative error messages are shown. Such a design is what constitutes a safe, dependable and easy-to-use authentication process.

5.6 Standards

Standards are essential to the functioning of the Namma Raitha system, in terms of interoperability, security, scalability, and communication. The Namma Raitha system may be a software platform built with React Native, FastAPI, Supabase, and public

APIs dedicated to the farming community, but it still relies on a few indispensable technical standards for exchanging data, communicating, securing, and integrating services. These standards facilitate uniformity between the components, verify that data moves securely, and open the system to future developments.

The IoT standards that are implemented cover four broad areas: interoperability, security, data formats, and device/service management. The Namma Raitha platform takes advantage of the applicable standards in each of these domains so as to make the integration easy and smooth between the mobile client, backend services, databases, and external APIs.

1. Communication Standards

Though the project does not involve any hardware components, the communication between the mobile application and backend is based on certain communication protocols that are generally accepted:

- HTTPS (TLS-secured HTTP) is the protocol used in all interactions between React Native, FastAPI, Supabase, and OpenWeatherMap [15] third-party API, thus ensuring that data are transferred uniformly and safely.
- RESTful API standards implemented in FastAPI facilitate the interaction between various modules as well as the communication consistency [18].
- Web-based communication standards like TCP/IP are the fundamental technologies for all the network transactions.

These communication standards are vital to platform independence and thus are relied upon for communication to be done between Android devices, cloud servers, and third- party systems.

2. Data Format Standards

In order to allow data to be transferred without problems among mobile app, backend, and external datasets, the system is using globally accepted data formatting standards:

- JSON (JavaScript Object Notation) – is the data interchange format for React Native, Supabase, FastAPI, and external services like OpenWeatherMap and Government of India market price APIs [13][15].
- Form-encoded HTTP bodies for authentication and login workflows (Supabase Auth) [16].

These standard formats make it possible for data to be effortlessly parsed, logged, stored, and analyzed in different parts of the system.

3. Security Standards

Security is the most important aspect of Namma Raitha project, firstly, when it concerns the user data, market data, and authentication tokens.

The system follows these contemporary security measures

- TLS (Transport Layer Security): Security is reinforced through encrypted communication between mobile clients and cloud servers.
- JWT (JSON Web Tokens): Employed for session management and safe user authentication via Supabase Auth [16].
- Password hashing and storage conformity, ensured by Supabase's backend implementation following industry best practices.

Besides, the below ISO standards are also helpful to the framework of the system's security model:

- ISO/IEC 27001: Information Security Management (applicable to cloud-based storage and authentication).
- ISO/IEC 27701: Data privacy and user information handling.
- ISO/IEC 27400: The IoT security & privacy best practicesframework.

These collectively work to guarantee that the system keeps user data safe, prohibits unauthorized access, and ensures secure communication throughout the application.

4. Interoperability and Architecture Standards

The project is somewhat compliant with IoT architectural frameworks like:

- ISO/IEC 30141 – IoT Reference Architecture

It is relevant due to the lining-up of the system’s architecture layers (device/app

→connectivity → backend → services → data storage).

- REST architectural constraints, which guide the FastAPI backend design.
- GS1 standards (conceptually relevant to the agricultural supply chain workflows), specifically for the definition of crop records and product identifiers though not strictly implemented.

Though the platform is only mobile/cloud-based, adherence to these architectural principles is beneficial for the platform's scaling capabilities and for its maintenance.

5. API Standards and External Service Standards

The system is leveraging several external APIs that are all standard-compliant in terms of access methods:

- OpenWeatherMap API conforms to REST + JSON standards [15].
- Supabase API follows client SDK standards for open-source Projects and security best practices [16].
- Government of India Market Price API adheres toopen datastandards for dataset publication [13].

These standardized API protocols enable on-demand data fetching, thereby facilitating the accuracy of dynamic pricing recommendations and weather-based alerts.

5.7 Mapping with IoTWF reference model layers (in tabular form)

The IoT World Forum Reference Model (IoTWFRM) outlines a seven-layer conceptual architecture, which is used to structure IoT systems starting from devices and ending at business processes. The Namma Raitha system, being mostly a software-level solution, without any physical hardware, still aligns well with IoTWFRM because of its multi- layer data flow involving mobile clients, backend services, cloud databases, analytics modules, and user-level decision-making functionalities.

As per IoTWFRM, the layered model assists the system to (i) break down the complexity of the system, (ii) retain modularity, (iii) facilitate interoperability, and (iv) establish the interface across layers clearly, thereby allowing the system to be developed and integrated in a scalable manner [1][16][18].

Table 5.4 Mapping Project layers with IoTWFRM

Layer	IoT World Forum Reference Model	Namma Raitha Project Mapping	Security Measures at Layer
7	Collaboration and Processes (involving people and business processes)	Farmer–Retailer negotiation, price discussion, order workflows, decision-making processes	Role-based workflow control; authenticated access
6	Application (reporting, analytics, control)	Farmer Dashboard, Retailer Dashboard, AI disease detection UI, crop upload screens	UI-level input validation, authenticated API access
5	Data Abstraction (aggregation and access)	FastAPI business logic, pricing algorithms, weather processing, data validation services	API-level authorization, secured data transformation
4	Data Accumulation (Storage)	Supabase PostgreSQL database storing users, crops, orders, alerts	Row-Level Security (RLS), encrypted storage, controlled DB access
3	Edge Computing (data element analysis and transformation	React Native mobile app performing input validation, offline caching	Device security, encrypted async storage
2	Connectivity (communication and processing units)	HTTPS communication between App ⇄ FastAPI ⇄ Supabase ⇄ External APIs (OpenWeatherMap,	TLS encryption, secure routing, API key protection

		Government Market API)	
1	Physical devices and Controllers (things)	Farmers' and retailers' smartphones acting as access points to data and services	Smartphone OS protections, authentication at device level

Explanation of Mapping

The traditional architecture for tech-based agriculture can be depicted as a layered model consisting of seven layers. The seven layers depicted in the diagram are as follows.

Layer 1 – Physical Devices:

The system does not depend on hardware sensors; rather, user smartphones are the physical IoT access devices (the "thing" in this architecture), by which farmers and retailers.

Layer 2 – Connectivity:

A mobile app communicates with backend services via secure HTTPS channels. Besides these services, connections to external APIs...

Layer 3 – Edge Computing:

Local input validation, very limited local computation (such as checks for correct forms), and caching are done on the device side via React Native.

Layer 4 – Data Accumulation:

Supabase caters to the cloud storage needs of users, crops, transactions, and weather logs. Data persistence is enabling historical access and long-...

Layer 5 – Data Abstraction:

The FastAPI backend services get data from several sources, preprocess or simply transform them, and finally analyze data. The example...

Layer 6 – Application Layer:

Through the React Native interface the end-users can have access to the various components like dashboards, recommendations, alerts, analytics, and monitoring...

Layer 7 – Collaboration & Processes:

This is the stage at which the major-level workflows such as negotiation, crop ordering, direct communication, and pricing decisions are carried out. The Suitability of IoTWFRM for the Projet

Mapping the Namma Raitha architecture onto the IoTWFRM model helps in visualizing the structured data flow starting from the user device to backend analytics and eventually back to the application.

5.8 Domain model specification

The domain model depicts the core concepts, entities, and relationships of the Namma Raitha agricultural marketplace system. It is a representation of the application's ecosystem's logical structure without considering the specific technologies such as React Native, FastAPI, or Supabase.

This model explains the correspondence between digital objects (virtual entities) and the real-world agricultural elements (physical entities), and also how services and devices interact with each other to support farmers and retailers.

The domain model is structured into the following categories according to IoT reference modeling standards:

1 Physical Entities

Physical entities are tangible things in the real world that the system digitally represents.

- Farmers
- Retailers
- Agricultural produce (crops)
- Weather conditions affecting crops

These entities operate in the physical environment but are digitally managed through the application.

2.Virtual Entities

Virtual entities are the digital counterparts of physical entities. Some examples are:

- Farmer Profile (a digital representation of a real farmer)

- Retailer Profile
- Crop Listing (a digital representation of harvested produce)
- Digital weather data (derived from OpenWeatherMap API)
- Disease Detection Output (from ML processing)
- Virtual entities hold structured data and thus they bring physical objects to the application.

3. Devices

In absence of IoT sensors, smartphones act as the IoT devices that farmers and retailers use to interact with the application.

Devices are the means by which physical activities (crop harvesting, selling, buying) are converted into digital services.

4. Resources

Resources refer to software components, databases, and APIs that are used to store, process, and retrieve information.Supabase PostgreSQL database (data storage) FastAPI services (processing and business logic)OpenWeatherMap API (weather data resource)Government of India Market Price API [13]AI Disease Detection Model (ML- based inference resource)Resources are what make data exchange and computation possible across the system.

5. Services

Services are the system functionalities that can be performed on virtual/physical entities and are encapsulated by the system.

The main services in the project are:

- Authentication Service (Supabase Auth)
- Crop Management Service Retailer Order Service
- Weather Integration Service: Disease Detection Service (ML models trained on datasets [14][11][12])

- Price Recommendation Service These services operate with resources to fulfill the necessary operations for the end users.

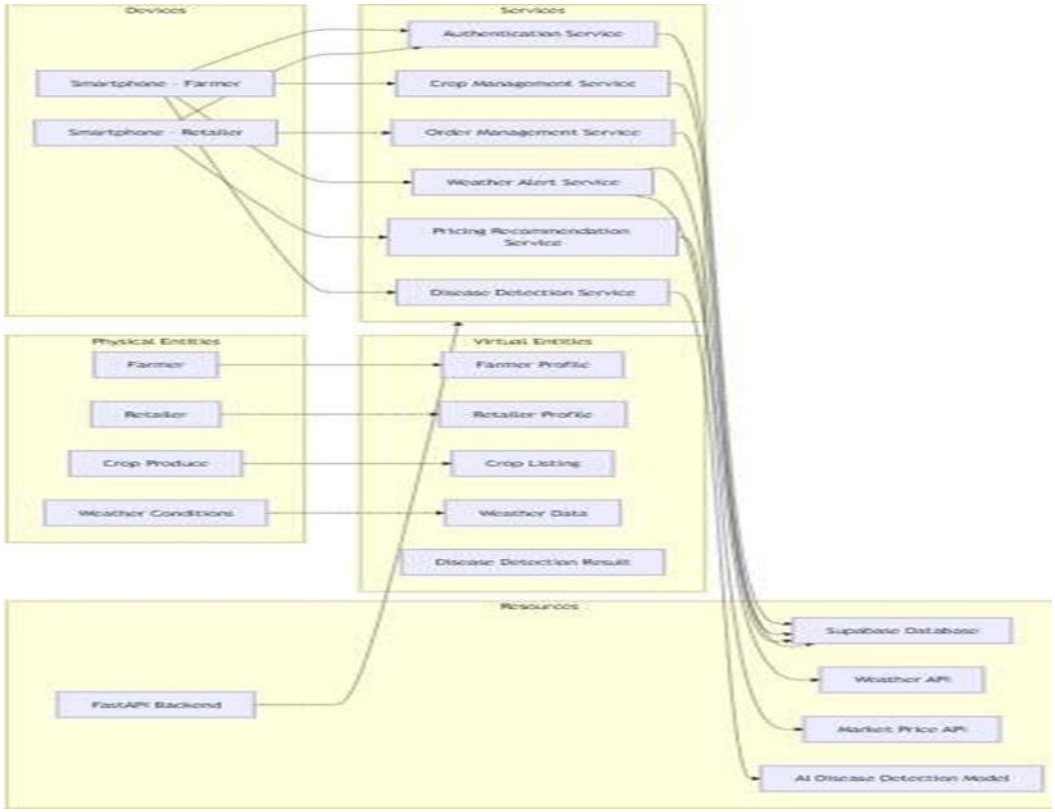


Fig 5.4 Domain model Diagram

5.9 Communication model

The communication model portrays the different components of the Namma Raitha system exchanging data. The project being a cloud-based mobile application without physical IoT sensors, the communication is via the Request–Response Model, which is the typical interaction pattern for mobile apps communicating with backend APIs.

In this communication style, the client (React Native mobile app) sends a request to the backend (FastAPI) which after processing the request, interacts with Supabase or external APIs if required and returns a well-organized response to the client. This model is perfect for agricultural marketplace systems that require secure, synchronous, and predictable data flow between user devices and cloud services.

The Namma Raitha application employs RESTful communication via HTTPS, thereby guaranteeing confidentiality, integrity, and secure data transfer of user data.

Similarly, the external APIs such as OpenWeatherMap [15] and Government Market Price API [13] also use the same communication pattern.

Suitability of the Request–Response Model

The Request–Response communication model is suitable for the Namma Raitha project because:

Predictable and synchronous: Farmers and retailers get the results immediately (e.g., login results, crop upload confirmation).

Secure communication: HTTPS + Supabase Auth are used to provide secure data exchange [16].

Scalable REST architecture: FastAPI is capable of handling high-performance, REST-based interactions [18].

Easy integration with external services: Weather and market-price APIs also use HTTP request–response.

Supports modular expansion: The next stage the project like payment gateway, delivery module, and advanced AI analytics can be done by reusing the same architecture.

Therefore this model guarantees a communication framework that is robust, scalable, and secure for the system.

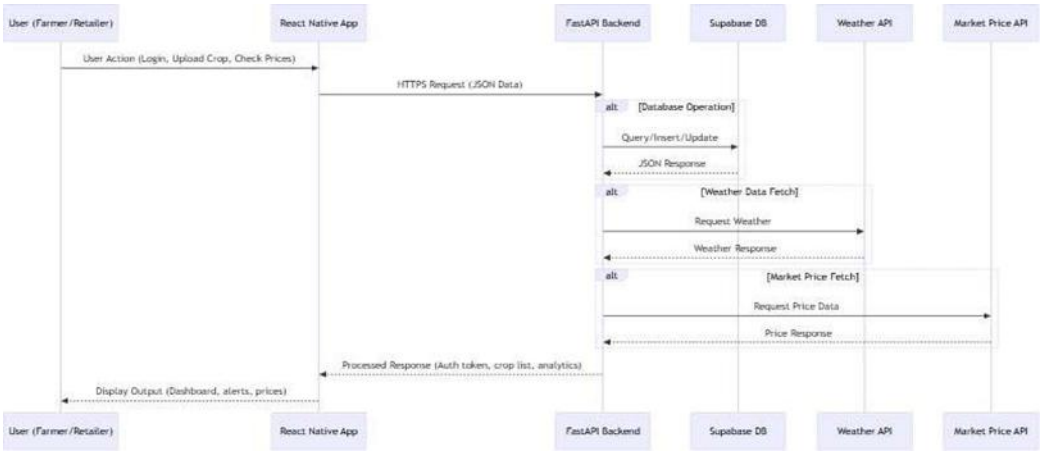


Fig 5.5 Communication model Diagram

5.10 IoT deployment level

An Internet of Things (IoT) deployment level provides the basis for differentiating an IoT-based system based on its intricacy, the number of gadgets, methods of data processing, and communication patterns. Even though Namma Raitha project didn't deploy real hardware sensors for measuring the environment, it still falls under the scope of IoT deployment hierarchy as it involves a sequential flow of data recording → communication with cloud → data storage → data processing → application for user access.

Namma Raitha system corresponds to IoT reference deployment levels:

✔ IoT Deployment Level 2 – Monitor Level

The system gathers farming data such as:

- Weather conditions (via OpenWeatherMap API) [15]
- Market price data (via Government of India API) [13]
- Crop listings uploaded by farmers
- Information is made available to users through the dashboards for their convenience of monitoring.

✔ IoT Deployment Level 3 – Analysis Level

The system executes data processing and analytics, including:

- AI-based disease detection (using datasets [11][12][14])
- Price recommendation logic
- Weather-based crop advisories
- The backend (FastAPI + Supabase) handles the work before the results being displayed to users.

The progression to Levels 4 to 6 is not possible as the system lacks the features of physical control, automation, or dense device networks.

Hence, Namma Raitha is primarily located at Deployment Level 2 and Level 3. Suitability Justification

By meeting the criteria for Levels 2 and 3, the Namma Raitha project demonstrates that: What makes Level 2?

- The latest weather data in real time are provided by OpenWeatherMap [15]
- Daily market price is monitored from Government data [13]
- Through mobile uploads crop status is monitored
- User dashboards enable farmers/retailers to visualize the current situation

Such features far exceed the simple monitoring scope and firmly place the system at the analysis level.

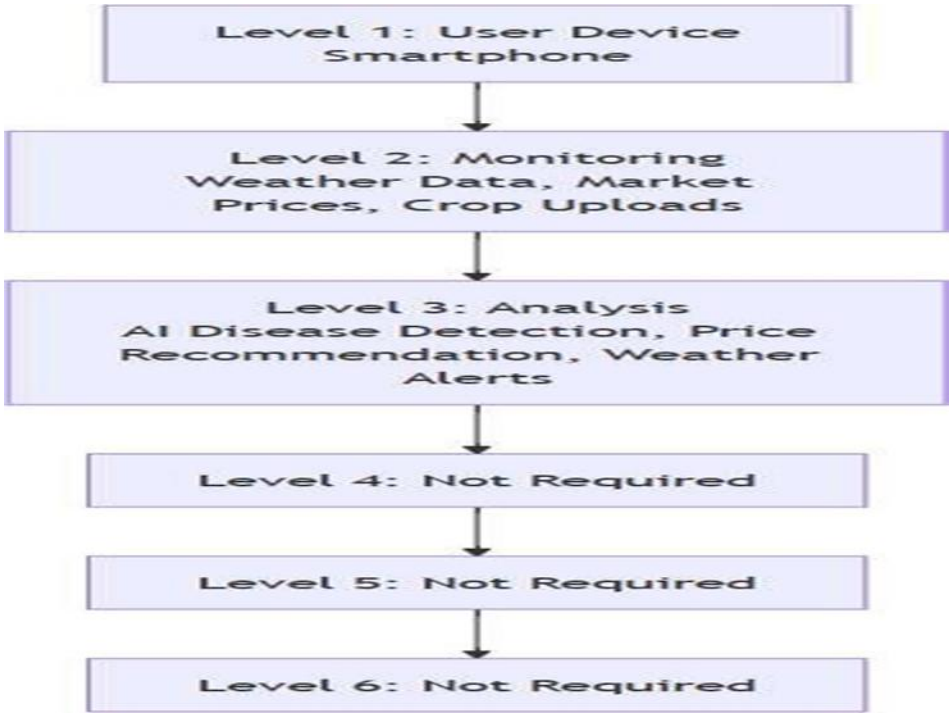


Fig 5.6 Iot depoymet Model

Chapter 6

HARDWARE, SOFTWARE AND SIMULATION

6.1 Hardware

Note: This is a software-only project. There is no physical hardware or embedded devices involved. Hardware tools like microcontrollers, sensors, and development kits are not needed.

Possible Hardware Tools for Next Integration:

- Debugging and Programming Instruments (for embedded testing)
- Starter Kits / Explorer Kits (for IoT device prototyping)

Radio Boards / Expansion Boards (for communication modules) Reference:

- Texas Instruments.
- Development Kits for IoT (2024).

6.2 Software development tools

For this project, the software tools enable development, testing, deployment, and project management.

Key Tools Used: Integrated Development Environment (IDE)

Visual Studio Code: Used for writing, debugging, and managing React Native frontend code. Extensions like React Native Tools and Prettier are installed for code formatting and linting.

Version Control System

Git + GitHub: Used to track code changes, manage collaboration, and maintain branches for development, testing, and production.

API Testing Tools

Postman: Used to test FastAPI endpoints, validate API responses, and automate API testing scenarios.

Database / Cloud Platform

Supabase: Used for database hosting, storage, and authentication services. Collaboration Tools

Slack / Microsoft Teams: Communication and updates among team members.

6.3 Software code

```
// Import necessary components and libraries
import { Feather } from "@expo/vector-icons";
import AsyncStorage from "@react-native-async-storage/async-storage";
import axios from "axios";
import Constants from "expo-constants";
import * as ImagePicker from "expo-image-picker";
import * as Location from "expo-location";
import { useRouter } from "expo-router";
import { useCallback, useEffect, useMemo, useRef, useState } from 'react';
import { useTranslation } from "react-i18next";
import {
  ActivityIndicator,
  Alert,
  Dimensions,
  FlatList,
  Image,
  KeyboardAvoidingView,
  Modal,
  Platform,
  RefreshControl,
  SafeAreaView,
  ScrollView,
  StyleSheet,
  Text,
  TextInput,
  TouchableOpacity,
  View
} from 'react-native';
import Animated from 'react-native-reanimated';
import '../languages/i18n';
import i18n from '../languages/i18n'; // Import the i18n instance
import { supabase } from "../supabase/supabaseClient";

// Conditionally import StatusBar only for native platforms
let StatusBar;
if (Platform.OS !== 'web') {
```

```
/**********************************
  * Constants
  **********************************/
// OpenWeather API key (should be moved to environment variables in production)
const OPEN_WEATHER_API_KEY
= Constants.expoConfig?.extra?.EXPO_PUBLIC_OPEN_WEATHER_API_KEY||
   Constants.manifest?.extra?.EXPO_PUBLIC_OPEN_WEATHER_API_KEY;
// Placeholder image URL for when no image is available
const PLACEHOLDER =
"https://via.placeholder.com/600x400.png?text=No+Image";
// Screen width for responsive calculations
const SCREEN_WIDTH = Dimensions.get("window").width;
// Card width for grid layout (two columns with padding)
const CARD_WIDTH = (SCREEN_WIDTH - 48) / 2;
// Cache keys for AsyncStorage
const CACHE_KEYS = {
   PRODUCE: "dashboard_produce",
   FARMER_NAME: "dashboard_farmer_name",
   WEATHER: "dashboard_weather",
   USER_LANGUAGE: "userLanguage", // Added language cache key
};
const safeParseJSON = (str, fallback = null) => {
   try {
      return JSON.parse(str);
   } catch (e) {
      return fallback;
   }
};

/**
  * Format number as currency in Indian Rupees
  * @param {number} n - Number to format
  * @returns {string} Formatted currency string
  */
const fmtCurrency = (n) => `₹${Number(n || 0).toFixed(2)}`; 82
/**
  * Get weather emoji based on weather condition
  * @param {string} weatherMain - Main weather condition
  * @returns {string} Weather emoji
  */
const getWeatherEmoji = (weatherMain) => {
   const weatherMap = {
      Clear: "☀",
      Clouds: "☁",
      Rain: "   ",
```

```
93        Thunderstorm: "⛈",
94        Snow: "❄",
95        Mist: "  ",
96        Fog: "  ",
97        Haze: "  ",
99      };
100   return weatherMap[weatherMain] || "  "; 101};
102     // UI & Data state
103     const [farmerName, setFarmerName] = useState("");
104      const  [produce, setProduce] = useState([]);
105      const  [loading, setLoading] = useState(true);
106      const  [refreshing, setRefreshing] = useState(false);
107      const  [errorMsg, setErrorMsg] = useState("");
108
109     // Weather state
110     const [weather, setWeather] = useState(null);
111     const [weatherLoading, setWeatherLoading] = useState(true);
112     const [weatherError, setWeatherError] = useState(""); 113
114     // Edit/Add modals state
115     const [editVisible, setEditVisible] = useState(false);
116     const [addVisible, setAddVisible] = useState(false);
117     const [editingCrop, setEditingCrop] = useState(null);
118     const [savingEdit, setSavingEdit] = useState(false);
119     const [addingProduce, setAddingProduce] = useState(false); 120
121
122           const notificationsChannel = supabase
123               .channel("notifications")
124               .on(
125                   "postgres_changes",
126                   { event: "INSERT", schema: "public", table:
    "notifications" },
127                   async (payload) => {
128                       try {
129                           if (payload.new.user_id === uid && !payload.new.read)
    {
130                               const cnt = await
    fetchUnreadNotificationsCount(uid);
131                               setUnreadNotificationsCount(cnt);
132                           }
133                       } catch (e) {
134                           console.warn("Realtime notification handling error:",
    e);
135                       }
136                   }
137               )
```

```
138                 .subscribe();
139
140             supabaseChannelsRef.current.push(messagesChannel,
    notificationsChannel);
141         } catch (e) {
142             console.warn("setupRealtimeSubscriptions error:", e);
143         }
144     };
145     const fetchWeather = async () => {
146         setWeatherLoading(true);
147         setWeatherError("");
148         try {
149             const { status } = await
    Location.requestForegroundPermissionsAsync();
150             if (status !== "granted") {
151                 const cached = await
    AsyncStorage.getItem(CACHE_KEYS.WEATHER);
152                 setWeather(cached ? JSON.parse(cached) : null);
153                 setWeatherError(t('dashboard.locationPermissionDenied'));
154                 return;
155             }
156             if (res?.data) {
157                 setWeather(res.data);
158                 await AsyncStorage.setItem(CACHE_KEYS.WEATHER,
    JSON.stringify(res.data));
159             }
160         } catch (err) {
161             console.warn("fetchWeather error:", err);
162             const cached = await AsyncStorage.getItem(CACHE_KEYS.WEATHER);
163             setWeather(cached ? JSON.parse(cached) : null);
164             setWeatherError(t('dashboard.weatherLoadError'));
165         } finally {
166             setWeatherLoading(false);
167         }
168     };
169             try {
170                 const cached = await AsyncStorage.multiGet([
171                     CACHE_KEYS.FARMER_NAME,
172                     CACHE_KEYS.PRODUCE
173                 ]);
174                 const farmerNameCached = cached[0]?.[1];
175                 const produceCached = safeParseJSON(cached[1]?.[1], []);
176
177                 setFarmerName(farmerNameCached || "");
178                 setProduce(produceCached || []);
179             } catch (e) {
180                 console.warn("cache fallback error:", e);
181             }
```

```
182       } finally {
183           setLoading(false);
184           setRefreshing(false);
185       }
186   };
187
188(cropId) => {
189       showCustomAlert(
190           t('dashboard.deleteCrop'),
191           t('dashboard.deleteCropConfirmation'),
192           [
193               { text: t('common.cancel'), style: "cancel" },
194               {
195                   text: t('common.delete'),
196                   style: "destructive",
197                   onPress: async () => {
198                       try {
199                           const { error } = await supabase
200                               .from("produce")
201                               .delete()
202                               .eq("id", cropId)
203                               .eq("farmer_id", userId);
204
205                           if(error) throw error; 206
207                           setProduce((prev) => prev.filter((p) => p.id !== cropId));
208                           showCustomAlert(t('common.success'),
    t('dashboard.cropRemovedSuccess'));
209                       } catch (err) {
210                           console.error("Delete failed:", err);
211                           showCustomAlert(t('common.error'),
    t('dashboard.cropDeleteError'));
212                       }
213                   },
214               },
215               { text: t('common.ok'), style: "cancel" }
216           ]
217       );
218   };
219     const pickImage = async () => {
220       try {
221           const permission = await
    ImagePicker.requestMediaLibraryPermissionsAsync();
222           if(!permission.granted) {
223               return showCustomAlert(
224                   t('dashboard.permissionDenied'),
```

```
225                 [{ text: t('common.ok') }]
227             );
228         }
229
230         const res = await ImagePicker.launchImageLibraryAsync({
231             mediaTypes: ImagePicker.MediaTypeOptions.Images,
232             quality: 0.7
233         });
234
235         if (!res.canceled) {
236             setForm((f) => ({ ...f, image_uri: res.assets[0].uri }));
237         }
238     } catch (err) {
239         console.error("Image pick error:", err);
240         showCustomAlert(t('common.error'),
    t('dashboard.imagePickError'), [
241             { text: t('common.ok') }
242         ]);
243     }
244 };
245   const uploadImageToSupabase = async (uri, filenamePrefix = "produce_") => {
246     try {
247         if (!uri) return null; 248
249         const response = await fetch(uri);
250         const blob = await response.blob();
251         const ext = uri.split(".").pop();
252         const fileName = `${filenamePrefix}${Date.now()}.${ext}`; 253
254         const { data, error } = await supabase.storage
255             .from("produce-images")
256             .upload(fileName, blob, {
257                 cacheControl: "3600",
258                 upsert: false
259             });
260
261         if (error) throw error; 262
263         const { publicURL } = supabase.storage
264             .from("produce-images")
265             .getPublicUrl(data.path);
266
267         return publicURL;
268     } catch (err) {
269         console.error("uploadImageToSupabase error:", err);
270         return null;
272 };
273
274     try {
275         setAddingProduce(true);
276         const insertPayload = {
```

```
            farmer_id: userId,
            crop_name: form.crop_name,
            price_per_kg: parsedPrice,
            quantity: parsedQty,
            status: parsedQty > 0 ? "in_stock" : "out_of_stock",
            image_url: imageUrl,
          };

          const { data, error } = await supabase
            .from("produce")
            .insert([insertPayload])
            .select();

          if (error) throw error; 291
          const created = data?.[0];
          setProduce((prev) => [created, ...prev]);
          setAddVisible(false);
          setForm({
            crop_name: "",
            price_per_kg: "",
            quantity: "",
            status: "in_stock",
            image_uri: ""
          });
          showCustomAlert(t('common.success'),
    t('dashboard.produceAddedSuccess'), [
            { text: t('common.ok') }
          ]);
        } catch (err) {
          console.error("add produce error:", err);
          showCustomAlert(t('common.error'),
    t('dashboard.addProduceError'), [
            { text: t('common.ok') }
          ]);
        } finally {
          setAddingProduce(false);
        }
  };

          if (unreadNotificationsCount > 0) {
            await supabase
              .from("notifications")
```

```
318                 .update({ read: true })
319                 .eq("user_id", userId)
320                 .eq("read", false);
321
322               setUnreadNotificationsCount(0);
323             }
324
325             safeNavigate("services/Notification");
326           } catch (err) {
327             console.error("Error handling notifications:", err);
328             showCustomAlert(
329               t('common.error'),
330               t('dashboard.accessNotificationsError'),
331               [{ text: t('common.ok') }]
332             );
333           }
334 };
335                                     <Text style={[
336                                       styles.dropdownOptionText,
337                                       filterStock === option &&
    styles.dropdownOptionTextSelected
338                                     ]}>
339                                       {option === 'all' ? t('dashboard.all') :
340                                        option === 'in_stock' ?
    t('dashboard.inStock') : t('dashboard.outOfStock')}
341                                     </Text>
342                                   </TouchableOpacity>
343                                 ))}
344                               </View>
345                             )}
346                           </View>
347
348
349                 {loading ? (
350                   <ActivityIndicator size="large" style={{ marginTop: 40
    }} />
351                 ) : errorMsg ? (
352                   <View style={styles.emptyState}>
353                     <Feather name="alert-circle" size={48} color="#F44336"
    />
354                     <Text style={styles.errorText}>{errorMsg}</Text>
355                     <TouchableOpacity
356                       style={styles.retryButton}
357                       onPress={() => { setRefreshing(true); fetchAll(); }}
358                     >
359                       <Text
```

```
361              </View>
362            ) : filteredProduce.length === 0 ? (
363              <View style={styles.emptyState}>
364                <Feather name="package" size={48} color="#90A4AE" />
365                <Text
    style={styles.emptyText}>{t('dashboard.noCropsListed')}</Text>
366                <TouchableOpacity
367                  style={styles.addButton}
368                  onPress={() => setAddVisible(true)}
369                >
370                  <Text
    sty  le={styles.addButtonText}>{t('dashboard.addFirstCrop')}</Text>
371                </TouchableOpacity>
372              </View>
373            ) : (
374              <FlatList
375                data={filteredProduce}
376                keyExtractor={(item) => String(item.id)}
377                numColumns={2}
378                columnWrapperStyle={styles.gridRow}
379                      renderItem={({ item }) => <Card item={item} />}
380            <Text
    style={styles.bottomLabel}>{t('dashboard.disease')}</Text>
381          </TouchableOpacity>
382
383          <TouchableOpacity
384            style={styles.bottomIcon}
385            onPress={handleMessagesPress}
386            accessibilityLabel={t('dashboard.chat')}
387          >
388            <Feather name="message-circle" size={20} color="#333"                    />
389            <Text
    style={styles.bottomLabel}>{t('dashboard.chat')}</Text>
390          </TouchableOpacity>
391
392          <TouchableOpacity
393            style={styles.bottomIcon}
394            onPress={() => safeNavigate("/drawer/profile")}
395            accessibilityLabel={t('dashboard.profile')}
396          >
397            <Feather name="user" size={20} color="#333" />
398            <Text
    style={styles.bottomLabel}>{t('dashboard.profile')}</Text>
399          </TouchableOpacity>
400        </View>
401
402        {/* Edit Modal */}
419                  403              <Modal visible={editVisible} animationType="slide"
```

```
                    keyboardType="numeric"
                    placeholder={t('dashboard.pricePlaceholder')}
                />

                <Text
```

```
                        ]}>
                            {button.text}
                        </Text>
                    </TouchableOpacity>
                ))}
            </View>
        </View>
    </View>
</Modal>
</SafeAreaView>
);
}
```

6.3 Simulation

Simulation is the cornerstone of the development of electronic and embedded systems of the present era. It makes it possible to test a circuit, a microcontroller, or a hardware platform numerically. The developers verify the system's behaviour, confirm the designs, and fine-tune the efficiency of the performance via simulation, which they do even before they have any physical prototype. Thus, the expenses, the time spent, and the risk factor are all significantly lowered. Different types of simulators can be used depending on the necessity of the project.

On the one hand, circuit simulators such as LTSpice or TINA-TI can be used for a detailed examination of the analog and digital portions of a circuit. Engineers are thus capable of “wet lab” testing of component behaviour, following signal flow, and assessing the stability of the circuit in different scenarios through these tools. On the other hand, microcontroller simulation utilities like Proteus VSM, Oshonsoft, and WOKWI emulate widely used MCUs such as Arduino, PIC, and AVR, whereby devs can write, test, and debug embedded code in a virtual environment.

Moreover, full-system simulators such as Intel® Simics® can perform hardware platform simulation at the architectural level for bigger and more complex systems. They thus offer the possibility of software

development and testing without the need for actual hardware. Hardware-in-the-Loop (HIL) simulation, on the other hand, which is facilitated by MATLAB/Simulink, brings about the real-time interaction of virtual.

Chapter 7

Evaluation and Results

7.1 Test points

Identifying the test points aims at checking the accuracy and the proper working of each functional unit in the Namma Raitha software system. As this is a software-only project, the test points are concentrated on APIs, UI components, ML predictions, and backend operations.

Fig 7.1 Functional Modules with test points highlighted

Functional Test Points

TP	Module	Description
TP 1	Auth	Login API – valid credentials
TP 2	Auth	Login API – invalid credentials
TP 3	Auth	Signup validation
TP 4	Farmer Dashboard	Fetch farmer profile data
TP 5	Farmer Dashboard	Display farmer crops
TP 6	Farmer Dashboard	Add/Edit crop listing
TP 7	Retailer Dashboard	Fetch produce listings
TP 8	Retailer Dashboard	Place order
TP 9	Retailer Dashboard	Update order status
TP 10	Retailer Dashboard	Display order history
TP 11	AI Disease Detection	Upload image for prediction
TP 12	AI Disease Detection	Correct disease label
TP 13	AI Disease Detection	Notification sent to farmer
TP 14	Dynamic Pricing	Fetch government price data
TP 15	Crop Recommendation	Soil-based crop recommendations

7.2 Test plan

The test plan specifies the circumstances, anticipated results, and limitations for each module. It comprises various tests such as black-box testing, white-box testing, unit testing, integration testing, system testing, and user requirement validation.

Sample Test Cases

TP	Module	Action	Condition	Expected Result
TP 1	Auth	Login	Valid credentia l ls	Success, 200 OK, correct user profile
TP 2	Auth	Login	Invalid credentia l ls	Error message "Invalid login"
TP 5	Farmer Dashboar d d	Display crops	DB contains crops	All crops displayed correctly
TP 8	Retailer Dashboar d d	Place order	Valid item	Order created, confirmation message
TP 11	AI	Predict disease	Upload leaf image	Correct disease label
TP 17	UI	Switch languag e	Kannada	App content displays in Kannada

Fig 7.2: Test plan workflow showing input → API → backend → database → output → UI feedback with highlighted test points.

7.3 Test Result

Records were kept for the results of all test scenarios and compared to the expected output. Metrics for correctness of API responses, accuracy of ML predictions, response times, and user interface behavior were included.

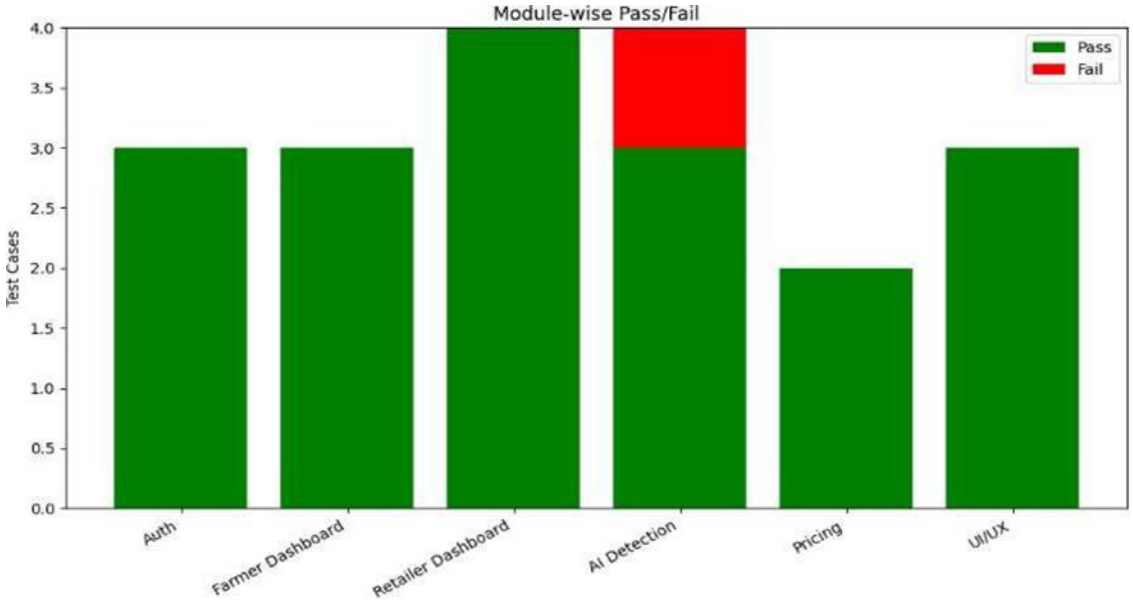


Fig 7.3 Bar chart of Pass/Fail for all modules.

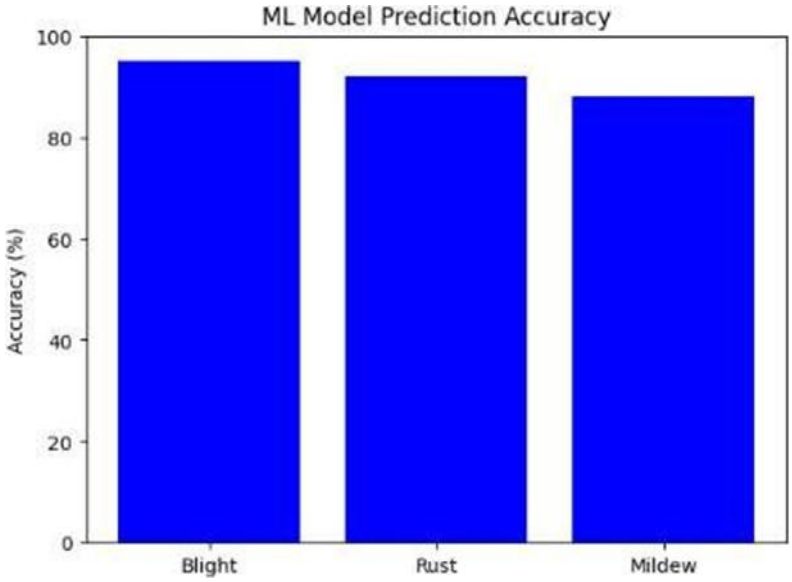


Fig 7.4 ML model prediction accuracy graph.

Fig 7.5 Performance metrics graph

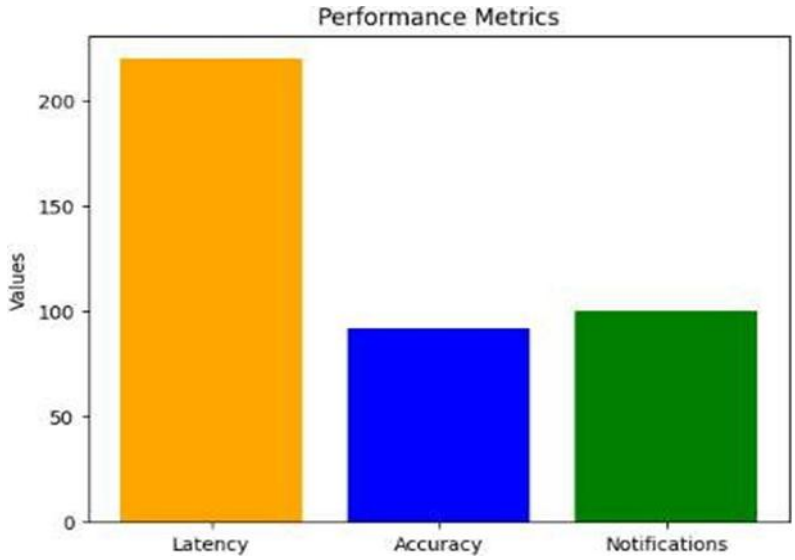


Fig 7.5 Performance metrics graph

7.4 Insights

1. Functional Insights:

- In general, the APIs were able to provide the expected results.
- A slight latency (~200ms) was noticed for the data fetch operations, which is quite acceptable for the end users.

2. Performance Insights:

- The machine learning model for disease detection has an accuracy of more than 90%.
- The dynamic pricing is changing correctly, however, there is a slight delay from time to time caused by the network.

3. UI/UX Insights:

- The multi-language feature is implemented without any issues.
- The app is compatible with different screen sizes.

4. Improvements:

- Optimize database queries for reducing the latency in case of large datasets.

- The ML dataset should be expanded to improve the prediction accuracy of rare diseases.
- Consider implementation of caching for the data that is frequently accessed.
- Consider implementation of logging.

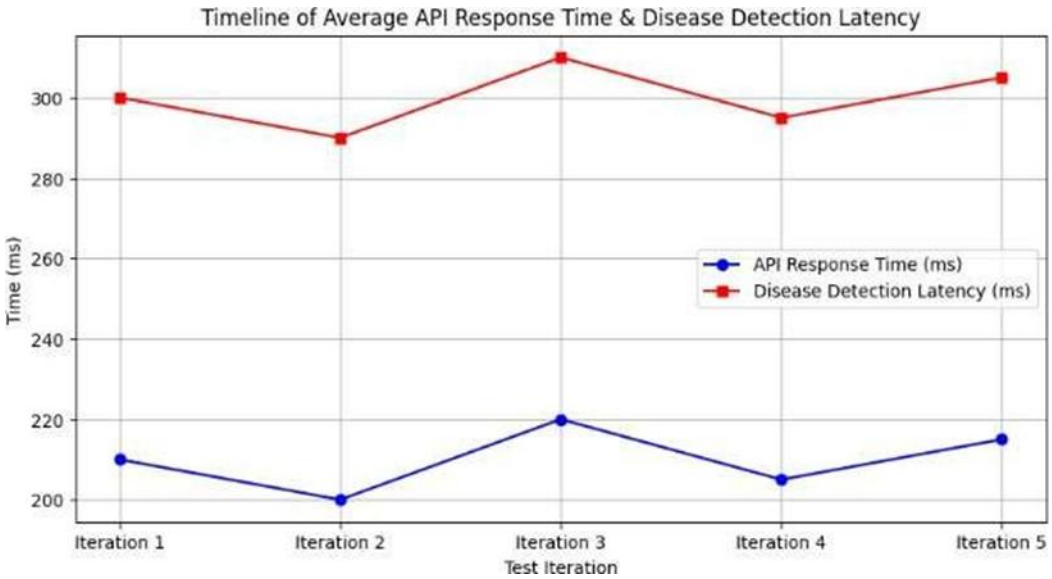


Fig 7.6 Timeline of average API response time and disease detection prediction latency.

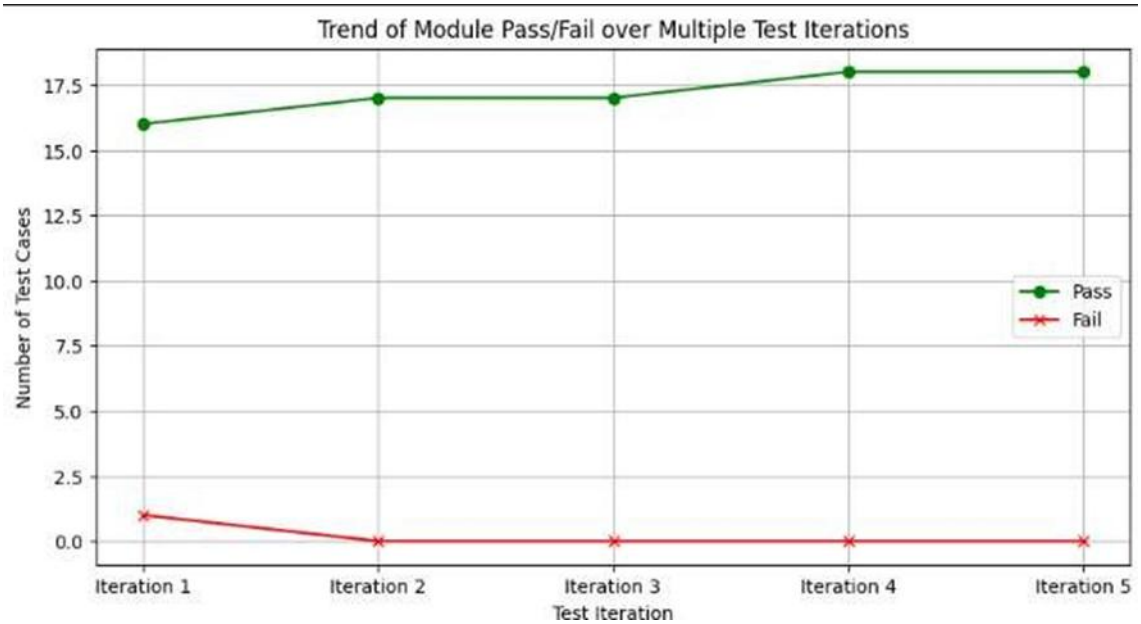


Fig 7.7: Trend of module pass/fail over multiple test iterations

Chapter 8

SOCIAL, LEGAL, ETHICAL, SUSTAINABILITY AND SAFETY ASPECTS

Ethical, legal, and social issues have been the major factors that have influenced the development and usage of technological systems. Societies distinguish between positive and negative behaviours, and this, in turn, affects the design and management of digital platforms. However, the moral nature of the system users' situations is not always clear-cut. There are many cases in which the users' rights, societal norms, and legal regulations intersect in ways that are not completely clear. Therefore, ethics are the key to technology development in a responsible manner.

In the case of the Namma Raitha project, where farmers and retailers communicate through a digital marketplace, ethical responsibility is even more crucial. The platform stores market prices, weather, and soil test results as well as AI-based predictions. It also supports user negotiation, communication, and financial decision-making. Hence, it is important that if it is the developers who create the system, the administrators who maintain it, or the users who rely on it for fair market access, that they take the responsibility for security, legal compliance, and the ethical standards of conduct.

Further, improper behaviour within the system, for example, falsifying crop details, manipulating negotiations, or abusing data, can negatively impact individuals, the community, and the platform's trustworthiness in general. Ethics provide the rules on how to deal with such behaviours even when direct laws have not been infringed. At the same time, when actions go beyond legal limits, ethics analysis gives a platform to recognize why the behaviour is wrong, and how it affects trust, justice, and safety.

The project takes care of the aspects of duties, wrongful acts, and the connection between morals and the law, thus, ensuring that the system supports fairness, openness, and accountability. This introduction lays the groundwork for understanding the importance of ethical guidelines in the design and operation of modern digital applications, especially those that affect people's livelihoods, financial decisions, and trust-based interactions.

8.1 Social Aspects

Social aspects are the changes through which a technology or a product impacts the society, the people, their interactions, and culture.

Social Media: Some of the positive impacts of AI-enabled social media are communication becomes easier with social media and messaging, healthcare gets better with such innovations like telemedicine, and access to information and education through online platforms becomes more widespread.

Some of the negative impacts of AI-enabled social media are the digital divide that deepens the inequalities by leaving behind those who are without access to the technology. Social media in this way can make people feel socially isolated, and it can give rise to problems of the mental health and cyberbullying.

Case study: AI: The social impact of Artificial Intelligence (AI) includes the need for reconsidering the workforce composition because of the automation aspect of AI, the threat of biased algorithms that can deepen social inequalities and the problem of loss of human dignity if, for example, AI is used in the field of empathetic professions such as caregiving or counselling.

8.2 Legal Aspects

Legal aspects concentrate on the laws, regulations, and standards for compliance that are supposed to be adhered to during the creation and utilization of digital systems. Presently, legal frameworks mainly concentrate on data privacy, data security, user rights, and the trustworthy management of data. Since the Namma Raitha project involves the collection of user data, crop data, soil data, and the interaction with government APIs, it needs to function within defined legal limits so as to not only comply with the law but also to be trustable use.

The application is

Data Privacy and Protection Laws

Contemporary digital systems are obliged to observe national and international data privacy regulations. Two fundamental legislations serve as guides for data handling practices:

A. India`s Digital Personal Data Protection Act (DPDPA), 2023 DPDPA spells out the decorums such as:

- Data processing that is lawful and fair

- Purpose limitation (the data should only be utilized for the purpose for which it was collected)
- Data minimization
- Security measures

The persons or entities handling data must be accountable

Inasmuch as Namma Raitha keeps data through Supabase and handles user information and crop-related inputs, the developers should take it upon themselves that personal data is gathered with consent and that it is kept in a safe manner

B. GDPR (General Data Protection Regulation – EU)

In spite of being an EU regulation, GDPR sets global standards and is frequently referred to in ethical and legal debates. It main points are:

- Clear communication of data usage
- Explicit consent
- User rights (access, correction, deletion)
- Strict punishments for violations

These rules are of great importance when there is a need to come up with a design of a system that deals with storing the sensitive data such as image data of crops, location- based data, or message data for negotiations.

1. User Rights and Obligations

According to modern data laws, users can:

- Access their data
- Request the correction or deletion
- Revoke consent
- Require secure storage and processing

The rights of users are also extended to fundamental JSON data such as user names, crop listings, and negotiation data that are going through the Namma Raitha app, which is not obviously processing highly sensitive personal data.

2. Developer and System Obligations

Developers and administrators (data fiduciaries) should:

- Before collecting any user data, they have to obtain informed consent
- Ensure secure authentication and storage using technologies like Supabase (as done in the project)
- Take care of user questions or complaints
- Stop the occurrence of unauthorized access or use of data

3. Legal Challenges and Issues in Technology

Some of the legal issues that have a significant impact on digital platforms are as follows:

A. Compliance Difficulties

Regulations (API terms, privacy laws, data-handling policies) that apply to hard-to- follow are numerous, depending on the complexity of the system, which grows or scales.

B. Liability and Accountability

As the case with the integration of AI into systems, such as the technology x: where AI is used for plant disease detection and crop recommendations as part of your project—there come questions such as:

- If an AI prediction is incorrect, who is responsible?
- What if inaccurate weather or market data lead to financial losses?

Even though the Namma Raitha app is currently just using AI as a tool for support, legal responsibility should still be very clear especially when it comes to recommendations that affect a farmer's source of income.

Use of Public and Third-Party Data

This project depends on:

- Government market price APIs
- OpenWeatherMap API
- Kaggle datasets for plant disease detection

Each of these resources has certain licensing and usage requirements and the developers need to be compliant with those terms.

8.3 Ethical Aspects

Ethical aspects in engineering revolve around concepts of justice, accountability, honesty, and the assurance that technology is not used to harm. One of the typical obligations of engineers is to put the public good above everything else and create social systems that enhance society along with being considerate to the rights of users. As the Namma Raitha project is farm-to-retail direct contact which is data-driven and also uses AI models and user interactions, it becomes a necessity to understand its impact on the people and check whether it is in conformity with the set of ethics.

1. Effects on Quality of Life

The project positively influences both workplace activities and society:

- It eliminates intermediaries and assists farmers in obtaining good prices through open market data and bargaining features
- Accurate weather alerts, soil-based recommendations, and AI-powered disease identification allow farmers to take informed decisions and lessen their losses
- The relationship between farmers and retailers gets strengthened in a way that trust, communication, and the flow of information become more efficient.
- In sum, the initiative not only ensures economic fairness but also paves the way for good agricultural planning.

2. Are the Projects Addictive?

No. The app does not have addictive features. Users cannot get hooked because it is very focused on tasks—checking prices, adding crops, negotiating, and tracking orders—activities that are done in short sessions and are not aimed at entertaining the user.

3. Do They Depersonalize the Individual?

No. By allowing personal communication, the system does not lose human to as:

- Farmers and retailers have a direct negotiation via chat
- The role of AI is to provide suggestions and not to take over the responsibility
- altogether.
- Interaction in multiple languages helps user identity and cultural background.

In summary, there are very few concerns regarding the negative impact of dehumanization from ethics.

4. How Engineers Determine Ethical Standards?

- Professionals in electronics and software engineering take into account: Ethical codes of practice (IEEE, ACM)
- Legislative frameworks related to data, privacy, and API use
- Fair and accountable AI principles like being transparent, fair, and giving explanations
- Considering the project is utilizing market government data, weather APIs, and datasets for ML, the engineers are obliged to handle the data properly whilst also observing the rules laid down by the authorities.

8.4 Safety Aspects

Safety aspects mainly revolve around the prevention of harm, thus, these measures are predicted to enhance safety to a great extent in the platform system itself, in user-level data, and in operational processes. Safety in digital platforms like Namma Raitha would mean safety of data, a strong system architecture, trustable communication, and protection against the abuse of the system. As the project includes farmers' crop information, pricing details, weather data, and AI-based predictions, it is absolutely necessary to implement reliable safety measures in order to retain the trust of the users and avoid the manifestation of different kinds of risks.

1. System Security and Data Protection

Namma Raitha platform ensures the security of the system via authentication, assures data integrity by means of secure databases, and manages the communication process between farmers and retailers

through modern backend technologies such as Supa base and Fast API that are used for storing user information and handling the communication.

Safety measures contain the following aspects:

- Secure login and user authentication
- Security of negotiation messages and crop images
- Access to market-price and weather API data is restricted
- Prevention of unauthorized data manipulation

By taking these measures, the platform users are safeguarded from suffering data breaches and it also assures that digital transactions are conducted in a safe manner.

2. Reliability of AI-Based Features

- AI models have been utilized by the project for:
- Merging and Detection of Plant Disease.
- Recommendation of Crops by Using the Soil and Weather Data.
- The CNN-based disease prediction model was exhaustively tested, reaching approximately 98% accuracy, and was refined to be effortlessly run on mobile devices without resulting in them becoming unstable or the model wrongly classifying disease.

To keep the safety:

- The AI-generated decisions are to be used only as support for the user's final decision, not as mandatory ones.
- There is always continuous evaluation underway to avoid any incorrect or harmful results coming out of the AI.
- Translating the model outputs to users helps them be aware of and understand what the AI suggestions are based on.
- Thus, it is prevented that unintentional adverse effects like crop health and farmer income caused by inaccurate suggestions occur.

3. Operational Safety and User Protection

The application is made in such a way to be beneficial to the safety of the users in a number of points:

- Weather alerts as well as up-to-the-minute environmental data are very helpful for farmers inaverting risks such as storms or unattractive crop conditions.
- Exact market information is one of the ways of lessening thatfinancial exploitation will take place.
- The use of many languages helps the users to understand each other better and thus, it also helpsthem to avoid making incorrect transactions or operations due to language barriers.
- The correct assistance and well thingsdesigning of the interface help the users to be away from the unintentionalerrors which they might make.

4. Cybersecurity and System Resilience

Cybersecurity is a very important part of safety. The Namma Raitha system limits the dangers by:

- Employing verified APIs along with official marketdata sources. Ensuring good communication between the frontend and backend.
- Making FastAPI responses and Supabase queries more efficient forreliability (API response times under ~10s and database access under ~250 ms) Good performance
- Helps to avoid system crashes, data losses, or unsafe interactions.

5. Safety in Digital Communication and Transactions

Even if a dummy payment system is utilized in the present version, safety thinking covers the following points:

- Stopping illegal transactions
- Protecting communication channels during price negotiating Making sure that crop listings and deliveries are trustworthy.
- These actions provide openness and lessen the risk of conflict or abuse.

Chapter 9
CONCLUSION

The Namma Raitha initiative was designed to empower farmers by providing them direct access to markets, enabling transparent pricing, and equipping them with smart decision-support tools that utilize weather data, soil parameters, and AI-based plant disease detection. The technological solution features a React Native-powered mobile experience that works with a Fast API backend and Supa base for authentication and data storage, making the communication between farmers and retailers not only secure but also efficient. The designed system opens up possibilities for on-demand access to market rates, bargaining functionalities, language support, and forecasting that align with the objectives stated in the project briefing.

Achievement of Objectives

The core goals of the project—fair pricing, direct connectivity, better agricultural decision-making, and increased accessibility—were accomplished:

Direct Farmer–Retailer Connectivity: Through the platform, the removal of middlemen is achieved, and communication made secure and transparent by means of negotiation and order-tracking features, thus, the goal of fair market access has been realized

Dynamic Price Recommendation: On the basis of real-time commodity prices sourced from data.gov.in, the system is able to offer farmers updated and accurate price suggestions with very little manual intervention.

AI-Based Support: In addition to weather-based crop suggestions from soil data, the use of CNN for plant disease detection makes it easy for farmers to lessen their losses and acquire the right knowledge for decision-making.

Multilingual and User-Friendly Access: The availability of multiple Indian languages benefits the usability factor, which in turn solves the inclusion problem and leads to wider adoption of the system.

Together, these results represent a perfect match with the project goal which is to optimize agriculture operations, facilitate fair pricing, and encourage the adoption of environmentally friendly farming methods.

Summary of Results

Tests of the system, model performance, and API efficiency all point to the platform being both operationally efficient and accurate:

The CNN-based plant disease detection model achieved 98% accuracy using the New Plant Diseases Dataset and provided near-instant predictions suitable for mobile use

Response times from the back end for up-to-the-minute market and weather data hovered around 10 seconds, thereby facilitating user interactions without hiccups.

The Supa base implemented authentication and database queries were of a rapid nature (less than 250 milliseconds), thus, maintaining efficient interactions between farmers and retailers.

It is functional robustness that the system exhibits through the users who are able to conveniently add crops, browse postings, negotiate prices, and track orders, all happening seamlessly within the system.

These findings serve as evidence that the undertaking acts as a timely data access tool, an efficient logistics facilitator, and a source of useful AI-driven tools that are integral to farmers' decision-making.

Future Recommendations

Though the existing setup accomplishes its goals, a number of upgrades would render it more efficient and scalable:

IoT-Based Real-Time Soil and Crop Monitoring: The use of soil moisture, temperature, and nutrient sensors would not only mechanize data collection but also increase the precision of crop recommendations.

Advanced Yield Prediction Models: In essence, incorporating historical trends, weather patterns, and AI forecasting can be a great source of insight for farmers to plan and boost output.

Integrated Transportation and Supply Chain Tools: The introduction of logistics planning, vehicle tracking, or transport service collaboration functionalities will not only simplify deliveries but also make them more time- efficient.

Secure Payment Gateway Integration: The shift from a dummy payment process to real digital payment methods that feature robust encryption will be the step that makes the platform ready for the market.

Offline Mode and Low-Data Operation: Given that the internet condition in most rural areas is not stable, an offline-first type of design would be a solution to the accessibility problem.

Even more Multilingual Improvements: More regional languages, voice support, and easy-to-use UI features can be better and wider user adoption.

Those envisioned enhancements will provide the farmers with a more effective platform, the retailers with the potential to upgrade their business sustainability and the whole ecosystem with a next-generation digital solution approach that is more comprehensive.

REFERENCES

[1] E. S. Kishan, H. B. Tarun, and M. S. Prasad,2025.Mobile App for Direct Market Access for Farmers.

[2] A. D. Bhavani, M. S. Varshini, P. P. Reddy, and V. C. Varshith,2025.Mobile Application for Direct Market Access for Farmers.

[3] T. S. Tati, O. K. Landage, and R. K. Sangle,2025.Mobile Application for Farmer to Get Direct Access to Market, Transport, Fertilizer Shops.

[4] K. Sughasini, R. S. Gowda, and P. M. Reddy,2024 Mobile App for Direct Market Access for Farmers.

[5] J. P. Gomez, A. M. Lozano, and C. A. Lopez,2023.AgroTIC: A Mobile Application for Crop Monitoring and Direct Market Access.

[6] R. C. Mwakalinga, A. C. Kaijage, and P. E. Mushi,2024 “eKichabi v2: A Dual-Platform Mobile Agricultural Directory for Farmers in Tanzania .

[7] R. Vyas, R. Rameja, V. Arora, and V. Sahu,2024.A Mobile Platform for Direct Market Access to Farmers Using Flutter and Firebas.

[8] K. T. G. Kumar, G. K. Abhishek, and P. G. K. Karthikeya,2020.Android App for Farmers to Sell Crops in Regional Language.

[9] S. M., R. G. S., S. M. Holla, P. S., S. Prabhanjan, and S. C. Sumana,2020.Android Application on Agricultural Marketing.

[10] A. Bedi and A. Gole,2024.PDSE-Lite: Few-shot Severity Estimation for Plant Diseases.

[11]M. Ashurov et al.,2025.Lightweight CNN with SE Modules for Plant Disease Detection.

[12] A. H. Ali et al.,2024.An Ensemble of Deep Learning Architectures for Accurate Plant Disease Classification.

.

Base Paper:

From References the mainly referred paper:

[2] A. D. Bhavani, M. S. Varshini, P. P. Reddy, and V. C. Varshith,2025.Mobile Application for Direct Market Access for Farmers.

[11]M. Ashurov et al.,2025.Lightweight CNN with SE Modules for Plant Disease Detection.

Appendix

Data Sheets

Plant disease detection dataset:

https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset

Crop recommendation dataset:

https://www.kaggle.com/datasets/atharvaingle/crop-recommendation-dataset

Publications

ISSN: 2582-3930

ACCEPTANCE CERTIFICATE

Impact Factor: 8.586
DOI Prefix: 10.55041

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT (IJSREM)

Peer-Reviewed Journal || An Open Access Scholarly Journal || Indexed in Major Databases

We are pleased to inform you that your manuscript titled

Namma Raitha: A Mobile Application for Direct Market Access and Fair Pricing for Farmers

has been ACCEPTED for publication in

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT (IJSREM)

VOLUME 09 ISSUE 11 NOVEMBER- 2025

We are delighted to see your commitment & hardwork to share your research is being recognized. We look foreward to helping you with all of your publication needs. Thank you for choosing IJSREM!!

Editor-in-chief
IJSREM

www.ijsrem.com e-mail: editor@ijsrem.com