



**Faculty of Computers and Information
Kafr El-Sheikh University**



SMART AGRICULTURAL ASSISTANT MOBILE APPLICATION

By

Khaled Kamal Ali

Nahla Mohamed Saeed

Khalid Ghonem AbdLhamid

Samar Basuny Haidar

Gaber Mohamed Lotfi

Esraa AbdElfttah Mohamed

Hussien Talha Hussien

Khaled Waled Mohamed

Supervisor

Assoc. Prof. Reda M. Hussien

Undergraduate Project

Submitted to the textbfFaculty of Computers and Information Kafr El-Sheikh University as a
partial fulfillment for BSc.

**Information Systems Department
June 2025**

Acknowledgments

We would like to extend our heartfelt thanks and deepest gratitude to our supervisor, **Dr. Reda M. Hussien** , for his invaluable guidance, continuous support, and unwavering belief in us. His insightful suggestions, patience, and encouragement were instrumental throughout our journey. Even during moments of frustration and uncertainty, he helped us regain focus and move forward with renewed determination. His sincere commitment to monitoring our progress inspired us to learn and grow, both academically and personally.

We also would like to recognize and appreciate ourselves as a team for the dedication, perseverance, and collaborative spirit that we brought to this project. Every challenge we faced was met with resilience and teamwork, and we are proud of what we have achieved together.

Finally, we express our sincere gratitude to our families and friends for their constant support, understanding, and motivation throughout this journey. A special thanks also goes to our wonderful college, Faculty of Computers and Information – Kafr El-Sheikh University, for providing us with the knowledge, resources, and environment that made this project possible.

Abstract

In a world where technology is rapidly transforming every aspect of life, the agricultural sector—so vital to food security and economic stability—must not be left behind. With the rise of AI-powered solutions and smart mobile applications, there is a growing opportunity to support farmers, agricultural engineers, and anyone interested in agriculture through accessible and intelligent tools. Smart Land is an innovative mobile application designed to provide practical, AI-driven agricultural support. At its core is a smart agricultural chatbot, trained on a wide dataset of agricultural questions and expert answers. It covers a broad range of topics including soil types, fertilizers, irrigation systems, crop diseases, planting methods, vegetables, fruits, and more—offering real-time responses and guidance to users of all backgrounds. A key feature of the app is its Soil Fertility Analyzer: users can input soil analysis data—such as nitrogen, phosphorus, potassium, and pH levels—and the model will assess the fertility level (high, medium, or low), helping users make informed decisions about soil treatment and crop selection. A knowledge base of frequently asked questions and expert answers. Dedicated sections with detailed information on different crops, plants, fertilizers, and pesticides, Practical tips on agricultural techniques and sustainable farming methods With a user-friendly interface and a focus on both simplicity and depth, Smart Land empowers users to enhance their agricultural productivity and knowledge—anytime, anywhere. Our vision is clear and focused: To make expert agricultural knowledge accessible to everyone, right from their mobile phones.

Keywords: chat bot, Soil Fertility Analyzer, knowledge base

Contents

Acknowledgments	ii
Abstract	iii
Contents	iii
List of Figures	vii
List of Tables	x
Nomenclature	xii
1 Introduction	1
1.1 Motivation	1
1.2 Project Objectives	1
1.3 Introduction to project	2
1.3.1 Scope	2
1.3.2 Model	3
1.3.3 Functional Components of the Project	3
1.3.4 Technologies	3
1.4 Organizing Profile	4
1.4.1 Range of Experts	4
1.4.2 Feasibility Analysis	4
1.5 Contributions of This Study	6
2 System Requirements and Analysis	7
2.1 Introduction	7
2.2 Analysis model	7
2.2.1 Requirements	8
2.3 Study of the System	8
2.3.1 GUI's (Graphical User Interfaces)	8
2.3.2 Number of Modules	8
2.3.3 Entities Involved in the Project	9
2.3.4 System Requirements	9
2.4 Software Requirements Specification	10
2.4.1 Introduction to SRS	10

2.4.2	Functional Requirements	11
2.4.3	Non-Functional Requirements	12
3	System Design	15
3.1	Introduction	15
3.2	Design	16
3.3	UML	17
3.3.1	Behavioral UML Diagram	18
3.3.2	Structural UML Diagram	21
3.3.3	DFD-data flow diagram	22
3.3.4	ERD Diagram	23
4	Software and Development Tools	25
4.1	Introduction	25
4.2	System Development Tools	25
4.2.1	Figma	26
4.2.2	Mobile Application (Flutter)	26
4.2.3	Backend Technologies	27
4.2.4	Agricultural Assistant Chat Bot	28
4.2.5	Soil Fertility Classification Using Machine Learning	31
5	System Implementation	39
5.1	Initial Screens	39
5.2	Gest	40
5.3	Sign Up And Log In	42
5.4	Forget Password	43
5.5	Home	44
5.6	Categories	46
5.7	Soil Analyzer screens	49
5.8	Chat screen	50
5.9	Filter screens	51
5.10	Notifications screen	52
5.11	Profile screens	53
6	System Testing	57
6.1	Introduction	57
6.2	Testing Process Goals	57
6.2.1	Validation Testing	57
6.2.2	Defect Testing	58
6.3	Strategic Approach to Software Testing	58
6.3.1	Unit Testing	58
6.3.2	Integration Testing	58
6.3.3	System Testing	58
6.3.4	Acceptance Testing	59
6.4	Test Cases	59

6.4.1	Register Test Cases	59
6.4.2	Register Test Cases	60
6.4.3	Login Test Cases	61
6.4.4	User Test Cases	62
6.4.5	user profile Test cases	63
7	Conclusion and Future Work	65
7.1	Conclusion	65
7.2	Proposed Enhancements and Future Modifications	65
7.3	Recommendations	66

List of Figures

3.1	Use Case	18
3.2	user activity diagram	19
3.3	User Registration	20
3.4	User Functionality	21
3.5	Class diagram	21
3.6	Context diagram	22
3.7	Level 1	22
3.8	Database	23
4.1	training loss screen shot	30
4.2	Gradio space web UI	31
4.3	Testing API on Postman	31
4.4	confusion matrix	36
4.5	Model download	37
4.6	Testing API in postman	37
5.1	Initial Screen Logo	39
5.2	Initial Screen 1	39
5.3	Initial Screen 2	40
5.4	Initial Screen 3	40
5.5	Guest screen	41
5.6	Sign up	42
5.7	Log in	42
5.8	Verify by Email	43
5.9	OTP screen	43
5.10	Create New Password	44
5.11	OTP screen	44
5.12	home screen	45
5.13	All Categories	46
5.14	Crops	46
5.15	Fertilizers	47
5.16	Examble of fertilizers screens	47
5.17	Plants	48
5.18	Examble of plants screens	48
5.19	Soil analyzer model screen	49

5.20	Result screen	49
5.21	Chat bot in Arabic	50
5.22	Filter Options screen	51
5.23	Result of filtering screen	51
5.24	Notifications	52
5.25	Main profile screen	53
5.26	Edit profile screen	53
5.27	About us screen	54
5.28	Privacy policy screen	54
5.29	Contact us screen	55
5.30	FAQS screen	55

List of Tables

2.1	Non-Functional Requirements	12
2.2	Non-Functional Requirements	13
4.1	Libraries Used in Soil Fertility Classification	32
4.2	Libraries and Tools for API Deployment	35
6.1	Test Cases for Registration Process	60
6.2	Login Test Cases	61
6.3	User Test Cases – Functional testing for user actions	62
6.4	User Profile Test Cases	63

Chapter 1

Introduction

Smart Land is a mobile application designed to serve as an intelligent companion for farmers, agricultural engineers, and anyone involved or interested in agriculture. The application aims to modernize and simplify agricultural decision making through artificial intelligence, data-driven tools, and accessible information.

1.1 Motivation

The motivation behind developing Smart Land comes from the challenges many farmers and agricultural workers face in accessing accurate, timely, and affordable agricultural guidance. In many areas, expert advice is either unavailable or difficult to obtain, leading to poor soil management, ineffective fertilization, and reduced crop productivity. With the growing capabilities of artificial intelligence and mobile technologies, our objective was to create a smart, accessible solution that provides farmers and agriculture enthusiasts with instant support, whether it's answering questions, analyzing soil fertility or learning about best farming practices, all from the palm of your hand.

1.2 Project Objectives

This section outlines the main objectives of the proposed project “Smart Agri Assistant.” These objectives are practical, interrelated, and achievable within the available time and resources. The

project aims to improve the agricultural experience for farmers, agricultural engineers, and enthusiasts through intelligent technology. The main objectives include:

- **Developing an AI-powered agricultural chat bot** Capable of interacting in Arabic and English, providing accurate answers on topics like crops, soil, fertilizers, and irrigation.
- **Providing soil fertility analysis** Using a machine learning model to analyze values such as nitrogen, phosphorus, potassium, and soil pH, and determine fertility levels.
- **Delivering real-time agricultural guidance** Offering practical recommendations on fertilization, irrigation, and crop care based on user queries.
- **Improving knowledge accessibility** Making expert agricultural information available to users in remote or underserved areas where consulting specialists is difficult.
- **Reducing dependency on traditional agricultural extension services** By offering personalized, on-demand guidance for users in the field.
- **Enhancing decision-making** Through educational content and sharing of best practices on a wide range of agricultural topics.
- **Delivering a smooth and user-friendly mobile experience** Including a rich and searchable knowledge base of plants, crops, fertilizers, and pesticides.

These objectives collectively aim to empower users with reliable, up-to-date information and tools that support smarter and more efficient farming practices.

1.3 Introduction to project

1.3.1 Scope

The application includes an AI chat bot trained on agricultural topics, a soil fertility analyzer, detailed information on crops and fertilizers, a FAQ section, and a clean, accessible design. It targets farmers, students, and professionals, offering them practical, on-demand support for better farming outcomes.

1.3.2 Model

The project was built using systematic methods and programs, which helps to complete the review and testing phase with each step of the project.

1.3.3 Functional Components of the Project

The Smart Assistant application is composed of several key functional components, each designed to serve a specific role in supporting agricultural users. These components include:

- **AI Chat bot Interface:** A smart conversational agent that answers users' agricultural questions in both Arabic and English, trained on a large dataset covering soil, crops, fertilizers, irrigation, and more.
- **Soil Fertility Analyzer:** A machine learning model that allows users to input soil data (e.g., Nitrogen, Phosphorus, Potassium, pH) and get instant classification of fertility level (High – Medium – Low), helping them make better planting decisions.
- **Knowledge Base:** A structured section providing detailed information about different plants, vegetables, fruits, planting methods, suitable environments, fertilizers, pesticides, and their safe usage.
- **FAQ Section:** A collection of commonly asked agricultural questions with their accurate answers, accessible offline for quick reference.
- **User-Friendly Mobile UI:** A clean and intuitive mobile interface designed to ensure smooth navigation and ease of use for farmers and general users

These components work together to provide an integrated digital assistant for agriculture, focused on usability, accessibility, and practical decision-making.

1.3.4 Technologies

The development of Smart Land integrates several modern technologies to build a scalable, intelligent, and user-friendly mobile solution. Figma was employed during the UI/UX design phase to

prototype and visualize the application layout, ensuring an intuitive and visually appealing user experience. Flutter used to develop a cross-platform mobile application with a smooth and responsive user interface for both Android and iOS. Python employed for building and training AI models, including the smart agricultural chat bot (based on LLM) and the soil fertility prediction model. FastAPI backend framework for building lightweight RESTful APIs that connect the app with AI and database services, .NET (ASP.NET Core) used for developing backend services and potential admin dashboards, offering robust server-side logic and scalability. SQL Database relational database system used to manage and store structured agricultural data, including soil records, user profiles, FAQs, crops, and fertilizer details.

1.4 Organizing Profile

1.4.1 Range of Experts

- UI/UX Designers
- Agricultural Consultants
- AI and Machine Learning Engineers
- Mobile Developers
- Backend Developers
- Database Engineers

1.4.2 Feasibility Analysis

Constraints (Problems in Existing Systems)

- **Lack of Smart Interaction:** Most agricultural apps lack intelligent conversational capabilities, making them difficult to use for non-expert users or those with limited literacy.
- **Limited Local Relevance:** Many existing platforms are not tailored to local agricultural conditions, language, or crops, reducing their usefulness for farmers in specific regions.

- **Inaccessible Expert Advice:** Farmers often struggle to get timely and personalized recommendations without direct access to agricultural engineers or specialists.
- **No Soil Fertility Prediction:** Few apps provide automated tools to analyze soil sample data and evaluate land fertility based on key indicators like nitrogen, phosphorus, and pH.
- **Fragmented Information:** Users have to search across different platforms to find data on fertilizers, crops, irrigation methods, and common plant diseases.
- **High Cost or Complexity:** Some solutions require paid subscriptions, special equipment, or complex usage steps, making them impractical for small-scale farmers.

Marketing

Our marketing plan focuses on reaching users quickly and raising awareness of the app's role in simplifying agriculture. We aim to attract farmers and agri-enthusiasts by highlighting the app's ability to provide expert knowledge efficiently, saving both time and effort.

Financial Analysis

We estimate costs for the development, maintenance, marketing, and support of **Smart Land**. These include expenses for: app design and programming, AI model training and hosting, marketing campaigns (especially in rural areas), and continuous support and updates. Potential revenue streams include partnerships with agricultural organizations, premium features, and sponsored content.

Legal and Ethical Considerations

We ensure full compliance with data privacy regulations, especially for handling user-submitted soil data and chat interactions. Ethical design principles are followed to provide accurate and responsible agricultural advice.

User Feedback and Testing

User testing is planned with farmers and agricultural experts to assess the usability and usefulness of the chat bot and soil analyzer. Feedback will guide continuous improvements for better performance and engagement.

Organizational

The goal is to complete the first phase of the project on time and implement the marketing plan efficiently. Internal collaboration and clear policies will ensure smooth operation and consistent updates.

Technical

Hardware specifications: All devices can use the application.

Software specifications: The software used to build the application is based on free and open-source tools.

1.5 Contributions of This Study

- **Innovative Agricultural Support:** *Smart Land* introduces a smart chat bot and soil analyzer to offer accurate, AI-powered agricultural guidance.
- **User-Centric Design:** Designed with farmers and agricultural engineers in mind, ensuring the app meets real-world needs and challenges in farming.
- **Feasibility Assessment:** Provides a comprehensive analysis of the project's technical, market, and financial viability to support long-term sustainability.
- **Support for Agricultural Research:** Agricultural experts and researchers can use the chat bot and soil data to enhance studies, expand datasets, and improve model accuracy over time.
- **Scalable Impact:** The app has the potential to transform how agricultural knowledge is accessed—empowering users with fast, localized, and scientifically sound advice across regions.

Chapter 2

System Requirements and Analysis

2.1 Introduction

After analyzing the requirements of the project to be performed, the next step is to analyze the problem and understand its context.[1]

2.2 Analysis model

Analysis is the process of gathering and interpreting facts, diagnosing problems and using the information to recommend improvements on the system. System analysis is a problem-solving activity that requires intensive communication between the system users and system developers. System analysis or study is an important phase of any system development process. The system is viewed as a whole, the inputs are identified and the system is subjected to close study to identify the problem areas. The solutions are given as a proposal. The proposal is reviewed on user request and suitable changes are made. This loop ends as soon as the user is satisfied with the proposal. Process model suitable is Agile Software Development.

- Adaptive approach which is able to respond to the changing requirements of the clients.
- Direct communication and constant feedback from customer representatives leave no space for any guesswork in the system.
- Basic functionality is delivered quickly.

- Provides easy access by developers to end users.
- Projects are delivered on time and within a specific budget.

2.2.1 Requirements

A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and programmer

2.3 Study of the System

2.3.1 GUI's (Graphical User Interfaces)

The application includes a set of intuitive and responsive GUI's designed to offer a seamless and accessible user experience for all types of users, especially farmers and agricultural professionals. These interfaces aim to simplify navigation and ensure that users can easily benefit from the system's core features.

- **Chatbot Interface:** A clean and interactive screen that allows users to ask agricultural questions and receive real-time answers from the AI assistant.
- **Soil Analysis Input Page:** A structured form where users can input soil test values (e.g., nitrogen, phosphorus, pH) and get instant feedback on soil fertility.
- **Knowledge Base Section:** Categorized pages displaying detailed information about crops, fertilizers, plants, irrigation methods, and more.
- **Settings/Profile Page:** Allows users to manage preferences, language, and personal settings.

2.3.2 Number of Modules

- **Registration Module**
- **Smart ArgiBot Module**

- **Soil Fertility Analyzer Module.**
- **Knowledge Base Module**
- **FAQs and Common Solutions Module**
- **User Interface and Profile Management Module**

2.3.3 Entities Involved in the Project

User (Farmer / Agricultural Engineer / General User)

- Register and log in to the application.
- View and update personal profile information.
- Interact with the Smart Agricultural Chatbot to ask agricultural questions.
- Manually enter soil test data (e.g., Nitrogen, Phosphorus, Potassium, pH) to get fertility analysis results.
- Browse the Knowledge Base for information on crops, fertilizers, and plants.
- View FAQs and common agricultural issues with their solutions.
- View recommendations based on chat or soil results.
- Receive notifications for important updates, tips, and alerts.
- Manage personal preferences and app settings.
- Log out from the application.

2.3.4 System Requirements

2.3.4.1 Hardware Requirements

- 2.8 GHz Processor or more
- 8 GB RAM or more

- Strong GPU (A-100 recommended) for AI model implementation

2.3.4.2 Software Requirements

- Windows OS (10)
- Figma
- Visual Studio Code / Visual Studio / Kaggle Notebook
- ASP.NET Core (.NET)
- SQL Database
- Python
- Flutter SDK
- Ngrok (for deployment/testing)

2.4 Software Requirements Specification

2.4.1 Introduction to SRS

Purpose: The primary purpose of this document is to define the operating characteristics, functional and non-functional requirements of the **Smart Land** system. It ensures that all stakeholders—developers, testers, and project supervisors—have a unified understanding of what the system is expected to do. This clarity helps avoid misunderstandings and ensures smoother progress throughout the project lifecycle.

Scope: This Software Requirements Specification (SRS) document plays a critical role in the Software Development Life Cycle (SDLC). It outlines in detail the full set of requirements for the **Smart Land** application, serving as a foundation for design, development, and testing. It is intended for use by developers, testers, and maintainers of the system. Any modifications to the system requirements after initial approval will be subject to a formal change control process to maintain consistency and traceability.[2]

2.4.2 Functional Requirements

- Users must be able to register, log in, and log out of the application securely.
- An email-based OTP verification system shall be used during account creation.
- Users should be able to reset their password in case of loss or change.
- Users can view and update their personal profile information at any time.
- Users can interact with the smart agricultural assistant chat bot to ask farming-related questions.
- The chat bot shall respond with clear, accurate, and context-aware agricultural advice (e.g., about crops, fertilizers, soil, irrigation).
- Users can input soil data (e.g., nitrogen, phosphorus, potassium, pH) manually through a form.
- The system shall analyze this data and return a fertility classification (e.g., High, Medium, Low) along with actionable recommendations.
- Users can browse a rich database of agricultural topics, including crop guides, plant diseases, fertilizer types, and pest control methods.
- Users will receive timely notifications related to tasks, recommendations, crop cycles, or reminders.
- The system will provide personalized agricultural recommendations based on user activity or soil data.

All data transactions operations made by user should be done through the usage of API link

2.4.3 Non-Functional Requirements

Table 2.1 – Non-Functional Requirements

Requirement	Description	Must
System Structure	The application must follow a modular and scalable architecture to ensure that future features can be added with minimal impact.	✓
	The system must integrate a well-structured SQL database to efficiently store user profiles, soil data, recommendations, FAQs, and other agricultural information.	✓
	Data storage must avoid redundancy through normalization and optimized schema design.	✓
Availability	The system must be available 24 hours a day with no bandwidth issues. If something goes wrong, the app should keep working without major disruptions.	✓
Performance	The system should only transmit essential data in API responses to minimize response time.	✓
	All interactions, such as taps or button clicks, must trigger fast responses without noticeable delays.	✓
	The overall system should feel responsive and interactive, minimizing loading times and avoiding freezes or lags.	✓
Usability	The app is easy to use, even for people who aren't good with technology.	✓
Reliability	The app should work all the time without crashing or freezing.	✓
Errors, Exception and Handling	Ensure that error messages are clear to the user.	✓
	All error responses must be handled gracefully to avoid application crashes.	✓

Table 2.2 – Non-Functional Requirements

Requirement	Description	Must
Security and Encryption	The system uses SSL (secured socket layer) in all transactions that include any confidential customer information.	✓
	Passwords are saved and secured using encryption algorithms.	✓
	Verification mail will be sent to users via email.	✓
	The system should use a security protocol when sending data over the internet.	✓

Chapter 3

System Design

3.1 Introduction

System design is a fundamental phase in the software development life cycle where the conceptual solution is transformed into a detailed blueprint ready for implementation. It defines the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements.

The primary goal of system design is to ensure that the developed system meets both functional and non-functional requirements with high efficiency, scalability, and maintainability. This phase involves making important decisions regarding how the system will be structured, how components will interact, and how data will be stored and processed.

Design activities typically include architectural design, which lays out the overall structure of the system; interface design, which defines communication between system components; data structure design, which organizes data for optimal access and management; and procedural design, which outlines algorithms and process flows.

By carefully planning the system's structure in advance, potential issues can be identified early, development efforts can be streamlined, and the resulting system will be more robust, adaptable, and easier to test and maintain.

3.2 Design

In the design phase of the **Smart Agricultural Assistant System**, several visual and structural design principles were considered to enhance usability and clarity. These include *emphasis*, *balance and alignment*, *contrast*, *repetition*, *proportion*, *movement*, and *white space*, all of which contribute to a consistent and user-friendly interface across the mobile application.

From a software architecture perspective, the system is structured using a **multi-layered architecture**, which is commonly adopted to ensure separation of concerns and facilitate scalability and maintainability. The architecture is divided into the following layers:

- **Presentation Layer:** This layer includes the user interface components of the mobile application. It handles user interactions such as chatting with the agricultural assistant, submitting soil data for analysis, and browsing information about crops, fertilizers, and best practices. It is responsible for capturing user input and displaying relevant outputs or responses.
- **Service Layer:** This acts as the intermediary between the user interface and the core functionalities of the system. It manages API requests, handles communication with external services (e.g., AI model APIs and backend servers), and ensures secure and efficient data exchange.
- **Business Layer:** The core logic of the system resides here. It includes the intelligent decision-making process of the AI chatbot, soil fertility classification algorithms, and rules for agricultural recommendations based on the user's input. This layer ensures the system behaves according to its intended purpose.
- **Data Layer:** This layer is responsible for storing and managing application data, including user information, chat history, and knowledge base entries (e.g., fertilizer types, crop information). It interacts with local databases to ensure reliable data access and storage.

This layered design supports the modular development of the system, facilitates testing, and allows future expansion such as integrating IoT sensors or additional AI models.

3.3 UML

- Behavioral UML Diagram
 - ▶ Use Case Diagram
 - ▶ Activity Diagram
 - ▶ Sequence Diagram
- Structural UML Diagram
 - ▶ Class Diagram
- DFD-data flow diagram
 - ▶ Context diagram
 - ▶ Level 1
- ERD Diagram

3.3.1 Behavioral UML Diagram

3.3.1.1 Use Case

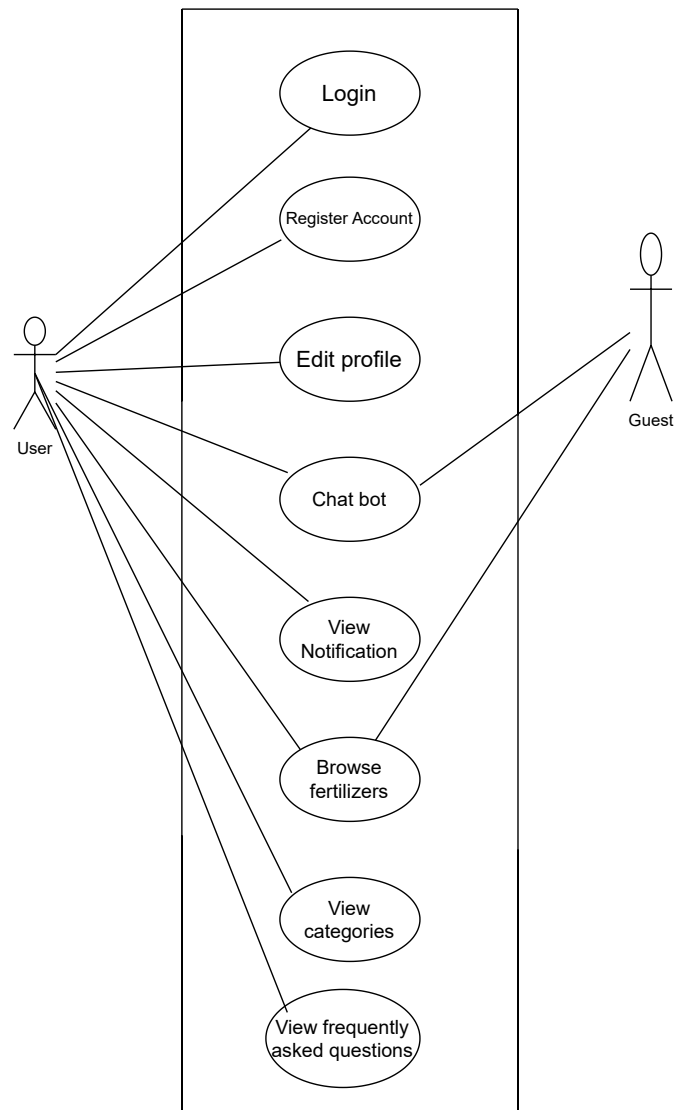


Figure 3.1 – Use Case

3.3.1.2 Activity Diagram

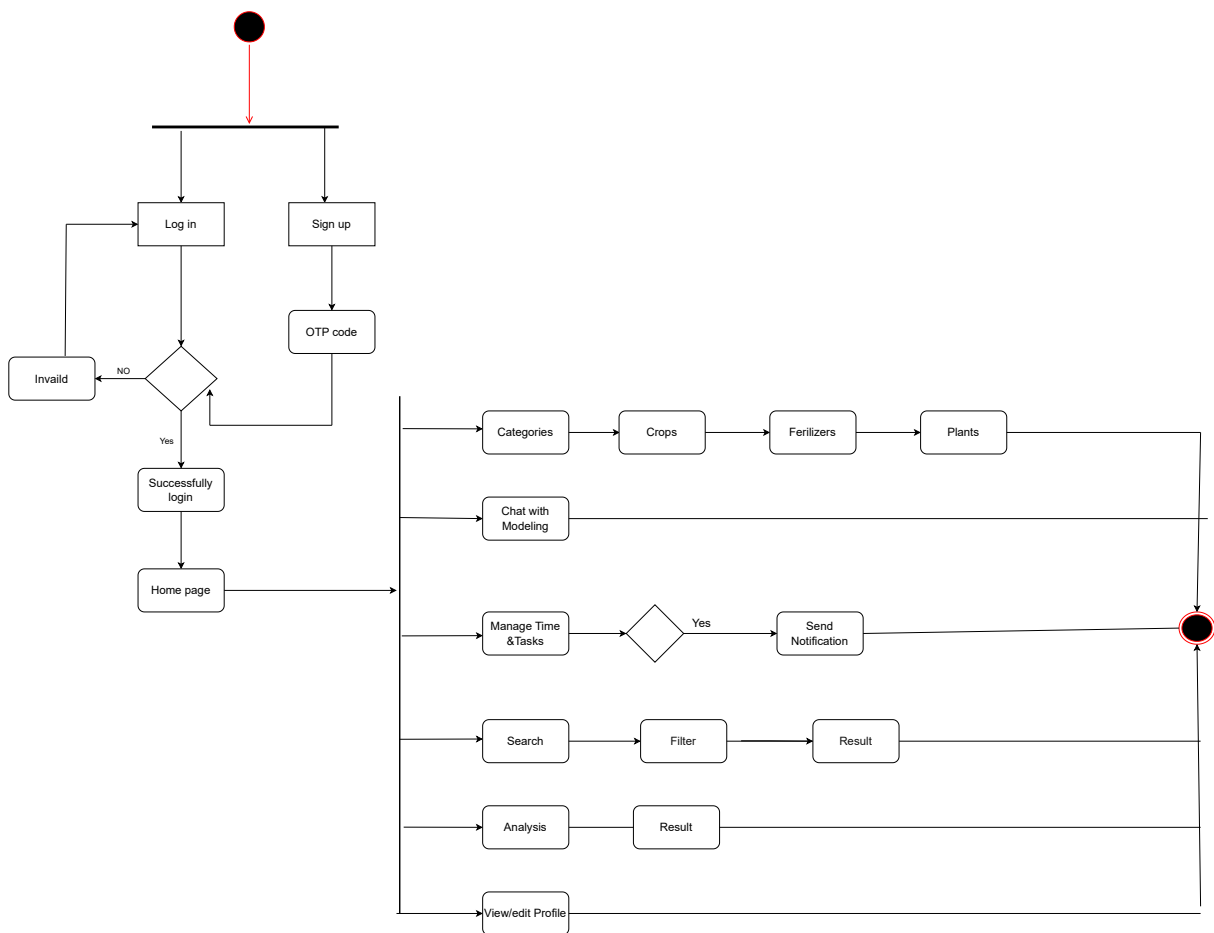


Figure 3.2 – user activity diagram

3.3.1.3 Sequence Diagram

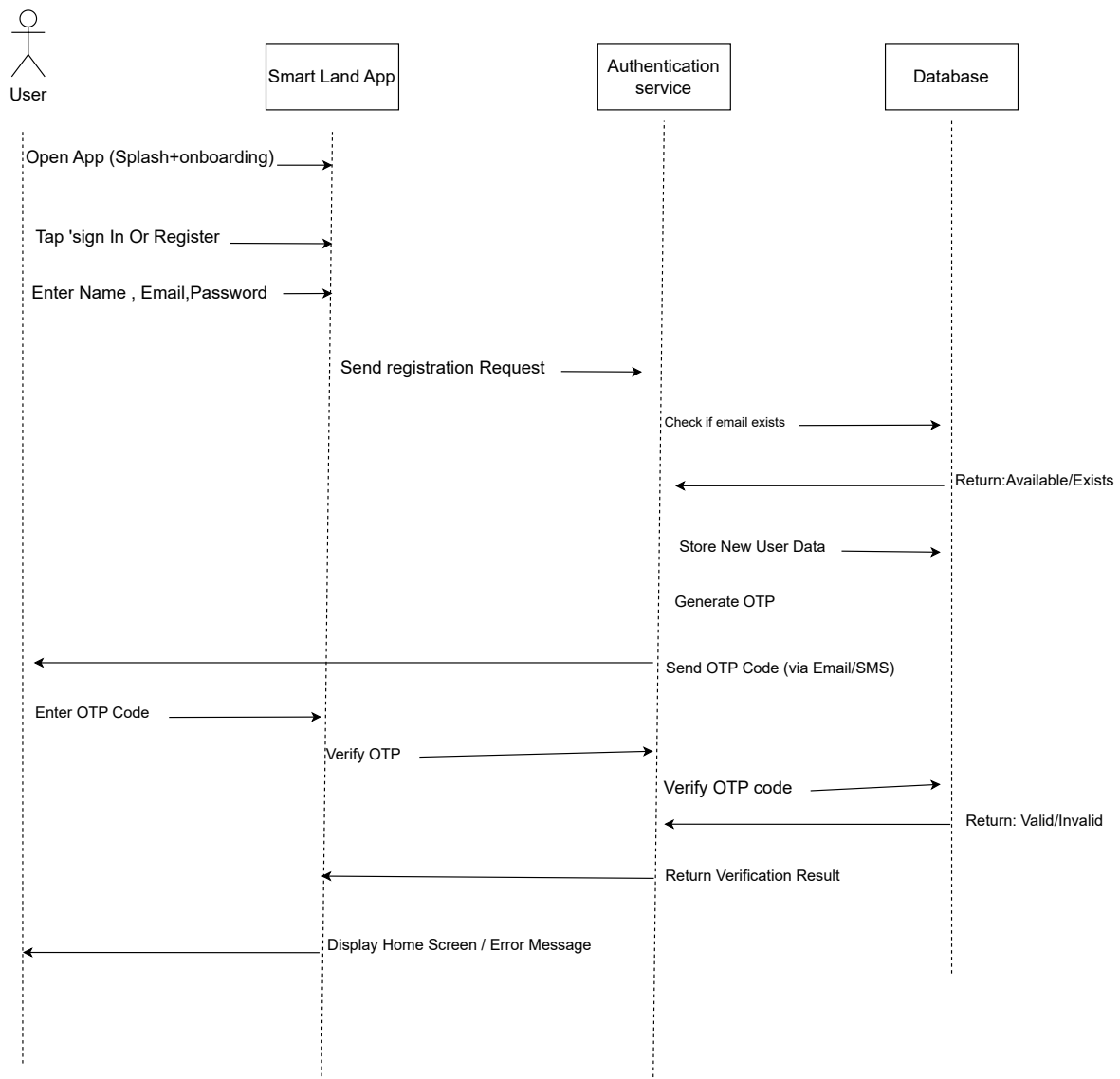


Figure 3.3 – User Registration

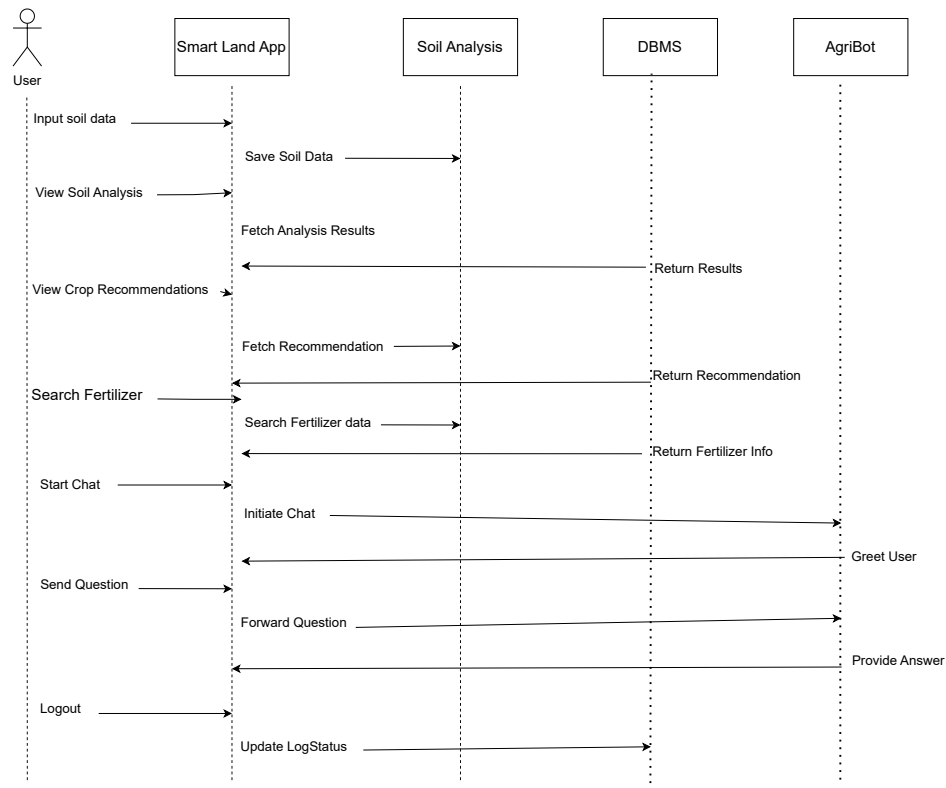


Figure 3.4 – User Functionality

3.3.2 Structural UML Diagram

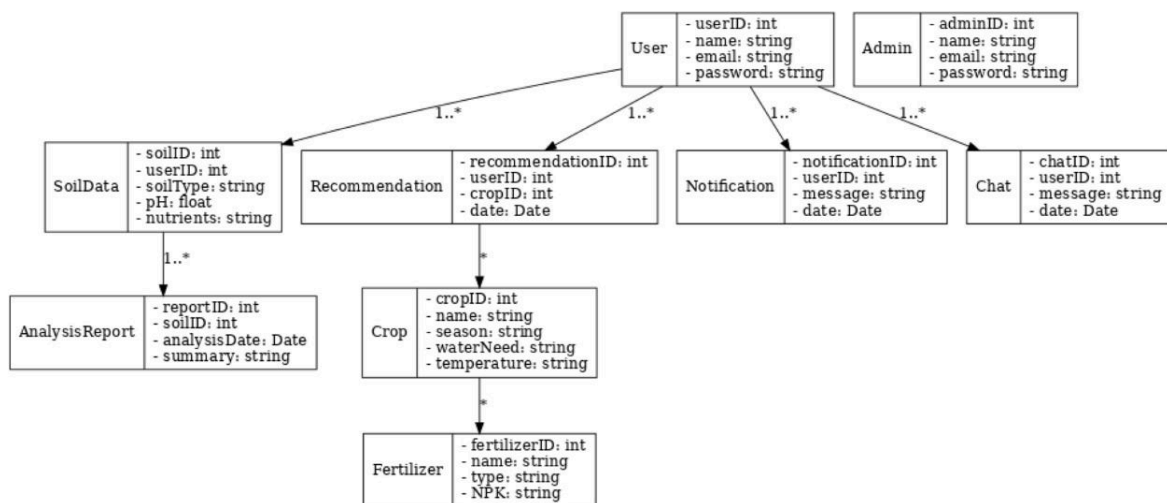


Figure 3.5 – Class diagram

3.3.3 DFD-data flow diagram

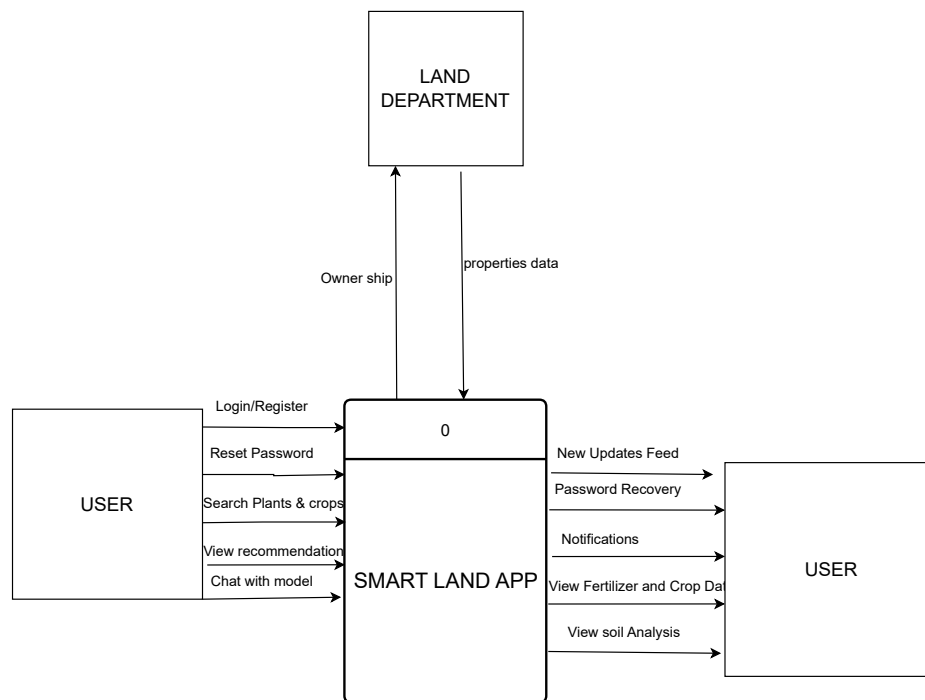


Figure 3.6 – Context diagram

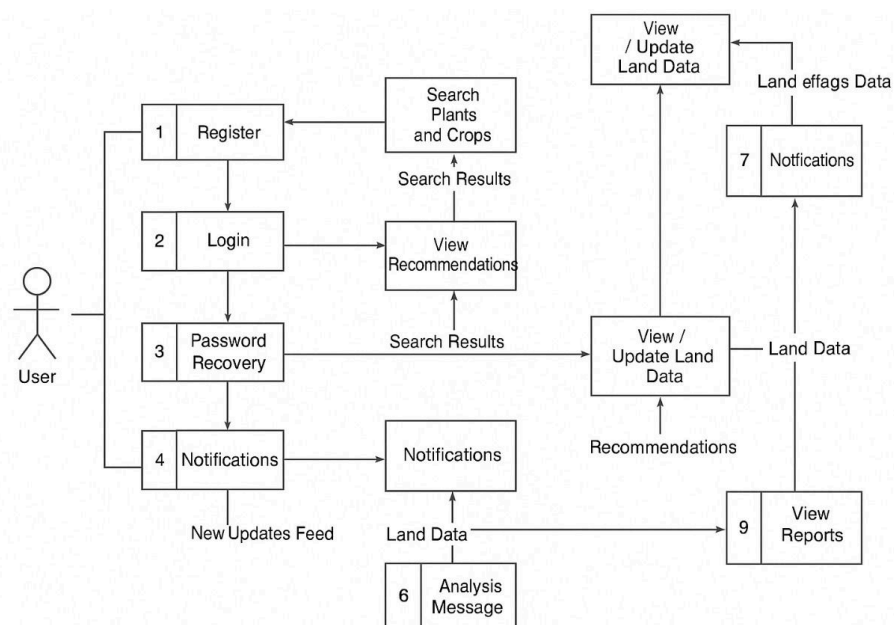


Figure 3.7 – Level 1

3.3.4 ERD Diagram

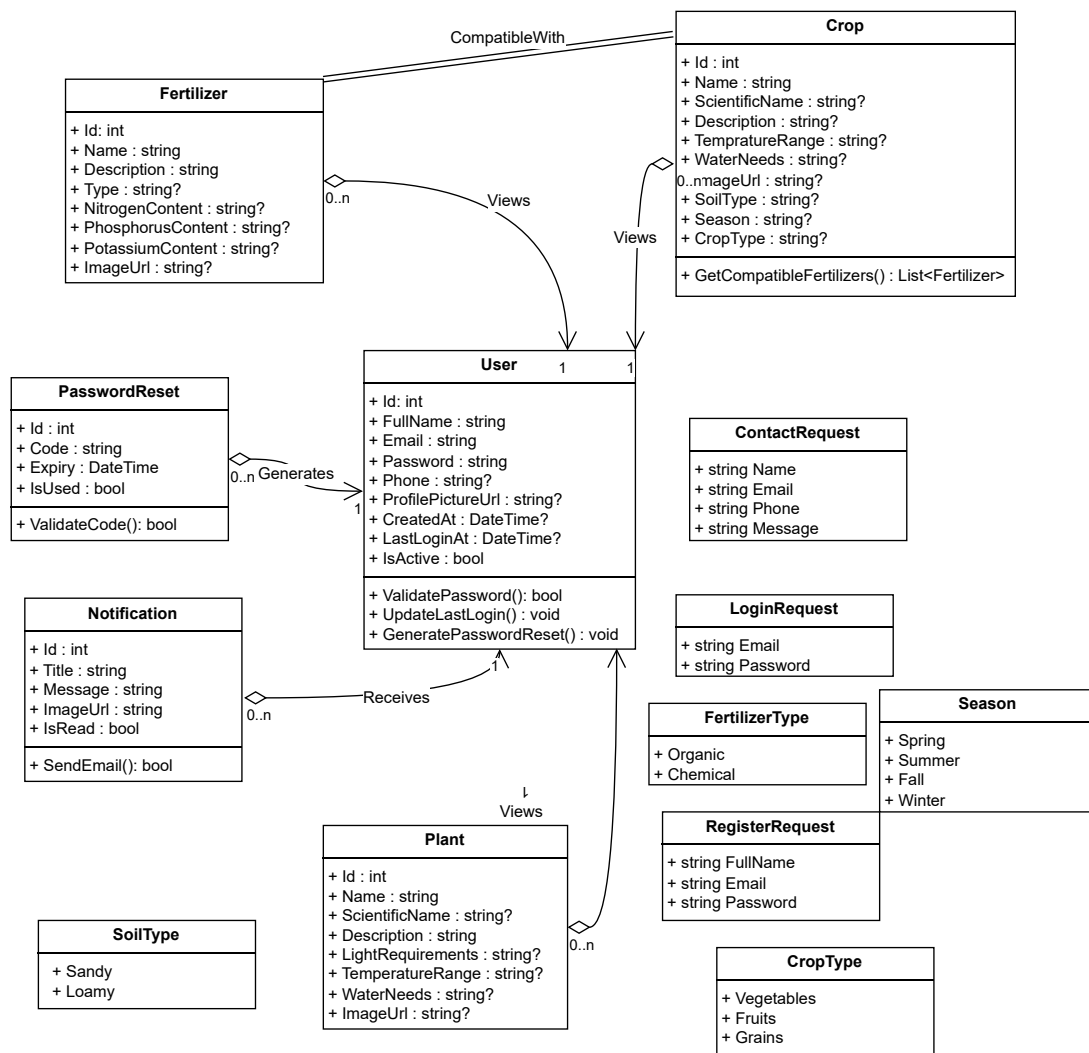


Figure 3.8 – Database

Chapter 4

Software and Development Tools

4.1 Introduction

This chapter presents the software tools, libraries, and platforms that were utilized throughout the development of the Smart Agricultural Assistant System. The selection of these tools was based on their compatibility with the system requirements, ease of integration, reliability, and community support.

The software environment includes programming languages, development frameworks, machine learning libraries, database management systems, and deployment tools. Each software component played a specific role in the design, implementation, testing, and deployment phases of the system.

By clearly identifying the technologies used, this chapter provides insight into the technical foundation of the project, and it ensures reproducibility, maintainability, and scalability for future enhancements or research based on this work.

4.2 System Development Tools

This section outlines the software tools, libraries, frameworks, and technologies utilized in the development of the Smart Agricultural Assistant System. Each component played a specific role in building the mobile application, backend services, and AI integration.

4.2.1 Figma

Figma is a collaborative interface design tool that was used extensively during the design phase of the Smart Agricultural Assistant System. It enabled the team to create high-fidelity prototypes, wireframes, and user interface mockups for the mobile application before actual development began. By providing a shared design environment, Figma allowed designers and developers to collaborate in real-time, ensuring consistency between the design and implementation.

Figma's features such as component libraries, design systems, and interactive prototyping helped accelerate the design process while maintaining visual coherence across the app. The tool was particularly beneficial in receiving feedback from stakeholders and making rapid design iterations based on usability concerns.

Using Figma helped the team visualize user journeys, plan screen transitions (such as from the chatbot to soil analysis pages), and align on a unified design language that fits the target users — primarily farmers who need a simple and intuitive interface. It also streamlined handoff to Flutter developers by allowing easy export of design specs, assets, and CSS properties.[3]

4.2.2 Mobile Application (Flutter)

The mobile application was developed using the Flutter SDK. A variety of dependencies were integrated to support essential functionalities:

- **Flutter SDK:** The core framework used to develop cross-platform mobile applications.
- **dio:** A powerful HTTP client used for making advanced and secure network requests.
- **http:** A simpler HTTP client, considered but not essential due to using dio.
- **image_picker:** Enables image selection from the device camera or gallery, useful in features like submitting soil photos.
- **cached_network_image:** Optimizes performance by caching network images for reuse.
- **device_preview:** Allows testing and previewing the app layout across multiple devices and screen sizes during development.

- **provider:** A lightweight and scalable state management solution for handling UI state and logic.
- **shared_preferences:** Facilitates local storage of key-value pairs, such as saving login status or user preferences.
- **chat_bubbles:** Provides customizable UI components for building the chat interface used in the AI assistant.
- **animate_do:** Simplifies the addition of animations like fade and zoom effects, enhancing user experience.
- **flutter_spinkit:** A library of animated loading indicators for visual feedback during processing.
- **cupertino_icons:** A set of iOS-styled icons used for consistent and elegant UI design.
- **flutter_localizations:** Provides built-in support for internationalization and localization in Flutter.
- **intl:** Offers internationalization and formatting utilities such as date time and number formatting.[4]

4.2.3 Backend Technologies

The backend of the system was developed using **C# and ASP.NET Core Web API** to ensure performance, scalability, and maintainability. The system provides RESTful APIs to handle core operations such as user management, agricultural data processing, and system interactions.

The backend is designed using a modular and clean architecture, where each model represents a core entity in the application. These backend models are directly mapped to the database, which simplifies development and maintains consistency throughout the system. The architecture is also designed to be extensible for future enhancements such as AI integration, analytics, or third-party API services.[5]

Key backend features include:

- Secure user authentication and authorization.

- Password reset functionality.
- Notification management.
- OTP Gmail verification for user account confirmation.

Database Design

The system uses a **SQL Server** relational database for storing and retrieving structured application data. The database schema follows normalization principles to ensure:

- Data integrity.
- Reduced redundancy.
- Optimized query performance.

Essential entities stored in the database include:

- Users
- Crops
- Fertilizers
- Plants
- Notifications
- Password reset records

Relationships between these entities are maintained through foreign keys, supporting one-to-many associations. The database design is scalable and adaptable for future domains or integrations.[6]

4.2.4 Agricultural Assistant Chat Bot

4.2.4.1 AI Model

The core component of our smart agricultural assistant is an intelligent chatbot powered by the LLAMA3 model. LLAMA3 (Language Model for Multimodal Accessibility), developed by Meta

AI, is a state-of-the-art large language model (LLM) designed to enhance accessibility and understanding across multimodal data sources. It is the latest in the LLAMA series, building on the capabilities of LLAMA1 and LLAMA2.

For this project, we adopted the **LLAMA3 8B-Instruct** model, a fine-tuned version of the base 8B model specifically optimized for instruction-following tasks. This version was selected due to several reasons:

It provides contextual and intelligent responses, which is essential for building a responsive and helpful chatbot, as opposed to static or rule-based systems.

Its instruct fine-tuning enables the model to better understand user queries and follow task-oriented instructions in both Arabic and English, which aligns with our bilingual agricultural use case.[7]

The chatbot,leverages the LLAMA3 architecture to offer natural and context-aware interactions with farmers. It can answer agricultural queries, provide recommendations on fertilizers, irrigation, and soil treatment, and support decision-making based on user input and local conditions.

4.2.4.2 Data Collection and Generation

A bilingual (Arabic-English) and Egyptian colloquial dialect dataset containing approximately 15,000 agricultural questions and answers covering topics such as crops, fertilizers, soil fertility, and pest control was collected from several sources, including the FAO, Kaggle, and Hugging Face databases. to enrich the dataset, additional follow-up questions were generated using AI language models such as ChatGPT. The collected data was then cleaned, structured, and converted into the LLaMA3 fine-tuning format [8]

4.2.4.3 Quantization and Fine Tuning

the LLAMA 3 model was quantized using the bitsandbytes library. Quantization significantly reduces memory usage and computational load by converting model weights to 4-bit precision, allowing the model to fit within the limited resources available in Google Colab and Kaggle environments. After quantization, the model was fine tuned using the **LoRA (Low Rank Adaptation)** method. LoRA enables parameter efficient training by injecting low-rank trainable matrices into selected attention layers of the model, instead of updating the entire model.[9]

4.2.4.4 Training Environment Setup

The training was conducted on **Google Colab** and **Kaggle Notebooks**, utilizing open-source tools and packages. Required libraries such as `transformers`, `peft`, `trl`, `accelerate`, and `bitsandbytes` were installed to enable quantization and fine-tuning. A custom **LoRA (Low-Rank Adaptation)** configuration was defined to fine-tune only specific layers of the LLaMA 3 model, optimizing memory and training efficiency. The **SFT (Supervised Fine-Tuning) Trainer** from Hugging Face was employed to train the model using the prepared dataset.[10]

4.2.4.5 Training

Training was done on a strong Hardware device with a GPU (P 100) , Using the SFT trainer from transformers and configuring suitable parameters as (learning rate, batch size ,epochs , etc), while monitoring the loss curve. Checkpoints were saved and merged as needed to prepare the final LoRA-adapted model. The model was tested locally to verify its ability to answer agricultural questions and engage in meaningful dialogue. The model was trained for 2 epochs with 110 logs obtaining a loss of 1.7:[10]



100	2.146000
110	1.736800
120	1.364900

Figure 4.1 – training loss screen shot

4.2.4.6 Deployment and Hosting

LoRA weights were uploaded to the **Hugging Face Model Hub**, ensuring public or private access. A **Gradio Space** was created to offer an interactive web UI for users to chat with the assistant. To allow API-based access, a **FastAPI server** was built, and **ngrok** was used to tunnel requests, enabling integration with the mobile app frontend (built using Flutter).[11] [12]



Figure 4.2 – Gradio space web UI



Figure 4.3 – Testing API on Postman

4.2.5 Soil Fertility Classification Using Machine Learning

4.2.5.1 Objective

To build a machine learning classification model that predicts soil fertility levels (**Low - Medium - High**) based on the soil's chemical and physical properties.[13]

4.2.5.2 Libraries Used

Library	Purpose
pandas	Data loading and manipulation
numpy	Numerical operations
sklearn.model_selection	Splitting the dataset
sklearn.ensemble.RandomForestClassifier	Main classification algorithm
sklearn.preprocessing.StandardScaler	Feature scaling (normalization)
sklearn.metrics	Model evaluation (accuracy, reports)
joblib	Saving/loading the model and scaler
matplotlib, seaborn	Visualization of confusion matrix

Table 4.1 – Libraries Used in Soil Fertility Classification

4.2.5.3 Project Workflow

1. Data Loading:

The dataset was loaded from a .csv file containing the following features:

- N (Nitrogen)
- P (Phosphorus)
- K (Potassium)
- pH
- EC (Electrical Conductivity)
- OC (Organic Carbon)
- S, Zn, Fe, Cu, Mn, B

The target variable Fertility has three classes:

- 0: Low Fertility
- 1: Medium Fertility
- 2: High Fertility

2. Data Cleaning:

- Checked for missing values (NaN) and removed incomplete rows.
- Split the dataset into:
 - ▶ X – Features (soil characteristics)
 - ▶ y – Target (fertility level)

3. Feature Scaling:

Used `StandardScaler` to normalize the feature values, ensuring better performance for machine learning algorithms.

4. Train/Test Split:

Split the data into:

- 80% training set (`X_train, y_train`)
- 20% testing set (`X_test, y_test`)

Used `stratify=y` to maintain fertility class proportions.

5. Model Training:

Trained a `Random Forest Classifier`, which:

- Is an ensemble of decision trees
- Offers high accuracy and robustness
- Effectively handles multiclass classification problems

6. Model Evaluation:

Evaluated the model using:

- **Accuracy Score:** Percentage of correct predictions
- **Classification Report:** Includes precision, recall, and F1-score
- **Confusion Matrix:** Visualized using Seaborn

7. Model Saving:

Saved using joblib:

- Trained model: `fertility_model.pkl`
- Fitted scaler: `scaler.pkl`

8. Prediction Example:

Tested with a new input sample of 12 soil features and predicted its fertility level using the trained model.

4.2.5.4 Model Performance

The model achieved high accuracy and demonstrated effective class separation. It is ready for deployment in agricultural or smart farming systems.

4.2.5.5 Why Random Forest?

- Handles high-dimensional data effectively
- Less prone to overfitting than single decision trees
- Provides insight into feature importance
- Works well with unbalanced and multiclass datasets

4.2.5.6 Output Files

- `fertility_model.pkl`: Trained Random Forest model
- `scaler.pkl`: StandardScaler instance
- `Soil_Fertility_Classification.ipynb`: Notebook with full implementation

4.2.5.7 Deploying the Soil Fertility Model as an API

Objective To deploy the trained soil fertility classification model as a RESTful API using **FastAPI**, enabling integration with mobile applications such as Flutter.[\[12\]](#)

Technologies Used

Tool / Library	Purpose
FastAPI	Build a high-performance REST API
Pydantic	Define and validate input data schemas
joblib	Load the trained model from file
NumPy	Handle numerical data
Uvicorn	Run the FastAPI server
Pyngrok	Create a public HTTPS URL from local server
nest_asyncio	Resolve async event loop issues in Colab
CORS Middleware	Allow cross-origin requests (e.g., from Flutter)

Table 4.2 – Libraries and Tools for API Deployment

API Workflow

1. **Model Loading:** The trained RandomForestClassifier model (`fertility_model.pkl`) is loaded using `joblib`.
2. **API Setup:** A FastAPI application is initialized, with CORS enabled for integration with external applications.[\[12\]](#)
3. **Endpoints:**
 - GET `/predict`: Accepts query parameters for testing.
 - POST `/predict`: Accepts JSON body with soil parameters and returns predicted fertility level in Arabic.
4. **Colab Integration:** Uses `nest_asyncio` and Python's threading to run Uvicorn server within Google Colab without blocking the notebook.
5. **Ngrok Tunneling:** Pyngrok creates a secure public endpoint, allowing the API to be accessed externally.

Advantages

- Seamless integration with mobile or web applications.
- No need for dedicated deployment infrastructure.
- Supports both GET (for testing) and POST (for production) requests.
- Public access for demonstrations using Ngrok.
- Easy extensibility for future features.

Use Case A mobile app developed in Flutter collects soil properties from the user, sends them via a POST request to the API, and receives a fertility prediction (Low, Medium, or High) displayed in the UI.

Future Enhancements

- Add authentication and access control.
- Log predictions in a connected database.
- Deploy to cloud platforms (e.g., Heroku, Google Cloud).

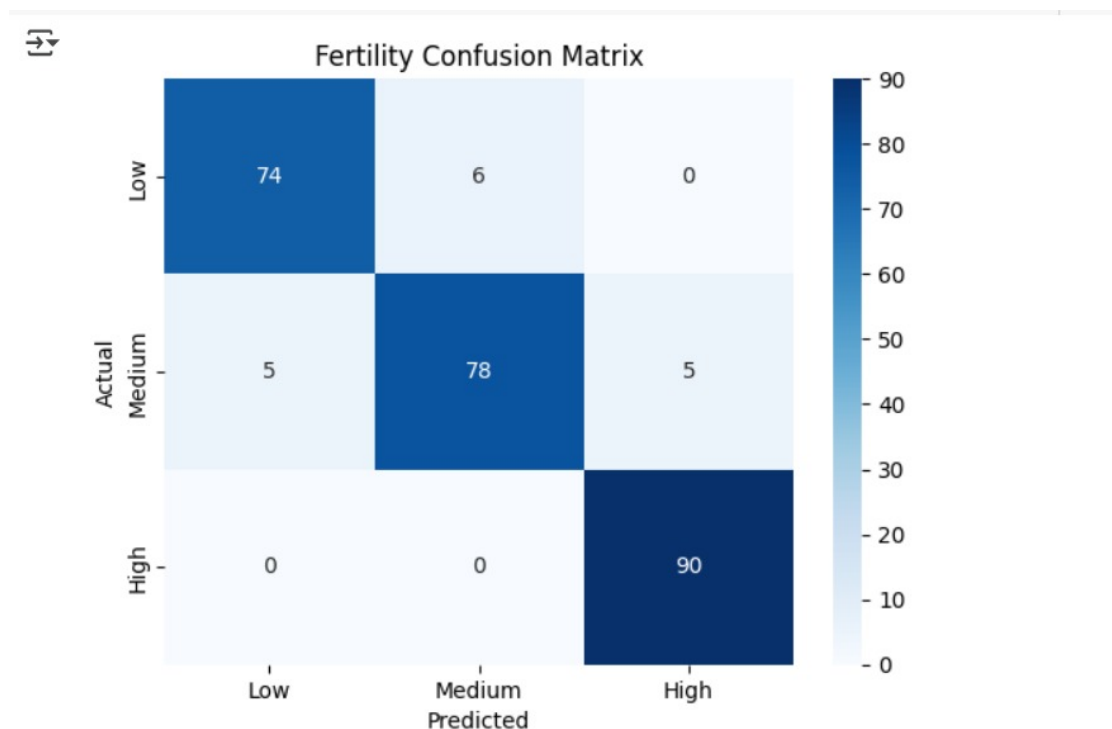


Figure 4.4 – confusion matrix

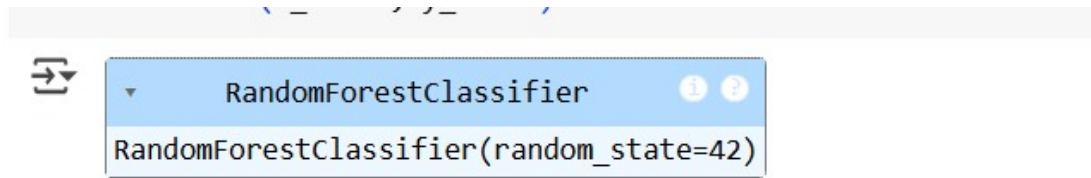


Figure 4.5 – Model download

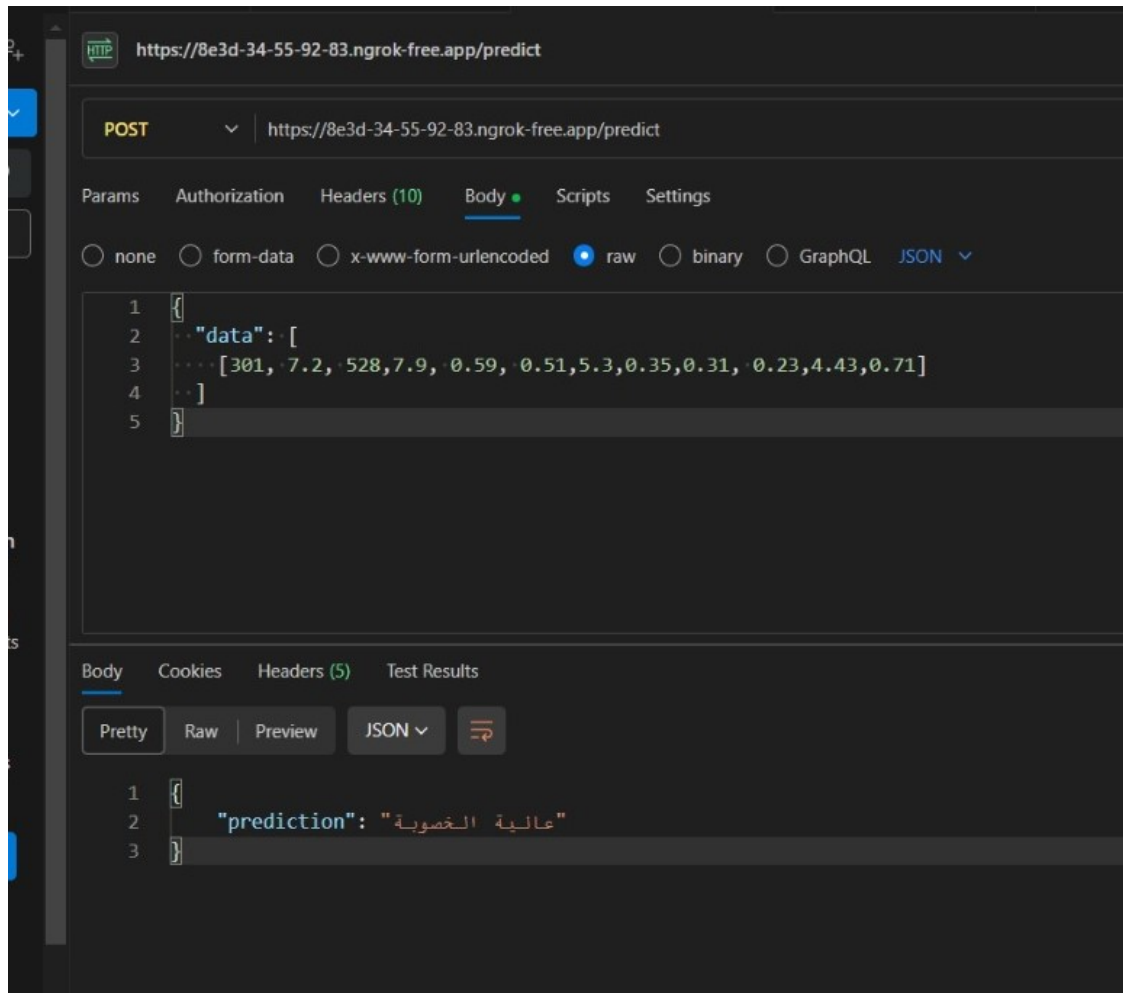


Figure 4.6 – Testing API in postman

Chapter 5

System Implementation

5.1 Initial Screens



Figure 5.1 – Initial Screen Logo

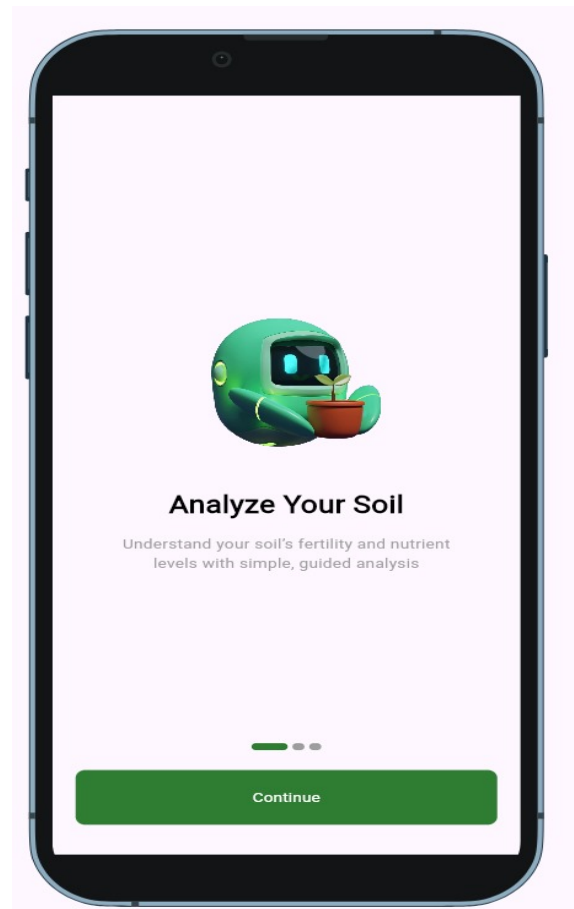


Figure 5.2 – Initial Screen 1

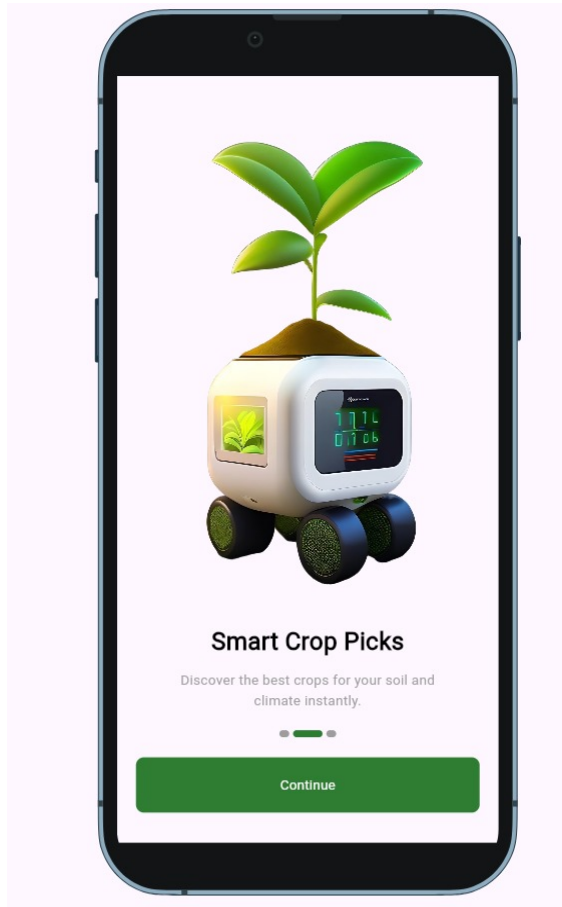


Figure 5.3 – Initial Screen 2

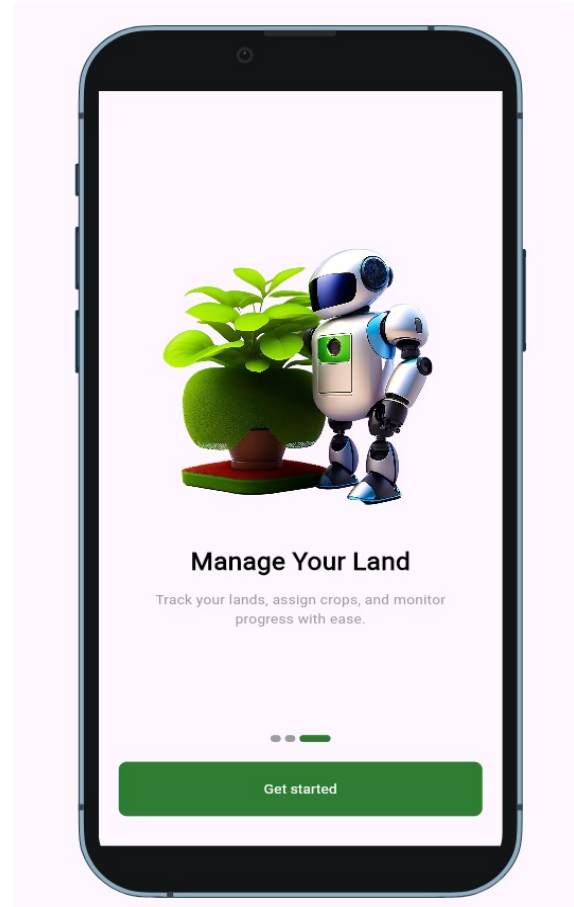


Figure 5.4 – Initial Screen 3

5.2 Gest

The Guest Mode feature in the Smart Land application allows users to access and explore parts of the app without the need to sign up or log in. This enhances user experience, especially for first-time users, by enabling them to preview the app's capabilities before creating an account

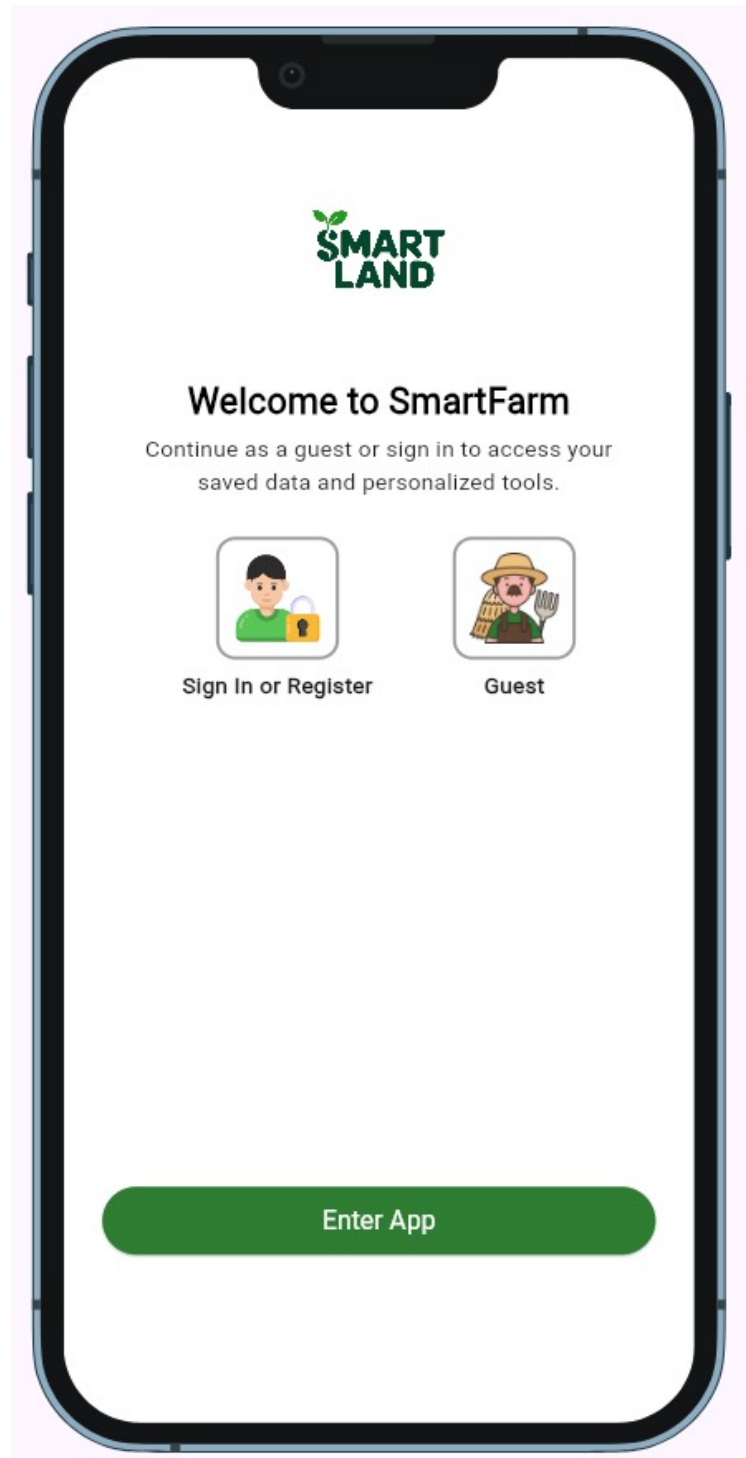


Figure 5.5 – Guest screen

5.3 Sign Up And Log In

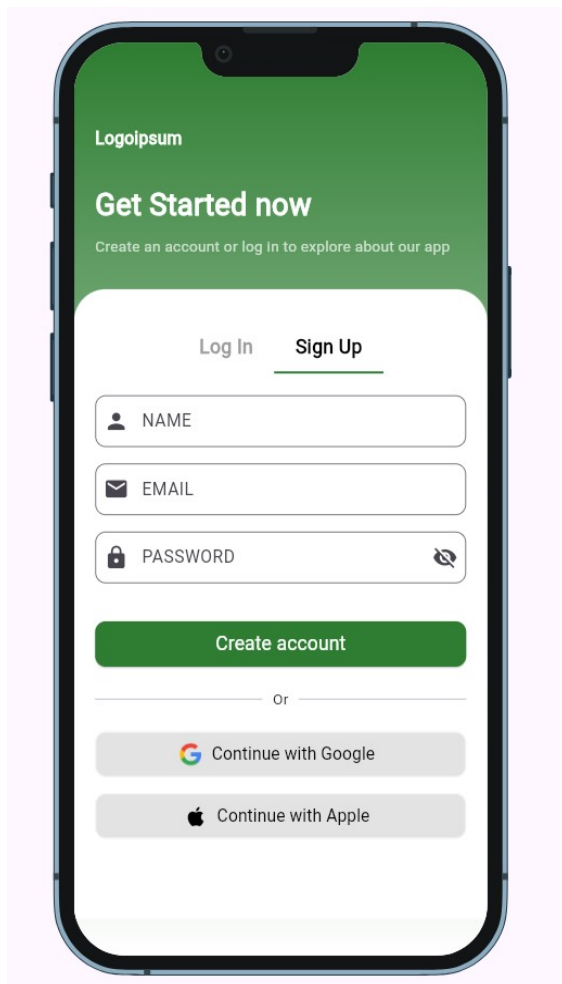


Figure 5.6 – Sign up

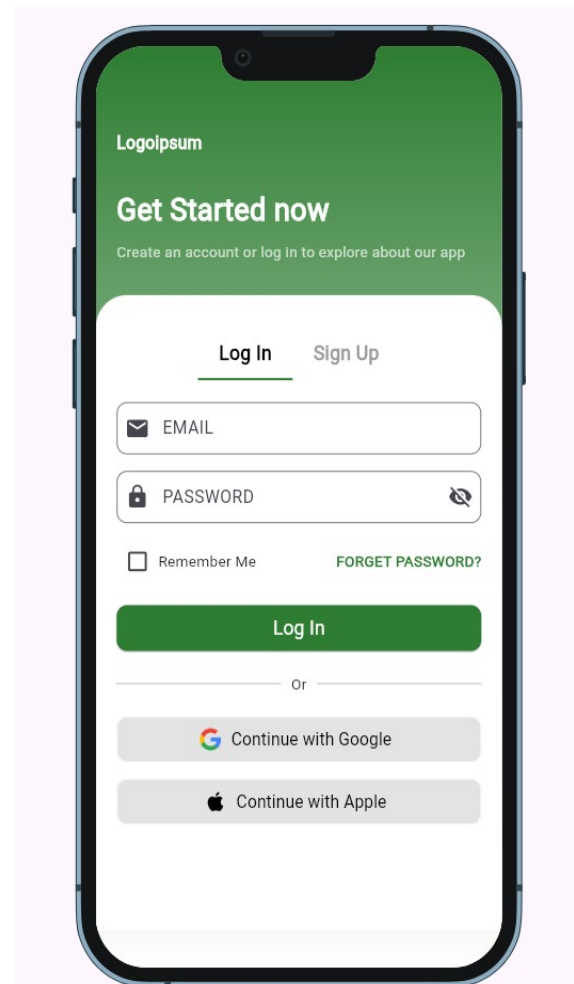


Figure 5.7 – Log in

5.4 Forget Password

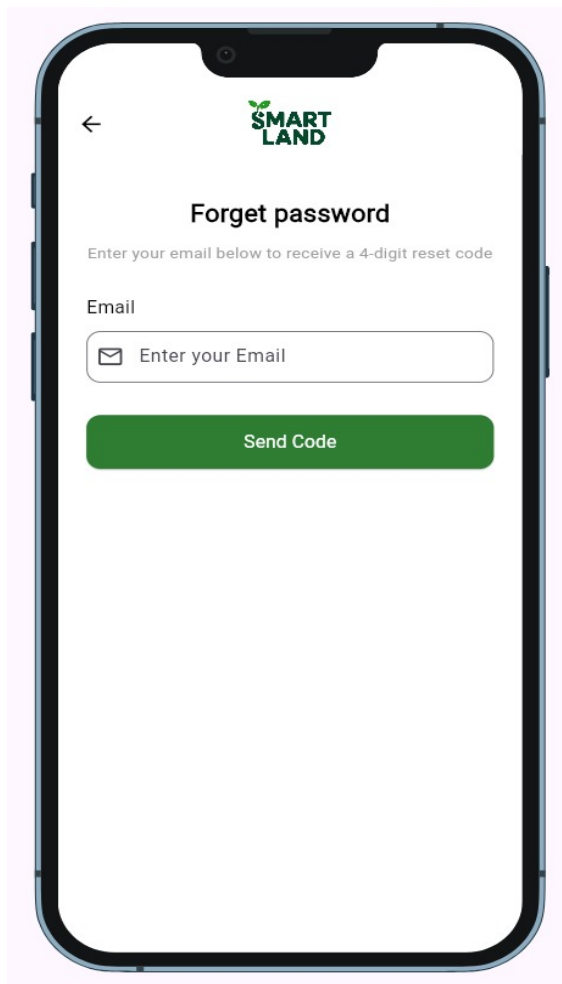


Figure 5.8 – Verify by Email

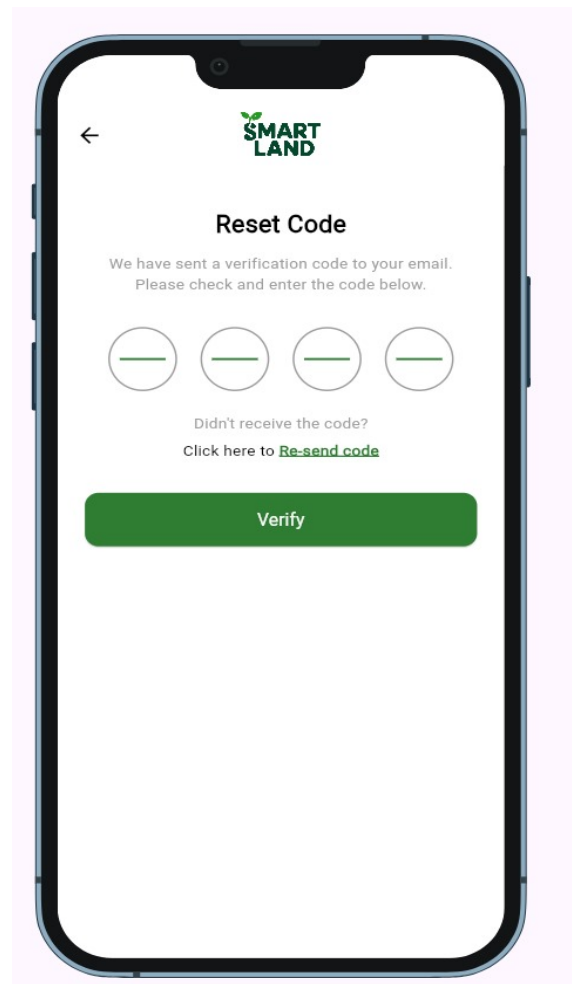


Figure 5.9 – OTP screen

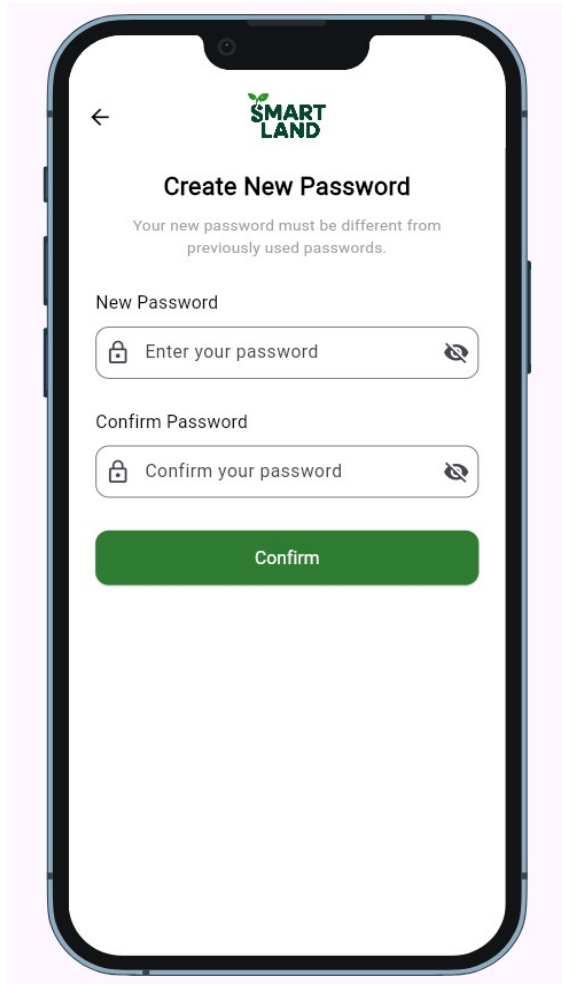


Figure 5.10 – Create New Password

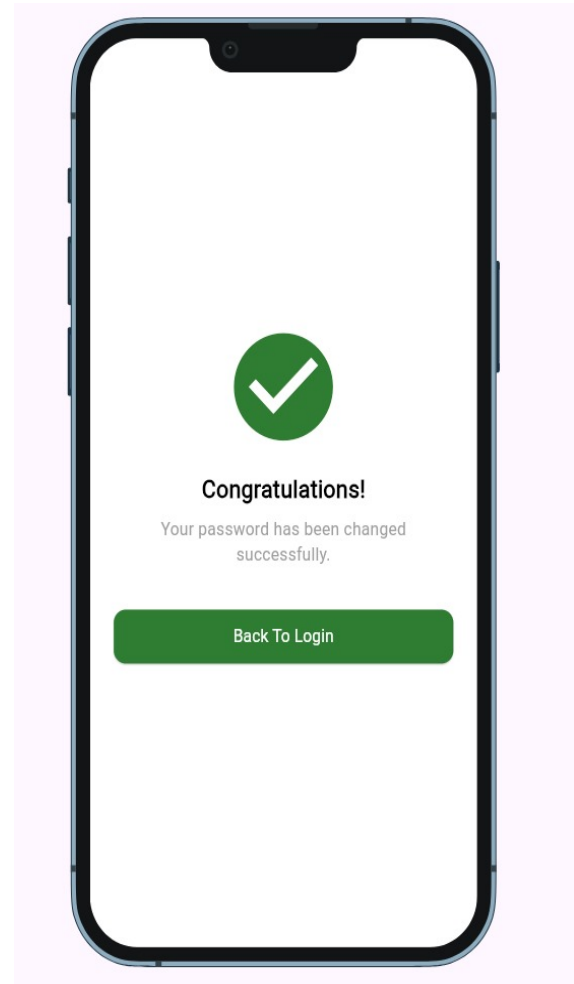


Figure 5.11 – OTP screen

5.5 Home

The home page is the main page in the application, from which the user can access the rest of the pages (profile , categories , soil analyzer , chat bot , search , notification , filter)

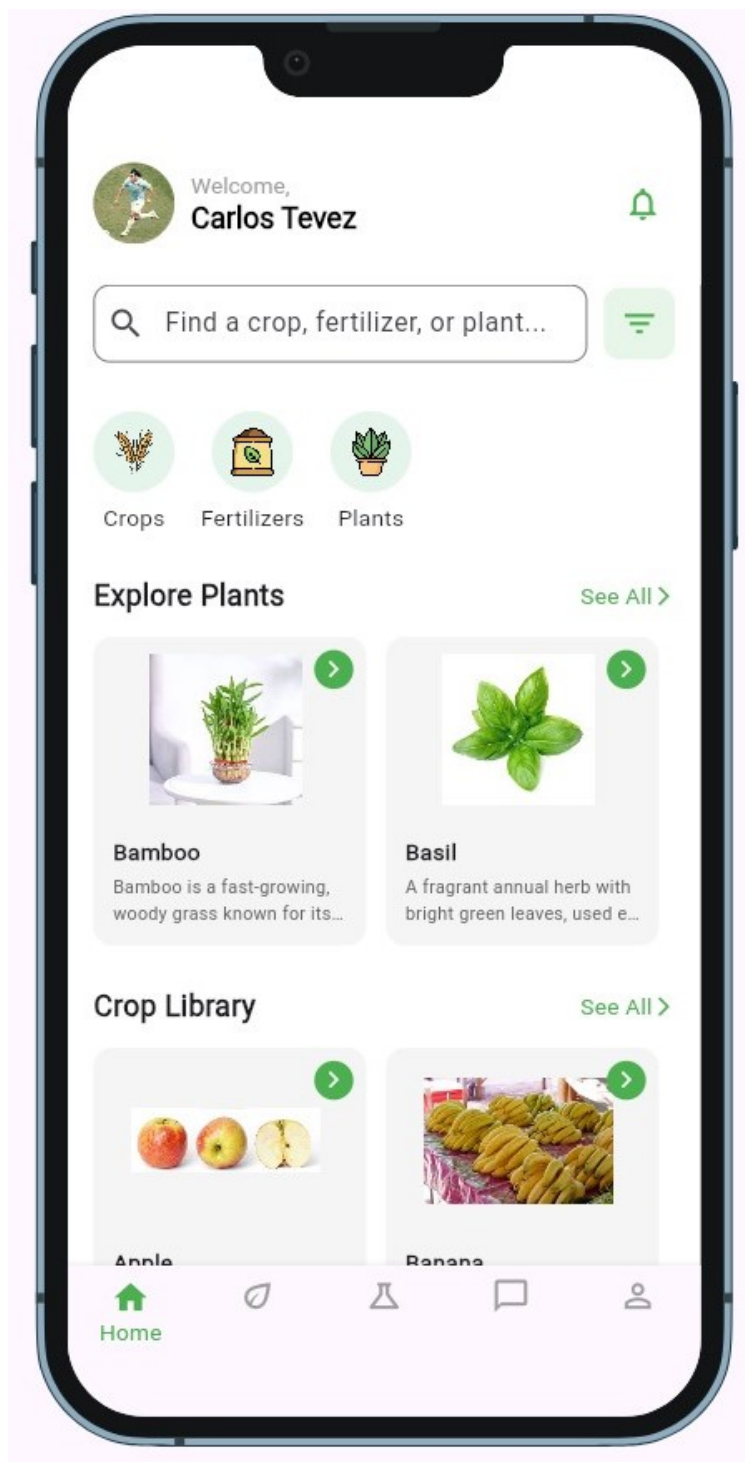


Figure 5.12 – home screen

5.6 Categories

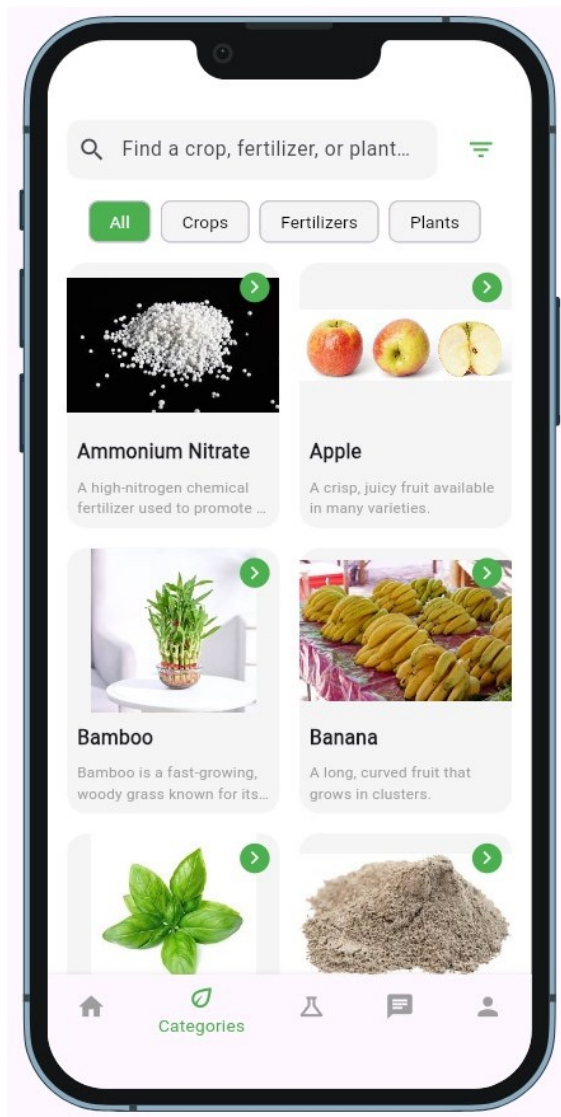


Figure 5.13 – All Categories

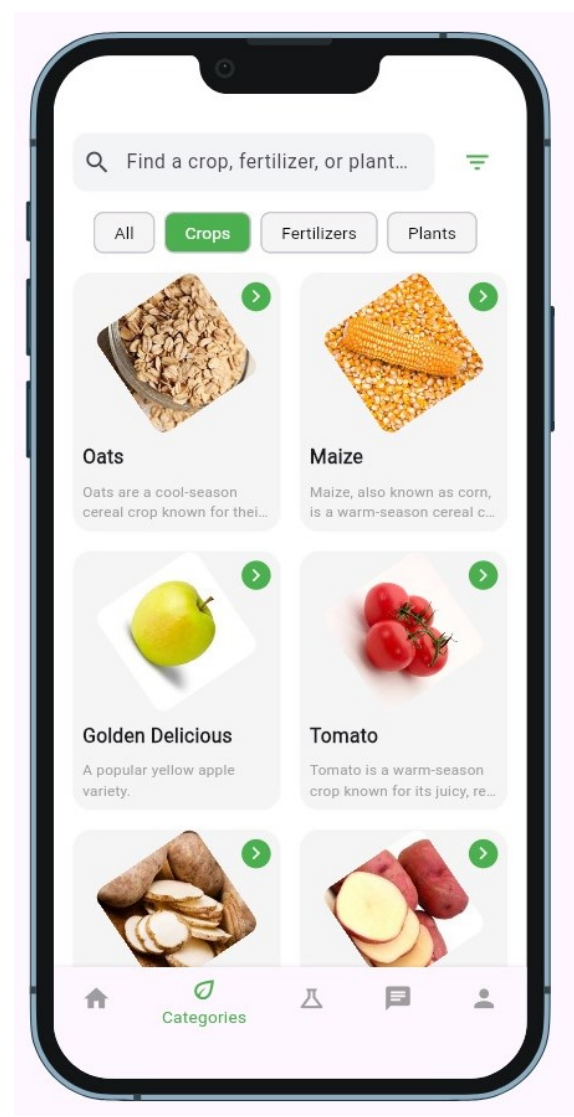


Figure 5.14 – Crops

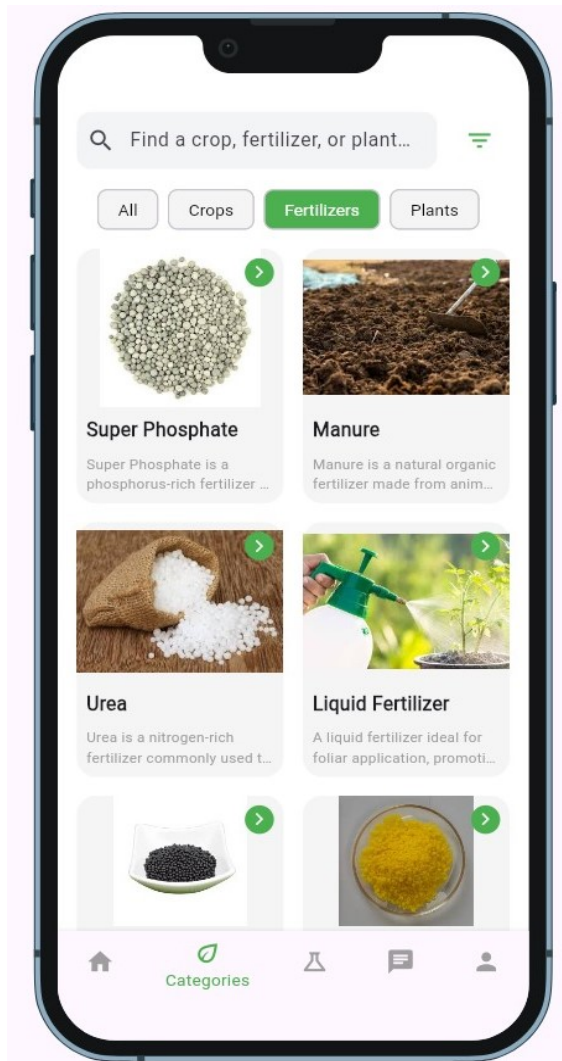


Figure 5.15 – Fertilizers

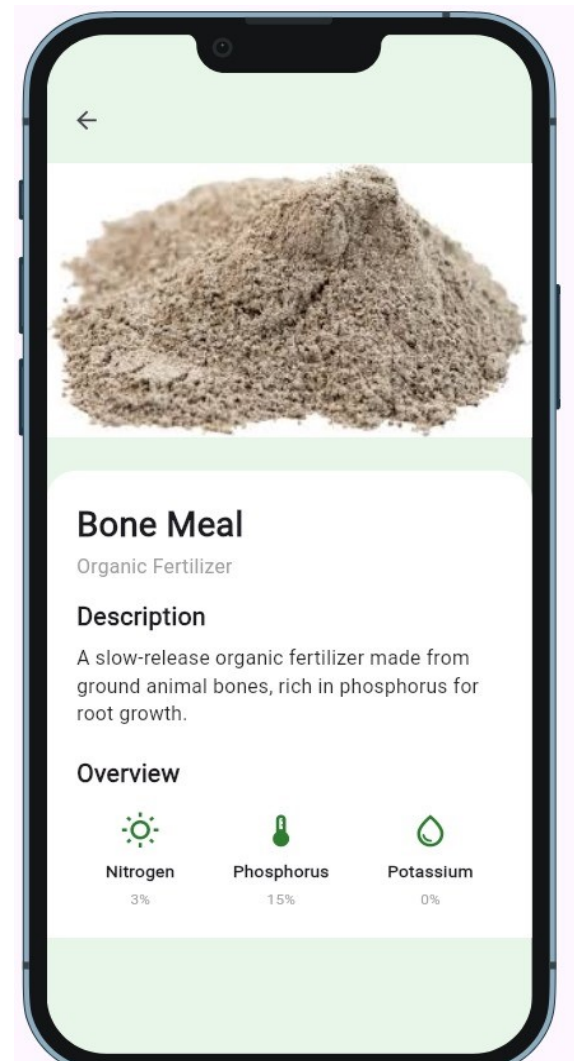


Figure 5.16 – Examble of fertilizers screens

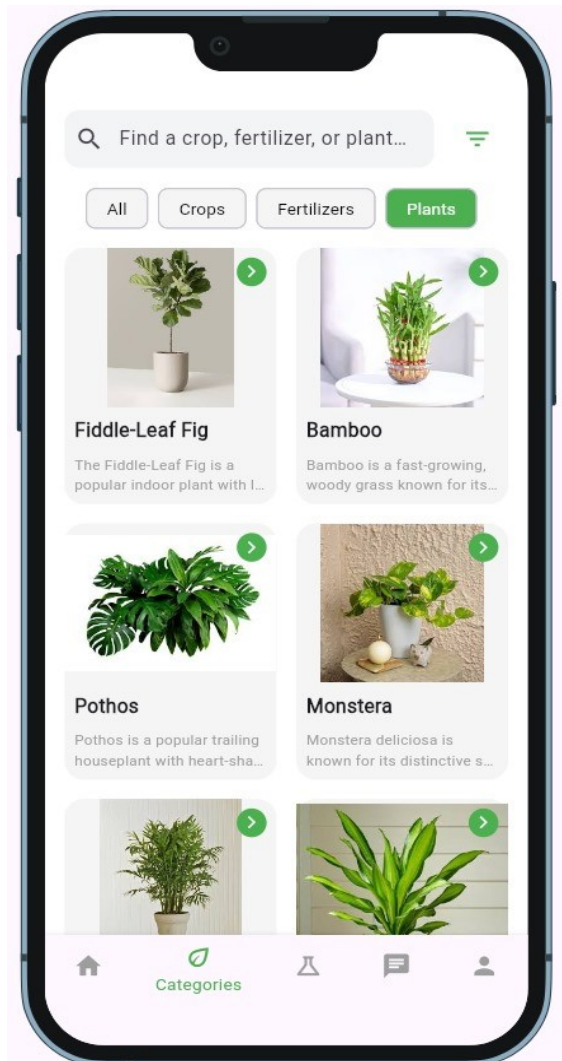


Figure 5.17 – Plants

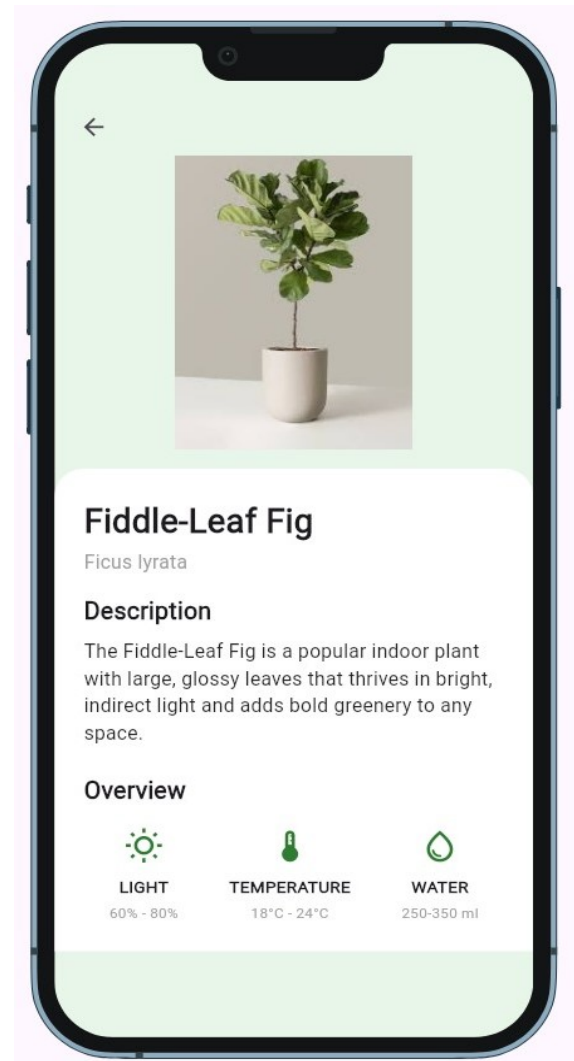
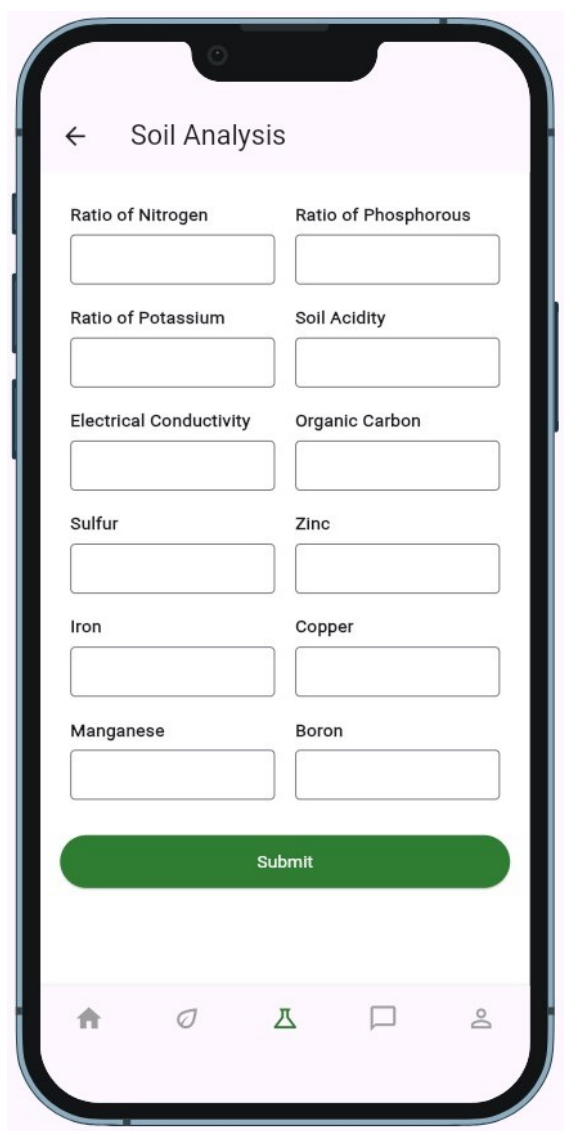


Figure 5.18 – Example of plants screens

5.7 Soil Analyzer screens



← Soil Analysis

Ratio of Nitrogen	Ratio of Phosphorous
<input type="text"/>	<input type="text"/>
Ratio of Potassium	Soil Acidity
<input type="text"/>	<input type="text"/>
Electrical Conductivity	Organic Carbon
<input type="text"/>	<input type="text"/>
Sulfur	Zinc
<input type="text"/>	<input type="text"/>
Iron	Copper
<input type="text"/>	<input type="text"/>
Manganese	Boron
<input type="text"/>	<input type="text"/>

Submit

Figure 5.19 – Soil analyzer model screen



Soil Analysis

Ratio of Nitrogen	Ratio of Phosphorous
300	100
Ratio of Potassium	Soil Acidity
800	7.0

Soil Analysis Report

تحليل التربة يوضح أنها متوسطة الخصوبة ⚠️
يمكن تحتاج تضيف أسمدة أو تعدل بعض
الخصائص في التربة علشان تتحسن
لو جاب تعرف إزاي تحسن أرضك أو تزرع
محاصيل مناسبة، تقدر تكلم الشات بوت
الزراعي، وهو هيرد على كل استفساراتك

Back

Submit

Figure 5.20 – Result screen

5.8 Chat screen



Figure 5.21 – Chat bot in Arabic

5.9 Filter screens

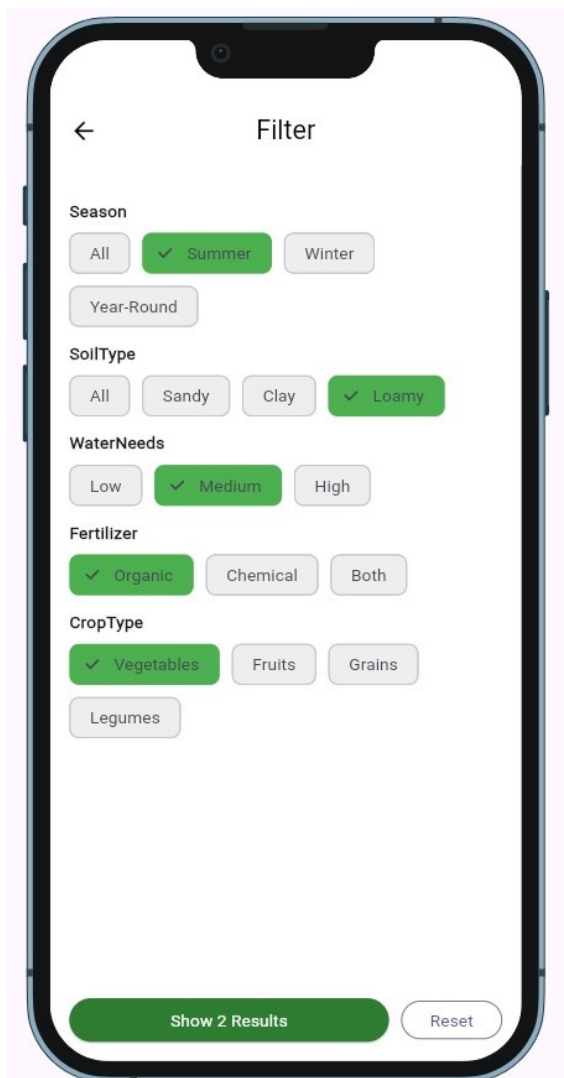


Figure 5.22 – Filter Options screen

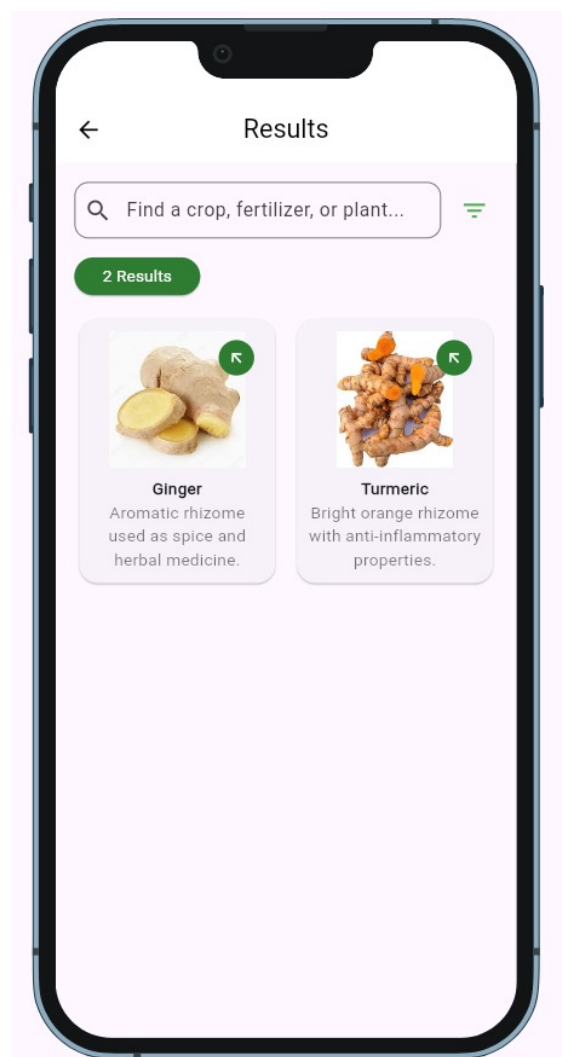


Figure 5.23 – Result of filtering screen

5.10 Notifications screen

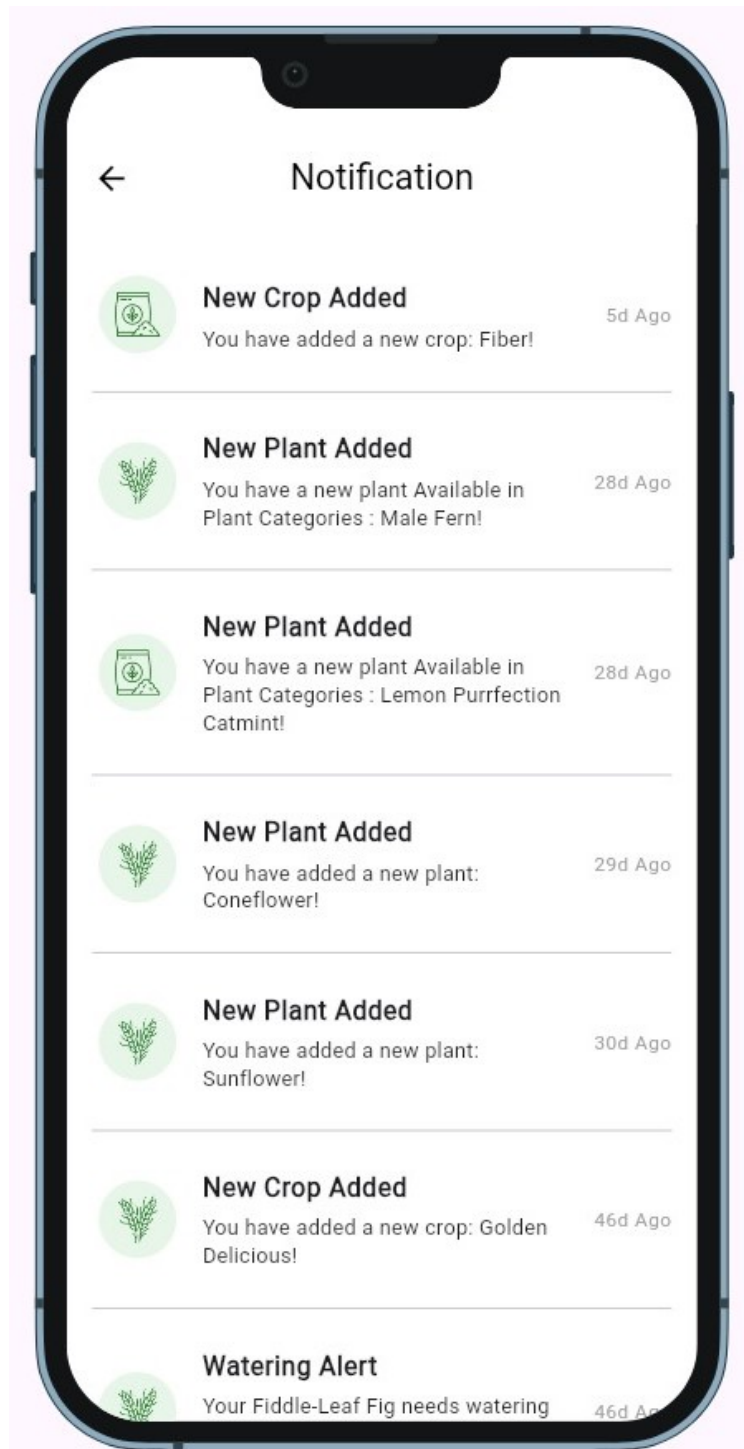


Figure 5.24 – Notifications

5.11 Profile screens

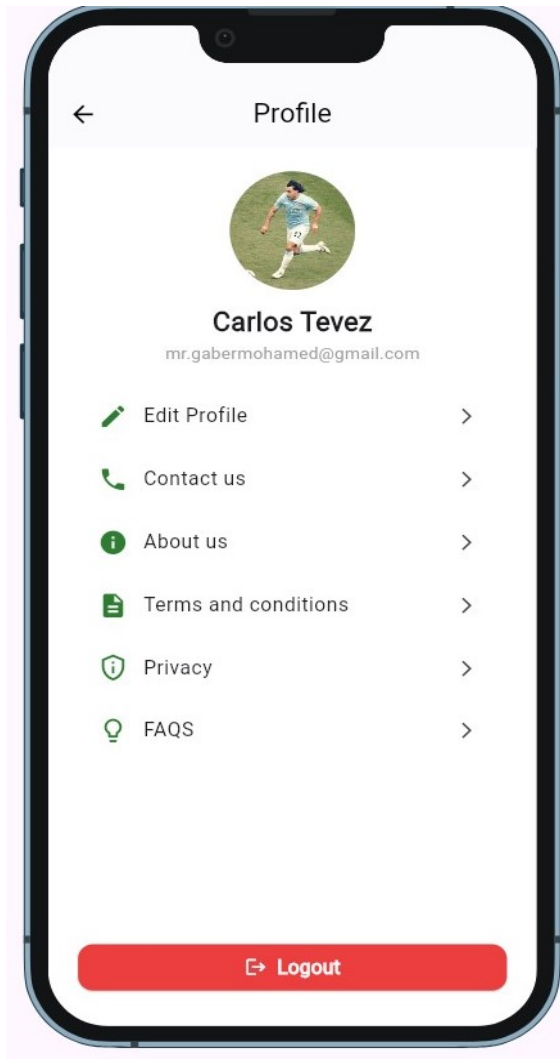


Figure 5.25 – Main profile screen

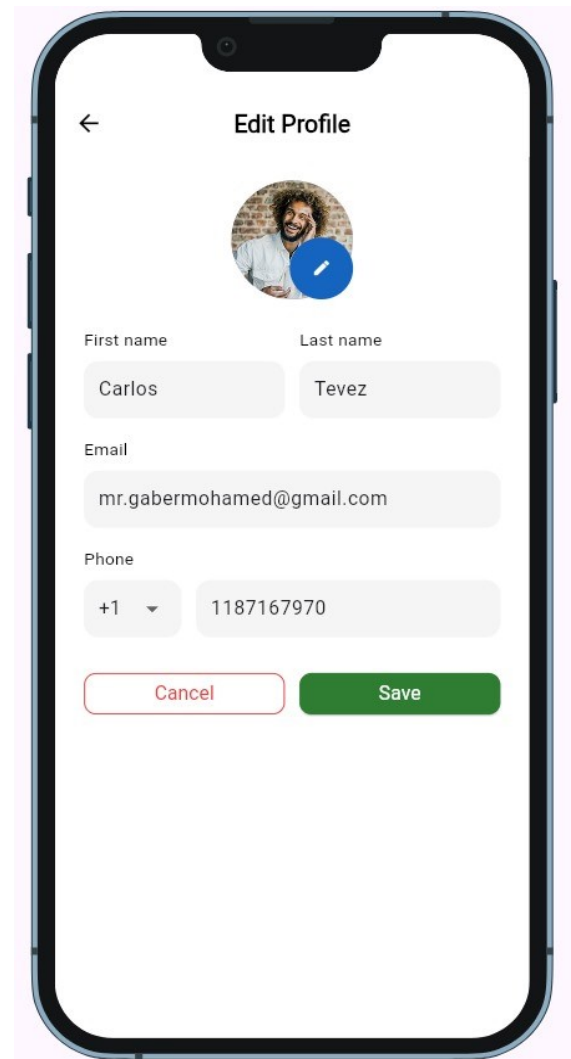


Figure 5.26 – Edit profile screen

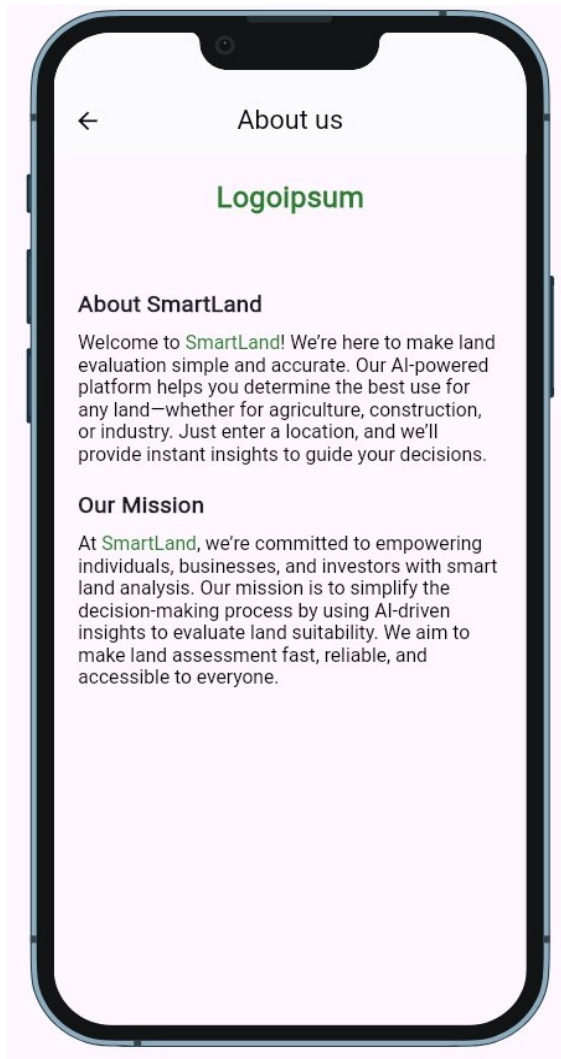


Figure 5.27 – About us screen

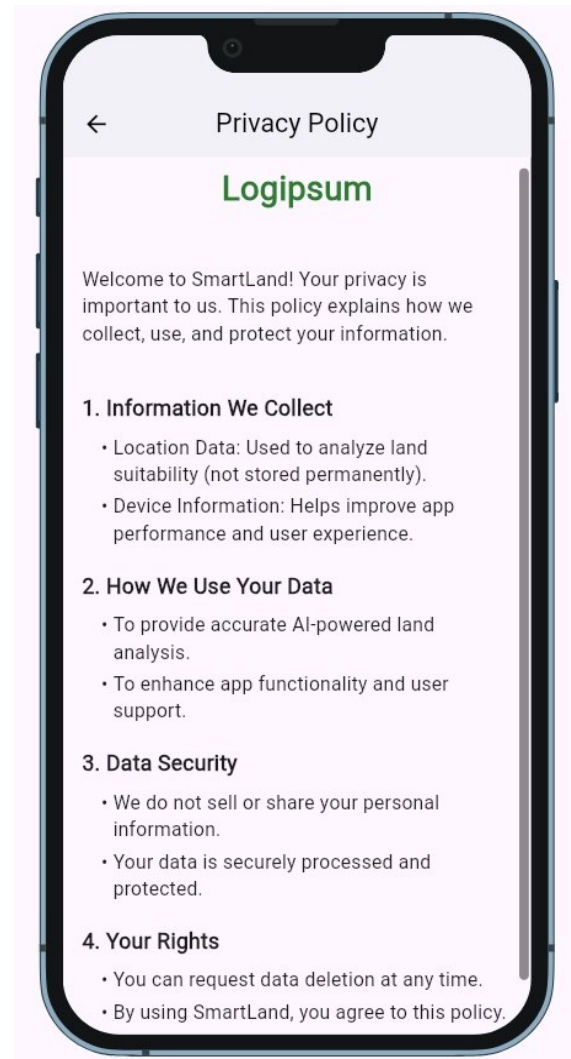
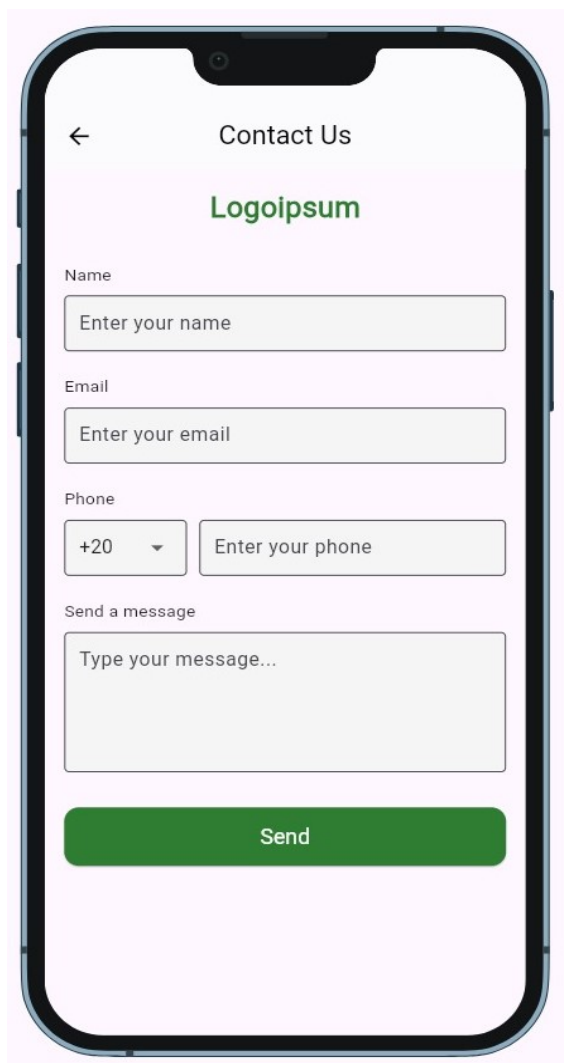
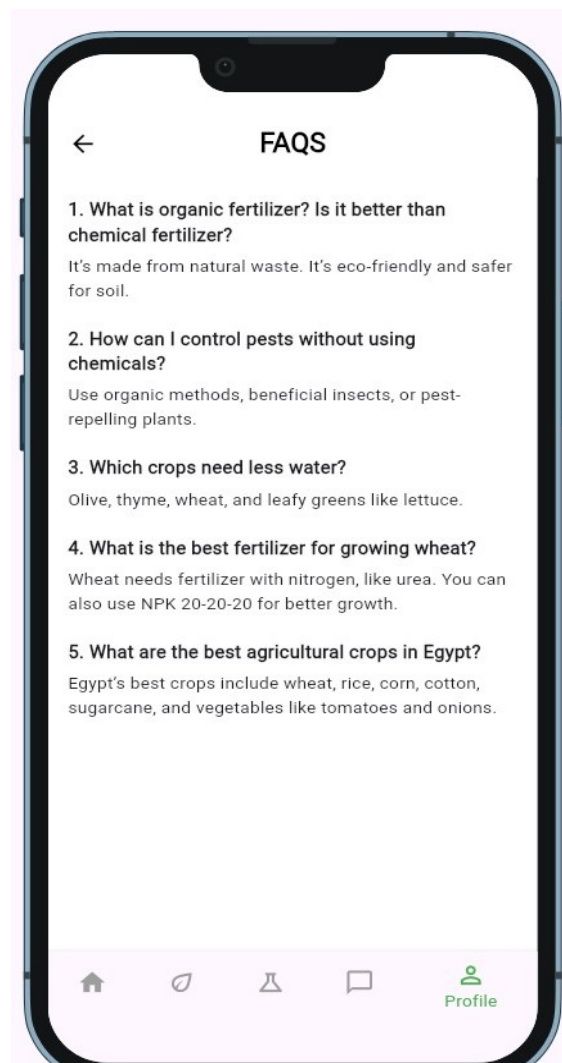


Figure 5.28 – Privacy policy screen



The 'Contact Us' screen features a light purple background. At the top, there is a back arrow and the title 'Contact Us'. Below the title is a green logo placeholder labeled 'Logoipsum'. The form includes three input fields: 'Name' with the placeholder 'Enter your name', 'Email' with the placeholder 'Enter your email', and 'Phone' with a dropdown menu showing '+20' and a placeholder 'Enter your phone'. Below these is a text area for 'Send a message' with the placeholder 'Type your message...'. At the bottom is a green 'Send' button.

Figure 5.29 – Contact us screen



The 'FAQS' screen has a light purple background. It features a back arrow and the title 'FAQS'. The content is organized into five numbered questions with their respective answers:

- 1. What is organic fertilizer? Is it better than chemical fertilizer?**
It's made from natural waste. It's eco-friendly and safer for soil.
- 2. How can I control pests without using chemicals?**
Use organic methods, beneficial insects, or pest-repelling plants.
- 3. Which crops need less water?**
Olive, thyme, wheat, and leafy greens like lettuce.
- 4. What is the best fertilizer for growing wheat?**
Wheat needs fertilizer with nitrogen, like urea. You can also use NPK 20-20-20 for better growth.
- 5. What are the best agricultural crops in Egypt?**
Egypt's best crops include wheat, rice, corn, cotton, sugarcane, and vegetables like tomatoes and onions.

At the bottom, there is a navigation bar with five icons: a home icon, a leaf icon, a triangle icon, a speech bubble icon, and a profile icon labeled 'Profile'.

Figure 5.30 – FAQS screen

Chapter 6

System Testing

6.1 Introduction

Software testing is an essential part of ensuring the quality and reliability of the Smart Land application. It verifies that the app's features such as soil fertility classification and chatbot responses work as intended and helps identify any defects before deployment.

Testing covered all key components, including the Flutter frontend, ASP.NET Core Web API, and the integrated AI models. Synthetic and real-world data were used to validate predictions and ensure accurate user interaction.

The testing phase ensured that the system behaves correctly under various scenarios and provides a reliable experience for users in agricultural decision-making.

6.2 Testing Process Goals

6.2.1 Validation Testing

The goal of validation testing in the *Smart Land* application is to ensure that the software meets user and system requirements. This includes verifying the accuracy of the soil fertility model, the relevance of chatbot responses, and the proper functioning of the mobile interface. A successful test confirms that key features perform as expected in real agricultural scenarios.

6.2.2 Defect Testing

Defect testing aims to identify bugs, errors, or unexpected behaviors within the application. This involves testing the app under edge cases or invalid input conditions to ensure robustness. For instance, the chatbot was tested for unusual agricultural questions or missing inputs, and the backend was evaluated for incorrect data handling or API failures. A successful defect test reveals a flaw in the system, allowing developers to correct it and improve system reliability.

6.3 Strategic Approach to Software Testing

6.3.1 Unit Testing

In the *Smart Land* project, unit testing was conducted on core modules such as the soil fertility prediction engine, chatbot logic, and backend APIs. This phase ensures that individual functions perform as expected. White-box techniques were applied to verify logic branches, input validation, and edge cases at the code level, enabling early detection of bugs during development.

6.3.2 Integration Testing

Integration testing focused on the interaction between key system components—such as the mobile frontend, API services, and machine learning models. The aim was to verify that these components communicate correctly, ensuring seamless data flow between the Flutter app, FastAPI backend, and the ML models hosted locally and via Hugging Face.

6.3.3 System Testing

System testing in the *Smart Land* application ensured that all integrated components—from the Flutter mobile app to the backend APIs and ML models—function cohesively as a complete system. This phase validated system-wide operations such as real-time prediction, chatbot responses, and database interactions. It also helped uncover hardware and platform compatibility issues across devices.

6.3.4 Acceptance Testing

Acceptance testing was carried out to evaluate the application in a real user environment. The goal was to confirm that the system meets the functional and non-functional requirements of agricultural users. Tests included checking the usability of the app interface, the accuracy of model predictions, and the responsiveness of the chatbot. This phase followed a black-box testing approach, focusing on outputs based on user inputs without knowledge of internal code logic.

6.4 Test Cases

6.4.1 Register Test Cases

The following test cases were designed to validate the user registration functionality in the *Smart Land* application:

6.4.2 Register Test Cases

Test ID	Description	Input Data	Expected Result	Result
Reg-001	Register with valid name, email, password, and matching confirm password	Name:nahla saeed Email:nahlasaeed3333@gmail.com Password:nahla3333 ConfirmPassword:nahla3333	OTP sent to email and registration successful	Pass
Reg-001b	Register with valid info but invalid OTP	Same as Reg-001 but with incorrect OTP	Registration failed due to invalid OTP	Pass
Reg-002	Valid name and password, invalid email format	Email:nahlasaeed.com	Registration failed due to invalid email	Pass
Reg-003	Attempt to register with email already in use	Email:nahlasaeed3333@gmail.com	Registration failed, email already in use	Pass
Reg-004	Valid name and email, weak password	Password: 123 ConfirmPassword: 123	Password must meet complexity requirements	Pass
Reg-005	Password and confirm password mismatch	Password:nahla3333 ConfirmPassword:nahla33	Registration failed, passwords do not match	Pass
Reg-006	Submit blank registration form	All fields left empty	Registration failed, required field warnings shown	Pass

Table 6.1 – Test Cases for Registration Process

6.4.3 Login Test Cases

Test case ID	Description	Test Data	Expected Result	Result
Log.001	Valid Email & valid password	Email: nahlasaeed3333@gmail.com Password: nahla3333	Login successfully	Pass as Expected
Log.002	Valid Email & invalid password	Email: nahlasaeed3333@gmail.com Password: nahla33	Unsuccessful Login Password is invalid	Pass as Expected
Log.003	Invalid Email & valid password	Email: nahlasaeed@gmail.com Password: nahla3333	Unsuccessful Login Email not valid	Pass as Expected
Log.004	Invalid Email & invalid password	Email: nahlasaeed33@gmail.com Password: 1234567	Unsuccessful login Email or password doesn't exist	Pass as Expected
Log.005	Email and password fields are blank and submit button is clicked	Email: Password:	Unsuccessful Login Fields are required	Pass as Expected
Log.006	Valid Email & password with Enter key	Email: nahlasaeed3333@gmail.com Password: nahla3333	Login successfully	Pass as Expected
Log.007	Password visible as asterisks or bullet signs	Email: Biolab123@gmail.com Password: ••••••••	Login successfully	Pass as Expected
Log.008	Valid Email with new password after reset	Email: nahlasaeed3333@gmail.com New Password: nahla1234	Login successfully	Pass as Expected

Table 6.2 – Login Test Cases

6.4.4 User Test Cases

Test ID	User Action	Input Data	Expected Result	Result
USR.001	Register new user	Valid name, email, password	Account created successfully	Pass
USR.002	Register with existing email	Existing email	Error: Email already registered	Pass
USR.003	Login with valid credentials	Registered email and password	Login successful	Pass
USR.004	Login with invalid credentials	Wrong email or password	Error message shown	Pass
USR.005	Use chatbot for question	Question: "When to plant wheat?"	Answer displayed from model	Pass
USR.006	Input soil data for analysis	Input NPK values, pH	Fertility result shown	Pass
USR.007	View crop information	Select "wheat" from crops page	Detailed crop info displayed	Pass
USR.008	View fertilizer types	Open fertilizers page	List of fertilizers displayed	Pass
USR.09	Logout from application	Click on logout button	Session ends and redirects to login	Pass

Table 6.3 – User Test Cases – Functional testing for user actions

6.4.5 user profile Test cases

Test ID	Profile Action	Input Data	Expected Result	Result
PROF.001	View profile info	Click "My Profile"	User info is displayed	Pass
PROF.002	Update full name	New name: Nahla Mohamed	Name updated successfully	Pass
PROF.003	Update email	Invalid email format	Error: Invalid email	Pass
PROF.004	Update email	Valid new email	Email updated successfully	Pass
PROF.005	Change password	Old pass + weak new pass	Error: Password must be strong	Pass
PROF.006	Change password	Old pass + valid new pass	Password changed successfully	Pass
PROF.007	Upload profile picture	Valid image file (JPG/PNG)	Image uploaded	Pass
PROF.008	Upload profile picture	Invalid format (e.g., .exe)	Error: Unsupported file type	Pass
PROF.009	Delete account	Confirm delete	Account deleted	Pass
PROF.010	Cancel account deletion	Click "Cancel" on confirm popup	Account remains active	Pass
PROF.011	Open Contact Us page	Click "Contact Us" button	Contact form is displayed	Pass
PROF.012	View Terms of Service	Click "Terms" link	Terms of Service page is displayed	Pass
PROF.013	View Privacy Policy	Click "Privacy Policy" link	Privacy Policy page is displayed	Pass

Table 6.4 – User Profile Test Cases

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In conclusion, this project successfully delivers a smart agricultural assistant application that harnesses the power of artificial intelligence to provide farmers with accessible, real-time support. By combining a fine-tuned LLaMA 3-based chatbot model with user-friendly mobile features, the application enhances agricultural knowledge, assists in decision-making, and addresses a critical need for localized, expert guidance in rural communities.

The chatbot is capable of handling bilingual agricultural inquiries in Arabic and English, including dialect support, and provides practical advice on a wide range of topics such as soil fertility, crop selection, irrigation, and fertilization. The integrated soil analysis tool further contributes to the user experience by allowing farmers to assess their soil's health and receive tailored recommendations.

7.2 Proposed Enhancements and Future Modifications

To ensure the system continues to grow and meet users' evolving needs, several enhancements are proposed for future development:

- **Voice Interaction Support:** Integrating voice input/output for users with low literacy or limited typing skills will improve accessibility and usability, especially in rural areas.

- **Expanded Dataset Coverage:** Enriching the training dataset with more agricultural scenarios, additional crops, regional farming techniques, and real user interactions will increase the accuracy and adaptability of the AI assistant.
- **Multilingual and Dialect Expansion:** Supporting more local dialects (e.g., Sa'idi, North African Arabic) and potentially other languages such as French or Swahili will widen the user base across different regions.
- **Offline Functionality:** Offering limited offline capabilities (e.g., cached responses or basic soil analysis) can improve usability in areas with unstable internet access.
- **Integration with Government and Agricultural Databases:** Linking the app to real-time agricultural alerts, subsidy programs, or weather data can make the assistant more context-aware and valuable to users.
- **Analytics Dashboard for Admins:** Providing backend access to user interaction data, common queries, and soil test patterns can help researchers and stakeholders understand agricultural challenges in real time.
- **Gamification and User Motivation:** Adding features like tips of the day, success stories, or user achievements can increase engagement and make learning more interactive.
- **Security and Data Privacy Enhancements:** Future iterations should strengthen user data protection, especially when collecting soil samples or profile data, and comply with international standards (such as GDPR).

7.3 Recommendations

- **User Feedback Collection** should continue to be prioritized in future versions to align development with real-world user needs.
- **Collaboration with Agricultural Experts** is recommended to continually refine the knowledge base and ensure information remains up-to-date.

- **Pilot Testing in Rural Communities** can provide actionable insights on usability, language preferences, and device compatibility.
- **Explore Funding and Partnership Opportunities** with agricultural ministries, NGOs, or research institutions to support the platform's growth and sustainability.

References

- [1] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2002.
- [2] K. E. Kendall and J. E. Kendall, *Systems Analysis and Design*, 10th ed. Pearson, 2019.
- [3] Figma Inc., “Figma: The collaborative interface design tool,” 2023, <https://www.figma.com>.
- [4] Google Developers, “Flutter - build apps for any screen,” 2024, <https://flutter.dev>.
- [5] Microsoft, “.net core documentation,” 2024, <https://learn.microsoft.com/en-us/dotnet/core/>.
- [6] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson, 2015.
- [7] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, “Llama: Open and efficient foundation language models,” 2023. [Online]. Available: <https://arxiv.org/abs/2302.13971>
- [8] H. Face, *Hugging Face: The AI Community Building the Future*, 2023, <https://huggingface.co>.
- [9] Google Research, “Google colaboratory,” 2024, <https://colab.research.google.com>.
- [10] Kaggle Inc., “Kaggle: Your home for data science,” 2024, <https://www.kaggle.com>.
- [11] A. Abid and other contributors, “Gradio: Build machine learning web apps in python,” 2023, <https://www.gradio.app>.

- [12] S. R. Tiangolo, “Fastapi: Fast web framework for building apis with python 3.7+,” 2023, <https://fastapi.tiangolo.com>.
- [13] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.