BUKIDNON STATE UNIVERSITY

COLLEGE OF TECHNOLOGIES

INFORMATION TECHNOLOGY DEPARTMENT

VertiGrow: Vertical Farm Monitoring App

Version: 1.0

May 9, 2025

Name	Roles	Contact Information
James Joshua Yosores	Project Manager	09364315158
Keith Einlou Pogoy	Lead Software Programmer	09462852550
Feby Angela Felices	Documentary	09944450774

TABLE OF CONTENTS

DOCUMENT OVERVIEW

Scope

The project covers the development of the Android version of VertiGrow, a vertical farming monitoring and control system designed to promote energy-efficient, sustainable agriculture in urban and semi-urban areas like Malaybalay City. This mobile version will enhance usability by enabling users to monitor farm conditions and irrigation directly from their Android devices.

Functional Scope:

- Real-time monitoring of sensor data (temperature, humidity, light, soil moisture, water level).
- Automatic irrigation control via fuzzy logic integration.
- Historical data visualization and basic analytics of water and energy consumption.

Technical Scope:

- Integration with IoT components (ESP32, sensors).
- Backend connectivity (Firebase API for real-time data).
- Data storage and syncing features using Firebase Realtime DB and Firestore.

Design Scope:

- Intuitive UI tailored for urban farmers and agricultural tech adopters.
- Use of simple language and visual indicators to make the system easy to use even for non-tech users.

Testing Scope:

- Unit testing of app functionalities
- Field testing with sensor hardware in an actual vertical farming setup

Out of Scope:

- iOS version development.
- Advanced AI-based predictions or machine learning integration
- Multi-language support

Audience:

The intended audience for the Android version of VertiGrow includes developers responsible for implementing and maintaining the app, particularly its integration with backend services and hardware components. Designers will focus on creating an intuitive interface tailored for users with limited technical experience. Testers will ensure the app's reliability across various environmental and sensor conditions. Stakeholders such as urban farmers, local agriculture officials, and educational institutions will assess the app's value in promoting sustainable farming. The City Agriculture Office is also considered a potential early adopter, offering real-world insights and feedback on its usability and impact.

PROJECT OVERVIEW

Executive Summary

Objective of the Project

This project aims to develop an Android-based vertical farming monitoring system that complements the existing IoT setup by providing real-time control and data visualization, integrating Fuzzy and LEACH algorithms to support sustainable and efficient farming practices. Specifically, it aims to;

Business Goals:

1. Support local and urban farmers with an accessible and affordable monitoring tool.
2. Enhance adoption of smart farming practices in areas with limited access to large-scale farming technologies.
3. Promote environmental sustainability through efficient water and energy use.
4. Empower academic institutions with hands-on tools for teaching smart agriculture.

Technical Goals:

1. Offer real-time access to sensor data (temperature, humidity, light, soil moisture, and water level) across all vertical farm layers.
2. Enable remote control of irrigation based on fuzzy logic or manual inputs.
3. Ensure seamless communication between the app and IoT devices via Firebase or RESTful APIs.
4. Develop a lightweight, responsive mobile interface optimized for low- to mid-range Android devices.

High-Level Features:

- Dashboard displaying sensor data in real-time using visual graphs and status indicators.
- Historical Data Logs for reviewing irrigation activity.
- Multi-layer Monitoring Support to reflect the unique structure of stacked vertical farms.
- Low-Bandwidth Mode to accommodate areas with weak or unstable internet connectivity.
- User Authentication and Role Management for secured access and personalized settings.
- System Diagnostics Panel to track the health of sensors and communication modules.

Problem Statement

Urban farmers face challenges in monitoring and managing vertical farms efficiently. Existing solutions are often complex or not suited for small-scale use, creating a need for a simple, accessible tool for real-time monitoring and irrigation control. Specifically, the project will focus on the following sub-questions

User needs:

1. How can the app provide real-time monitoring of key farm variables like temperature, humidity, and soil moisture?
2. How can the app offer simple, intuitive controls for irrigation systems without requiring technical expertise from users?
3. How can the app help optimize resource use, especially water and energy, in vertical farming?
4. How can the app ensure accessibility and affordability for small-scale urban farmers?

Market Analysis:

1. Who are the target users, and what farming management tools do they need?
2. How do existing farming apps compare to VertiGrow in terms of functionality and ease of use?
3. What features make VertiGrow stand out from competitors in small-scale vertical farming?
4. What barriers could prevent adoption of the app by urban farmers and institutions?

FUNCTIONAL SPECIFICATIONS

The VertiGrow Android app is designed to enhance vertical farming by providing real-time monitoring, automated irrigation control, and data visualization. The following outlines the key features, user interactions, and use cases to ensure efficient resource management and ease of use for urban farmers.

Feature List:

User

1. **Real-Time Sensor Monitoring**
    a. **Description**: The app will display real-time data from sensors monitoring temperature, humidity, soil moisture, and water level of the vertical farm.
    b. **User Interaction Flow**:

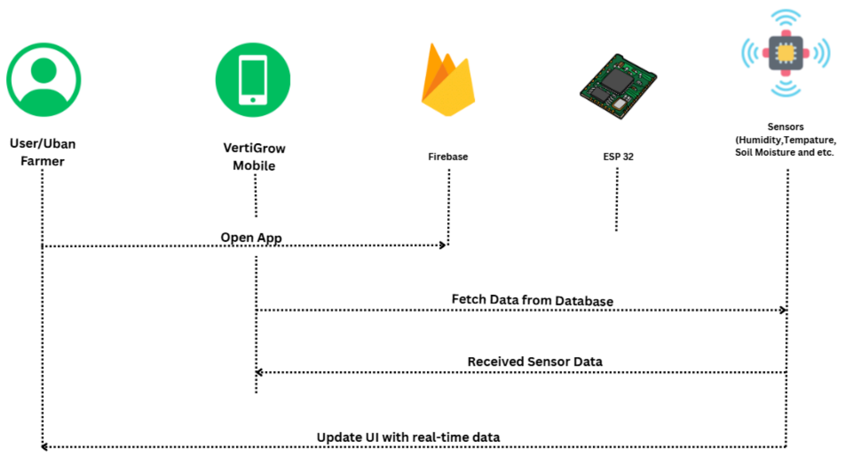


Figure 1. *Real-Time Sensor Monitoring*

Users will open the app, navigate to the farm view, and see an user-friendly UI with real-time sensor data.

c. **Use Case:**

- **Primary Use Case:** A user views the real-time data to assess farm conditions
- **Alternate Use Case**: The user views an indicator if any sensor data goes beyond the normal range, prompting them to take action.

2. **Automatic Irrigation Control**

a. **Description:** The app automatically controls irrigation using fuzzy logic, adjusting watering based on real-time sensor data like soil moisture and temperature.

b. **User Interaction Flow**:

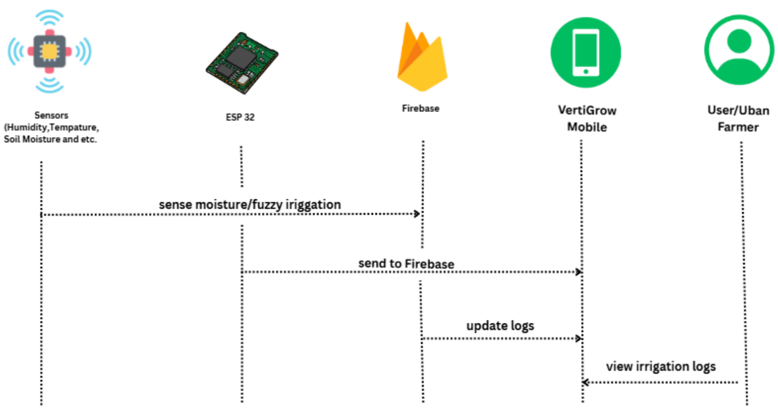


Figure 2. *Automatic Irrigation Control (Fuzzy Logic)*

Users can access and view the irrigation logs to monitor when and how the irrigation system was activated.

c. **Use Case:**

- **Primary Use Case:** The user views irrigation logs to see watering history and system activity.

3. **Irrigations Data Logs**

a. **Description:** The system stores past irrigation activity logs to support pattern analysis and decision

b. **User Interaction Flow:**

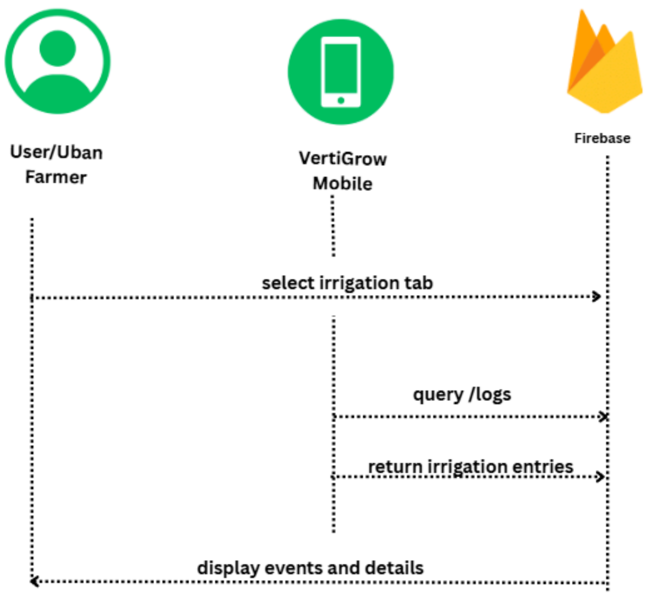


Figure 3. *Irrigation Data Logs*

Users navigate to the history tab and select a date range to view past environmental data and system actions.

c. **Use Case**

- **Primary Use Case:** A user checks irrigation logs to see the list of the irrigation events that happened in the past, helping them understand watering frequency and patterns

Admin

1. **User Account Management**

   a. **Description:** Admins can create user accounts by sending email invites and generating default passwords.

   b. **User Interaction Flow:**

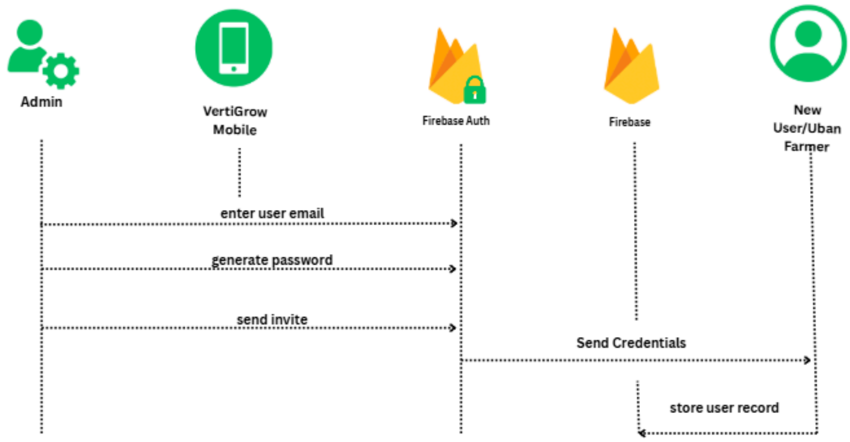


Figure 4. *User Account Management (Admin Only)*

The admin goes to "User Management," enters the user's email, generates a password, and sends an invite via email.

   c. **Use Case:**

   - **Primary Use Case:** The admin adds a new urban farmer to the system securely.

2. Vertical Farm Rack Monitoring

   a. **Description:** Admins can view every user's farm setup, including individual rack and sensor statuses.

   b. **Interaction Flow:**

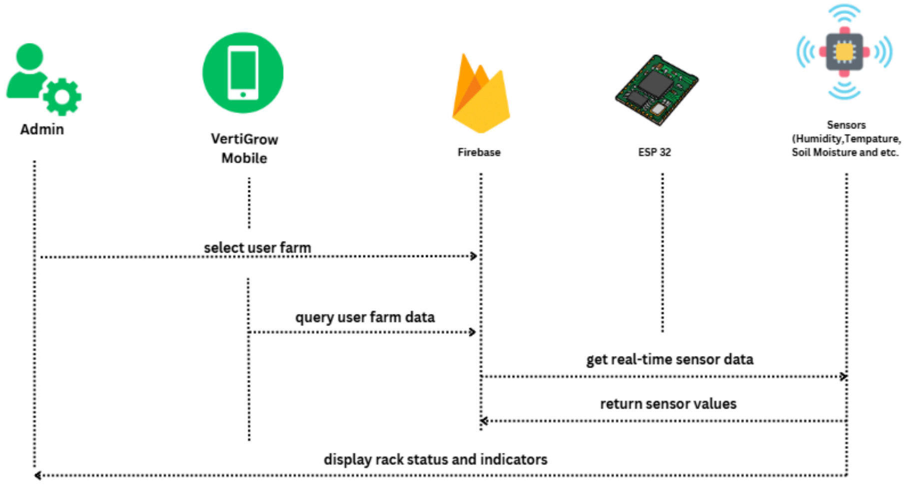


Figure 5. *Vertical Farm Rack Monitoring (Admin side)*

The admin selects a user in the drop down and clicks the view farm to see their racks and real-time sensor data.

   c. **Use Case:**

      - **Primary Use Case:** The admin audits multiple farms to ensure sensor activity and health across users.

3. **Analytics Overview**

   a. **Description:** Displays a full dashboard of all users' cumulative water and energy usage for insights and reporting.

b. **Interaction Flow:**

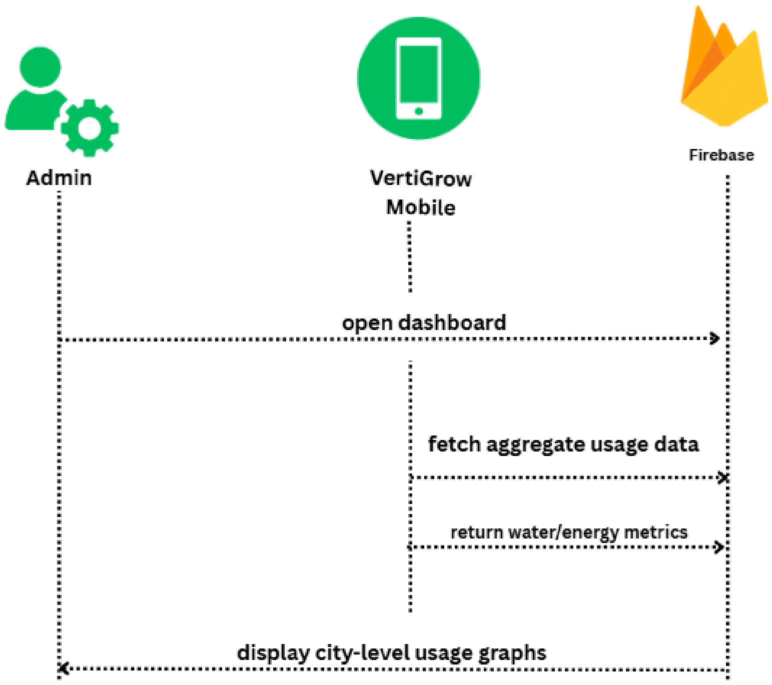


Figure 6. *Analytics Overview on Admin Dashboard*

The admin opens a "**Dashboar**d" to view charts of water and energy usage across all farms**.**

c. **Use Case:**

- **Primary Use Case:** The admin analyzes water and energy usage trends at a city or community level (by week, one and in three months).

4. **Activity Logs**

a. **Description:** shows a log of all actions performed within the app (user creation, user deletion, farm deletion, farm add and etc.)

b. Interaction Flow:

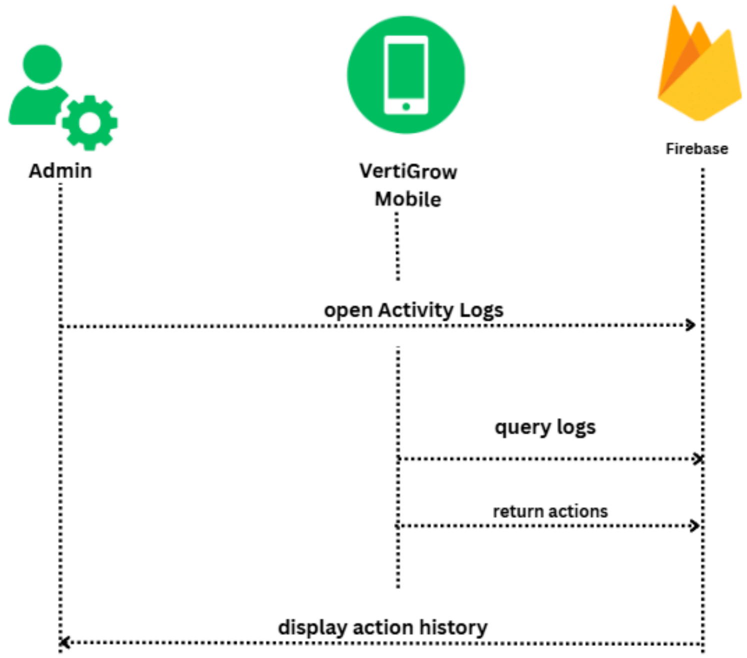


Figure 7. *Activity Logs*

The admin checks "Activity Logs" to see system actions like creations, deletions etc.

c. Use Case:

- **Primary Use Case:** The admin investigates specific events or reviews system behavior over time.

User Stories & Requirements

User Story:

As an urban farmer, I want to view real-time sensor data (temperature, humidity, soil moisture, and water level) in the app so I can monitor the condition of my vertical farm and respond immediately if necessary.

Acceptance Criteria

- The sensor data must be updated automatically without requiring manual refresh.
- The values should be clearly labeled and visually distinct (e.g., icons, colors, or units).
- The system should display a fault indicator (eg. green for no fault and red if there's a fault)

Non-functional Requirements

- **Performance:** Sensor data must be displayed with a maximum latency of 2 seconds.
- **Scalability:** The app should support real-time monitoring for up to 3 layers simultaneously without performance degradation.
- **Usability:** Display sensor data clearly with icons or graphs for easy mobile viewing.

TECHNICAL SPECIFICATIONS

Architecture

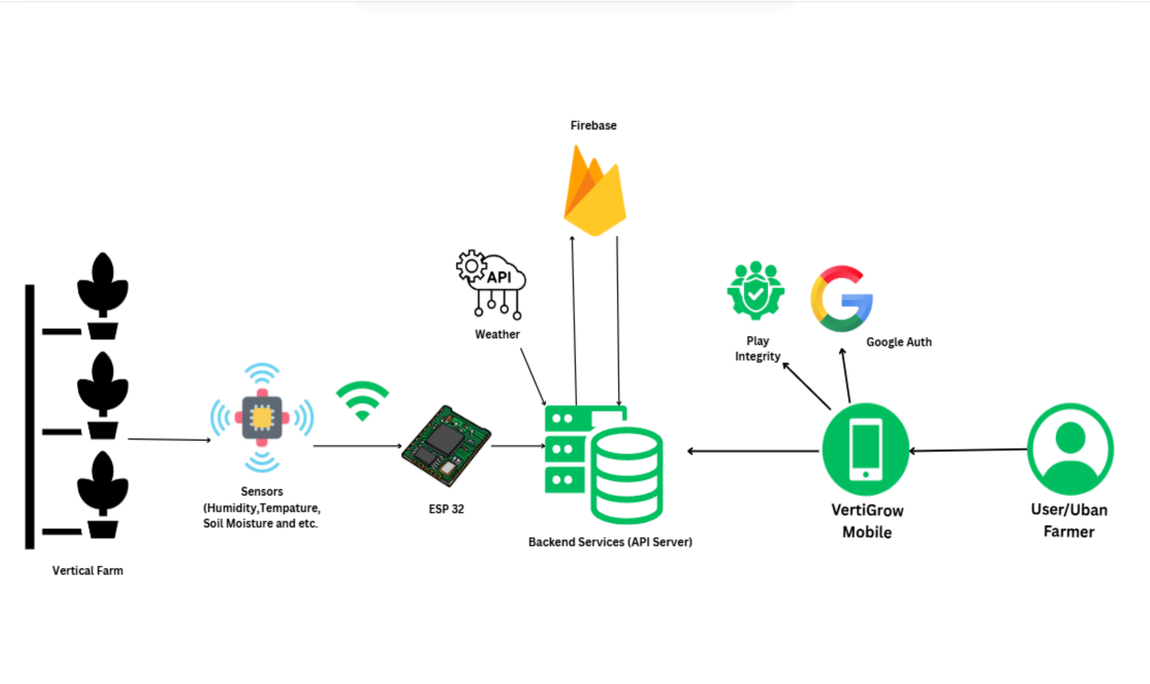


Figure 8. *VertiGrow Architecture*

This architecture connects a vertical farming setup with an Android mobile app through an IoT and cloud-based system. Sensors in the farm measure environmental data (e.g., temperature, humidity, soil moisture) and send it via an ESP32 microcontroller to backend services. The backend, integrated with weather APIs and Firebase, processes the data and enables real-time syncing with the VertiGrow mobile app. The app allows urban farmers to monitor conditions, control irrigation manually or through fuzzy logic, and authenticate securely via Google. The system also supports energy-efficient sensor communication using the LEACH algorithm, promoting sustainable and smart farming practices.

MVVM (Model-View-ViewModel) as Design Pattern

The MVVM (Model–View–ViewModel) design pattern is a way to organize code so that an app is easier to build, test, and maintain. It works by separating the user interface (View) from the business logic and data (Model), with the ViewModel acting as a bridge between the two. The Model is in charge of managing the data—like sensor readings, weather info, or user profiles—and handles communication with sources like Firebase or APIs. The View is what the user sees and interacts with, such as buttons, charts, or sensor readings on the screen. It doesn't handle any logic directly. Instead, it listens to the ViewModel, which holds and prepares the data needed for the UI. The ViewModel takes care of processing that data—like converting sensor values into readable information—and updates the View whenever the data changes. It also handles user actions by triggering the right functions in the Model. In an app like VertiGrow, MVVM helps keep things organized: the sensors send data to Firebase (Model), the ViewModel listens for changes and decides how that data should look, and the View updates automatically. This pattern keeps the code cleaner and helps developers focus on each part of the app without getting overwhelmed.

<u>Platform-Specific Considerations</u>

- Android Design Guidelines (UX/UI)
    - Follow Material Design for consistent look and feel.
    - Use bottom navigation or drawers for clear app structure.
    - Ensure 48x48 dp touch targets and readable fonts.

  - Support dark mode, responsive layouts, and smooth animations.
  - Provide clear loading indicators and adaptive icons.
- Hardware & OS Compatibility
  - Min SDK: Android 7.0 (API 24)
  - Target SDK: Latest version (e.g., Android 14, API 34)
  - Supports ARMv7 & ARM64 devices
  - Requires internet access and Google Services for Firebase, Google Auth, and Play Integrity
  - Optimized for phones and tablets with various screen sizes.

Data Management

The VertiGrow mobile app is designed to ensure smooth communication between the user interface, device storage, cloud databases, and external services. The flow of data begins with the user interacting with the VertiGrow app. Any input or action—such as logging in, monitoring sensor data, or adjusting farm settings—is first handled by the mobile interface. Sensor data collected from ESP32 microcontrollers is sent directly to Firebase in real time, where it can be accessed by the mobile app or viewed through dashboards. Firebase also handles authentication via Google Sign-In and maintains integration with Play Integrity to ensure secure sessions. External APIs, such as a weather API, are also called by the app to fetch contextual environmental data for decision-making.

The app uses both local and cloud-based databases. Locally, tables like UserPrefs, SensorCache, and FarmLayout are used to store user preferences, recent sensor readings, and configurable settings for each vertical layer. On the server side (Firebase), structured collections such as /users, /farms, and /sensors store detailed information about user accounts, farm configurations, and real-time sensor data. Each document includes fields such as temperature, humidity, soil moisture, timestamps, and ownership metadata.

API Endpoints

1. Firebase Authentication Endpoints
   - Base URL: Firebase Authentication Service
   - Endpoints:
     - Email/Password Authentication
     - Google Sign-In Authentication
     - Token-based authentication with Play Integrity verification
2. **Firebase Realtime Database Endpoints**
   - Base URL: https://vertigrow-ae776-default-rtdb.asia-southeast1.firebasedatabase.app
   - Collections:
     - /sensors - Real-time sensor data

- Structure:
  - JSON:


```json
{
  "-ONxNaj9U2eIsp2ij2co": {
    "batch_id": "481985",
    "battery_level": 100,
    "energy": 0.5,
    "farm": 1,
    "grow_lights_status": "off",
    "humidity": {
      "fault": "none",
      "humidity_value": 51.12
    },
    "layer_1": {
      "fault": "none",
      "moisture_value": 1756
    },
    "layer_2": {
      "fault": "none",
      "moisture_value": 2345
    },
    "layer_3": {
      "fault": "none",
      "moisture_value": 2475
    },
```

```json
    "light": {
      "fault": "none",
      "light_value": 867.5
    },
    "ph_level": {
      "fault": "none",
      "ph_value": 6.842
    },
    "temperature": {
      "fault": "none",
      "temperature_value": 29.07
    },
    "timestamp": 1745174400000,
    "water_level": {
      "fault": "none",
      "status": "low"
    },
    "water_pump_status": {
      "fault": "none",
      "pump_runtime": 25,
      "status": "off"
    }
```


3. Firebase Firestore Endpoints

- Collections:
  - /farms - Farm management data
  - Fields:
    1. userId: string
    2. farm: string
    3. isOnline: boolean
  - /users - User management
  - /logs - Activity logging
  - Fields:
    1. userId: string
    2. farmId: string (optional)

3. description: string
4. timestamp: number
5. type: string

4. OpenWeatherMap API

- Base URL: https://api.openweathermap.org/data/2.5/weather
- Endpoint: /weather
- Parameters:
    - lat: latitude
    - lon: longitude
    - units: metric
    - appid: API key
- Response Structure:

```
{
  "main": {
    "temp": number,
    "humidity": number
  },
  "wind": {
    "speed": number
  },
  "weather": [{
    "id": number,
    "description": string
  }],
  "name": string
}
```

5. Google Play Integrity API

- Used for security verification
- Endpoint: Play Integrity Service
- Response: Integrity token for verification

6. Firebase App Check

- Used for additional security
- Provider: Play Integrity App Check Provider

Sample API Request:

1. Weather Data Request

```java
public void getWeatherByLocation(double latitude, double longitude, WeatherCallbac
    String url = BASE_URL + "?lat=" + latitude + "&lon=" + longitude +
            "&units=metric&appid=" + API_KEY;
```

2. Farm Data Query:

```java
// Query farms where userId matches the current user
farmsCollection.whereEqualTo( field: "userId", userId)
        .addSnapshotListener((value, error) -> {
            if (error != null) {
                Log.e(TAG,  msg: "Error loading farms: " + error.getMessage());
                Toast.makeText(getContext(),  text: "Error loading farms: " + error.getMessage(),
                        Toast.LENGTH_SHORT).show();
                return;
            }

            if (value == null) {
```

3. Sensor Data Query

```java
// Query for any sensor data for this farm
sensorsRef.orderByChild( path: "farm").equalTo(farm.getFarm())
        .addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                Log.d(TAG,  msg: "Received sensor data snapshot for farm " + farm.getFarm() + ", exists: " + data
                Log.d(TAG,  msg: "Number of children: " + dataSnapshot.getChildrenCount());

                if (dataSnapshot.exists() && dataSnapshot.getChildrenCount() > 0) {
                    Log.d(TAG,  msg: "Sensor data exists for farm " + farm.getFarm() + ", setting online=true");
                    // If any sensor data exists for this farm, it's online
                    farm.setOnline(true);
                } else {
                    Log.d(TAG,  msg: "No sensor data exists for farm " + farm.getFarm() + ", setting online=false
                    farm.setOnline(false);
                }
```

4. Log Entry Creation

```java
LogEntry logEntry = new LogEntry(
         id: null,
        currentUserId,
         farmId: null,
        description,
        System.currentTimeMillis(),
        action
);

db.collection( collectionPath: "logs")  CollectionReference
        .add(logEntry)  Task<DocumentReference>
        .addOnSuccessListener(documentReference -> {
            Log.d(TAG,  msg: "Log entry added with ID: " + documentReference.getId());
        })
        .addOnFailureListener(e -> {
            Log.e(TAG,  msg: "Error adding log entry: " + e.getMessage());
        });
```

Security & Privacy

1. Authentication & Authorization:
    - Authentication Methods:
        - Email/Password authentication using Firebase Auth
        - Google Sign-In integration using OAuth 2.0
        - Role-based access control (admin/user roles)
        - Secure password generation with PasswordGenerator class
    - Security Features:
        - Play Integrity API integration for app security verification
        - Firebase App Check for additional security layer
        - Secure password requirements (minimum 8 characters, mix of character types)
        - First-time password change requirement for new users
2. Data Security
    - In-Transit Security:
        - HTTPS communication with Firebase services
        - Secure API calls using OkHttp
        - Encrypted communication channels
    - At-Rest Security:
        - Firebase Firestore for secure data storage
        - Secure password storage using Firebase Auth
        - User data stored with proper access controls

3. Privacy & Data Protection:
    - User Data Management:
        - Structured user data storage in Firestore
        - Clear user roles and permissions
        - Secure credential distribution via email
        - Proper error handling and logging
    - Security UI/UX:
        - Security verification dialogs
        - Clear error messages for security failures
        - Visual feedback for security checks
        - Proper session management
4. Additional Security Features:
    - Play Integrity API for app verification
    - Firebase App Check for additional security
    - Secure password generation with multiple character sets
    - Proper error handling and user feedback
    - Role-based access control implementation

<u>Third-Party Integration</u>

1. **Authentication & Security SDKs**
    - **Google Sign-In SDK**
    - Version: com.google.android.gms:play-services-auth:20.5.0
    - Integration Details:

        - Used in MainActivity.java for Google authentication
        - Configured with OAuth client ID from google-services.json
        - Handles user sign-in flow with proper error handling
        - Integrates with Firebase Authentication
    - **Firebase Authentication**
        - Integration Details:
            1. Used for both email/password and Google authentication
            2. Implemented in MainActivity.java and LinkAccountActivity.java
            3. Handles user session management
            4. Provides secure token-based authentication
    - **Play Integrity API**
        - Version: com.google.android.play:integrity:1.1.0
        - Integration Details:
            1. Used for app security verification
            2. Implemented in MainActivity.java
            3. Verifies app integrity before login
            4. Uses cloud project number: 42961822919
            5. Provides security checks for both email and Google sign-in
    - **Firebase App Check**
        - Version: com.google.firebase:firebase-appcheck:17.1.1

        - Integration Details:
            1. Initialized in VertiGrowApplication.java
            2. Uses Play Integrity provider
            3. Provides additional security layer
            4. Prevents unauthorized access to Firebase resources

2. **Data & Storage SDKs**
    - **Firebase Firestore**
    - Integration Details:
        - Used for database operations
        - Stores user data and application information
        - Implemented across multiple activities
        - Provides real-time data synchronization
    - **OpenWeatherMap API**
    - Integration Details:
        - Implemented in WeatherService.java
        - Uses OkHttp for API calls
        - API Key: bd5e378503939ddaee76f12ad7a97608
        - Provides weather data based on location
        - Includes error handling and response parsing
3. **UI & Visualization Libraries**
    - **MPAndroidChart**
    - Version: com.github.PhilJay:MPAndroidChart:v3.1.0
        - Integration Details:

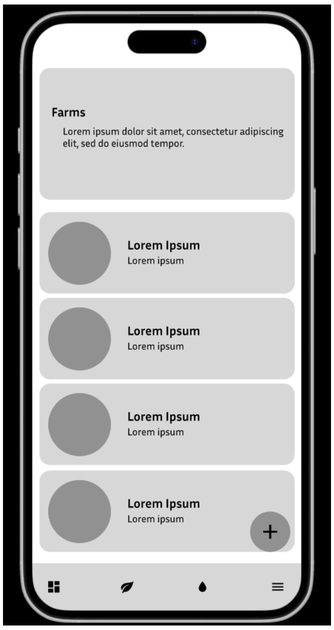
        - Used for data visualization
        - Implemented in AdminDashboardFragment.java
        - Handles water consumption data visualization
        - Provides interactive charts and graphs

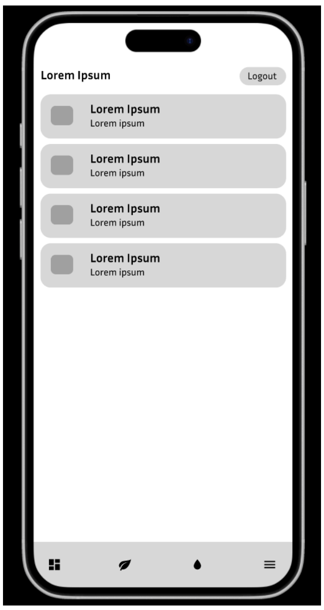
4. **Network & Utility Libraries**
    - **OkHttp**
    - Version: com.squareup.okhttp3:okhttp:4.11.0
    - Integration Details:
        - Used for network requests
        - Implemented in WeatherService.java
        - Handles API calls to OpenWeatherMap
        - Includes timeout configurations:
        - Connect timeout: 10 seconds
        - Read timeout: 30 seconds
    - **JSON Library**
    - Version: org.json:json:20231013
    - Integration Details:
        - Used for JSON parsing
        - Implemented in WeatherService.java
        - Handles API response parsing
        - Provides data structure manipulation

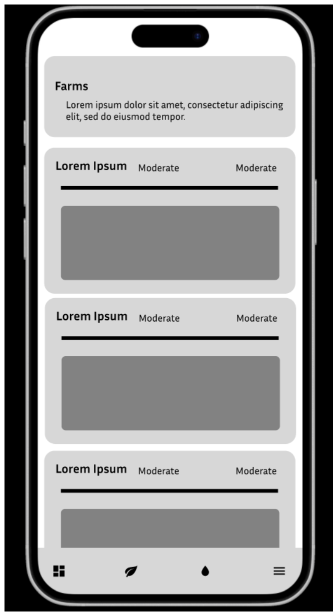
UI/UX DESIGN SPECIFICATION

Wireframes & Mockups

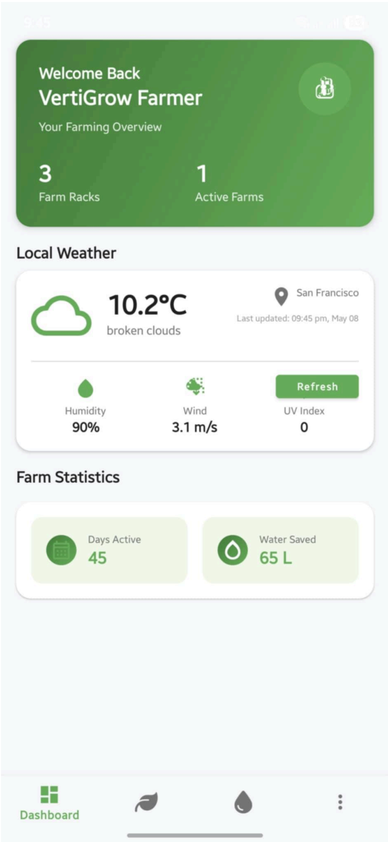
Screens Overview

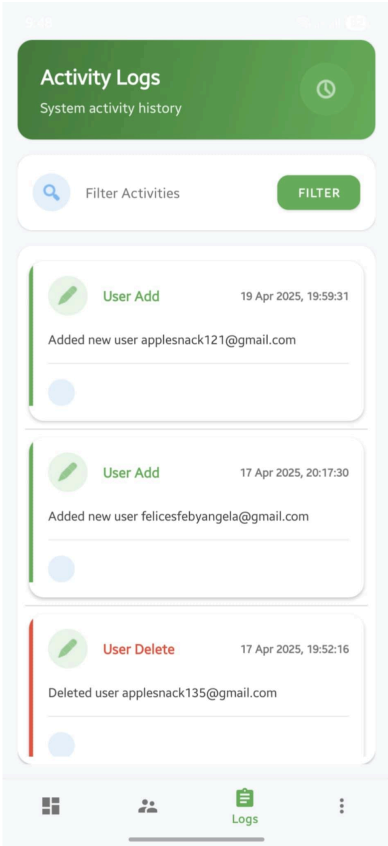


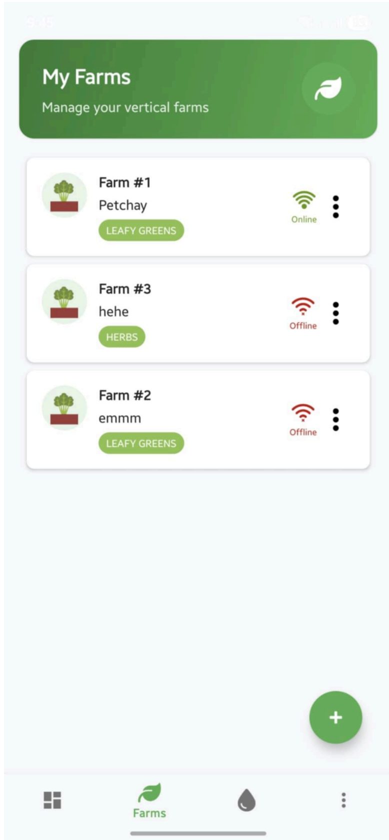


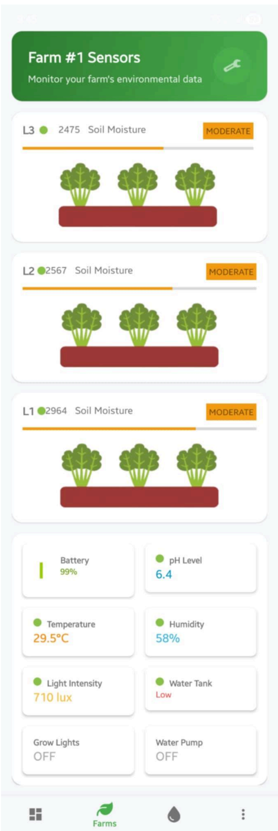


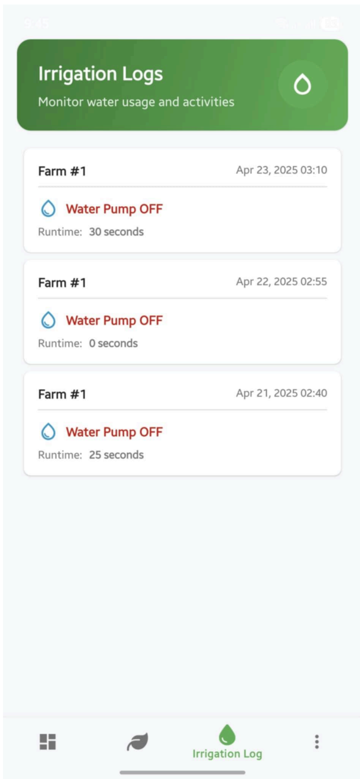
High-Fidelity Designs (Examples)

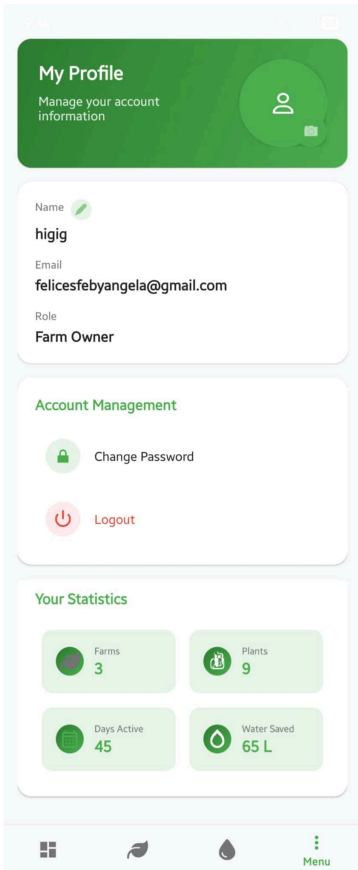


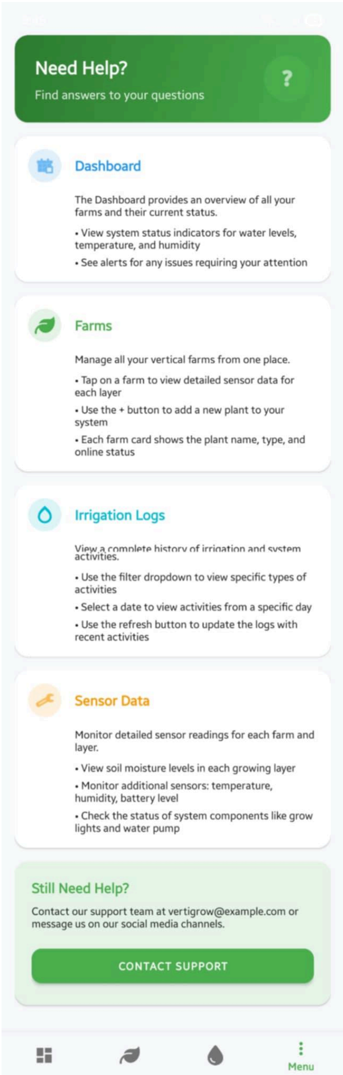


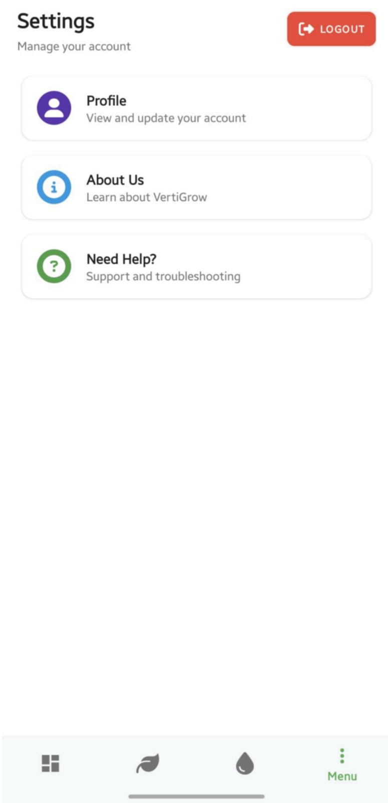


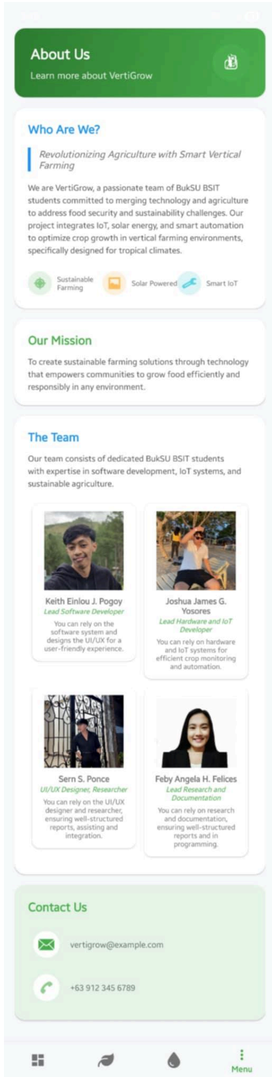







Need Help?
Find answers to your questions
Dashboard
The Dashboard provides an overview of all your farms and their current status.
• View system status indicators for water levels, temperature, and humidity
• See alerts for any issues requiring your attention
Farms
Manage all your vertical farms from one place.
• Tap on a farm to view detailed sensor data for each layer
• Use the + button to add a new plant to your system
• Each farm card shows the plant name, type, and online status
Irrigation Logs
View a complete history of irrigation and system activities.
• Use the filter dropdown to view specific types of activities
• Select a date to view activities from a specific day
• Use the refresh button to update the logs with recent activities
Sensor Data
Monitor detailed sensor readings for each farm and layer.
• View soil moisture levels in each growing layer
• Monitor additional sensors: temperature, humidity, battery level
• Check the status of system components like grow lights and water pump
Still Need Help?
Contact our support team at vertigrow@example.com or message us on our social media channels.
CONTACT SUPPORT
Menu


Settings
Manage your account
LOGOUT
Profile
View and update your account
About Us
Learn about VertiGrow
Need Help?
Support and troubleshooting
Menu


About Us
Learn more about VertiGrow
Who Are We?
Revolutionizing Agriculture with Smart Vertical Farming
We are VertiGrow, a passionate team of BukSU BSIT students committed to merging technology and agriculture to address food security and sustainability challenges. Our project integrates IoT, solar energy, and smart automation to optimize crop growth in vertical farming environments, specifically designed for tropical climates.
Sustainable Farming
Solar Powered
Smart IoT
Our Mission
To create sustainable farming solutions through technology that empowers communities to grow food efficiently and responsibly in any environment.
The Team
Our team consists of dedicated BukSU BSIT students with expertise in software development, IoT systems, and sustainable agriculture.
Keith Einlou J. Pogoy
Lead Software Developer
You can rely on the software system and designs the UI/UX for a user-friendly experience.
Joshua James G. Yosores
Lead Hardware and IoT Developer
You can rely on hardware and IoT systems for efficient crop monitoring and automation.
Sern S. Ponce
UI/UX Designer, Researcher
You can rely on the UI/UX designer and researcher, ensuring well-structured reports, assisting and integration.
Feby Angela H. Felices
Lead Research and Documentation
You can rely on research and documentation, ensuring well-structured reports and in programming.
Contact Us
vertigrow@example.com
+63 912 345 6789
Menu

Navigation & Flow

User Flow Diagrams

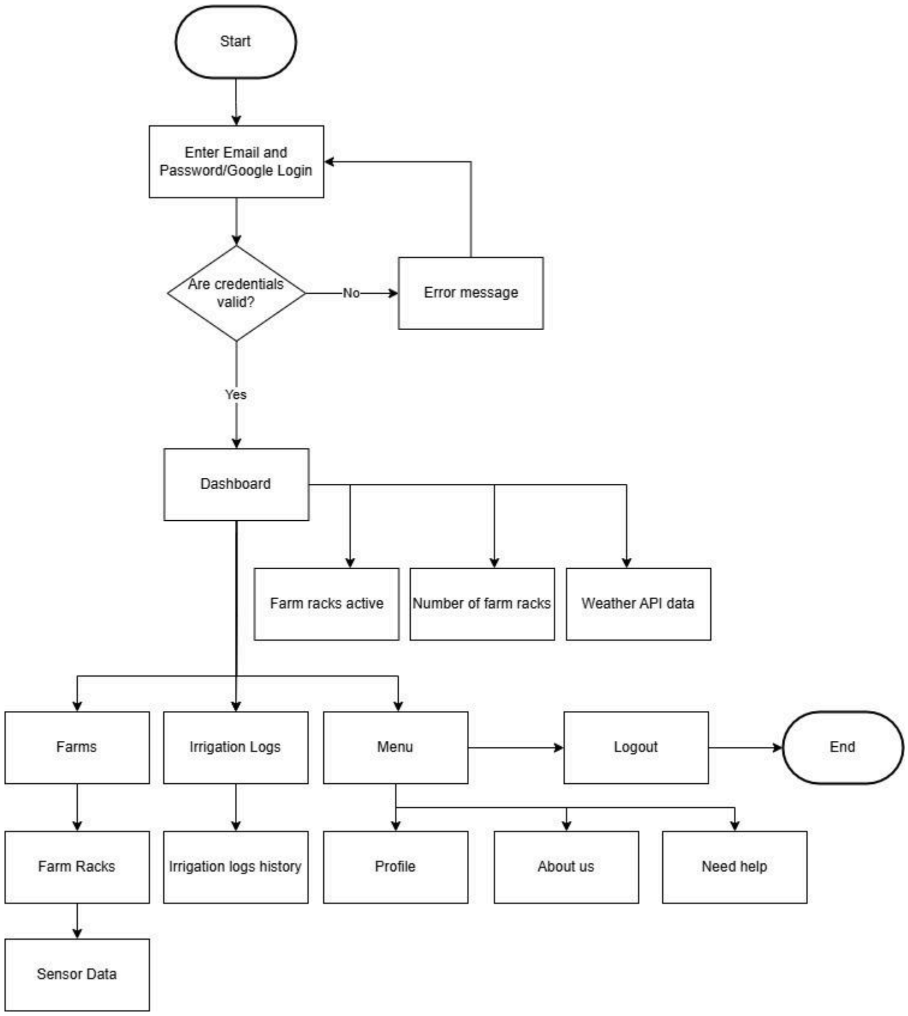


Figure 9. *VertiGrow User Flow Diagram*

The flowchart shows how users move through a VertiGrow monitoring app, starting with the login screen. They can sign in using their email and password or through Google. If the login details are incorrect, an error message appears and they're asked to try again. Once they successfully log in, they're taken to the main dashboard.

The dashboard acts as a central hub where users can see key info like how many farm racks are active, the total number of racks, and the latest weather data. From there, they can navigate to different sections like Farms, Irrigation Logs, or open the Menu. In the Farms section, they can dig deeper into individual farm racks and check out sensor data. The Irrigation Logs show a history of watering activity. The Menu gives access to the user's profile, an About Us section, and a Help page. When they're done, users can log out to end their session.

Accessibility Guidelines

To make VertiGrow accessible and user-friendly for a diverse group of users, including farmers with varying levels of tech familiarity, we applied the following design and accessibility principles:

1. Color & Contrast
   - The app uses a clean **white background** to reduce visual strain and keep the interface clear.
   - **Cards** use varying shades of **green (light and dark)** to visually separate information blocks while maintaining brand identity.
   - **White text** on green backgrounds ensures high contrast for readability.
   - Colors are selected to avoid reliance on hue alone—icons and labels supplement the information.

2. Navigation
    - The bottom navigation bar includes **four large, well-spaced buttons** (Dashboard, Farms, Irrigation Logs, Menu) that are easily tappable for most users, including those with motor impairments.
    - Navigation is consistent across all screens, reducing cognitive load and improving ease of use.
3. Guided Text & Feedback
    - Each page includes a **short instructional sentence** at the top, helping guide users (especially first-time users or non-tech-savvy farmers) on what the page does or how to use it.
    - Clear error messages and page-level cues are included when necessary (e.g., during login failures).
4. Icons & Visual Aids
    - Icons are added alongside descriptions to provide quick visual context.
    - In the **Farm Sensor Data** page, an **interactive image of a plant** is used to show sensor data locations, enhancing understanding through visualization.
    - Sensor values are accompanied by **distinct icons** for easier identification and quick scanning.
5. Visibility & Readability
    - Font sizes are selected to be comfortably readable across various device sizes.

- Content is center-aligned or consistently laid out to reduce scanning effort.
- Bright visual cues and green card hierarchy improve user attention and data focus.

DEPLOY & MAINTENANCE

Deployment Plan

Release Process

The Android application will be distributed through a GitHub-hosted landing page. The APK file will be uploaded to the project's GitHub repository, and a direct download link will be embedded in the landing page. Users can access the page via a web browser and click the download button to obtain the APK file for manual installation. Prior to release, the APK will be signed and tested to ensure security and stability across supported Android devices.

Rollout Strategy

As this app is developed for project purposes, the APK will be released directly through a GitHub-hosted landing page. The APK will be shared with project evaluators, instructors, or a small group of testers to demonstrate functionality. No public release or staged deployment is planned beyond this academic use.

APPENDICES

Glossary

Table 1. *Technical Terms and Acronyms*

Term	Description
API (Application Programming Interface)	A set of protocols and tools for building software and applications. VertiGrow uses APIs like Firebase and OpenWeatherMap to handle data and services.
App Check (Firebase App Check)	A Firebase service that helps ensure only your app can access your backend services, adding an extra layer of security.
Authentication	The process of verifying a user's identity. In VertiGrow, users can sign in via email/password or Google Sign-In.
ESP32	A low-cost microcontroller with Wi-Fi and Bluetooth capabilities used in the system to collect and transmit sensor data to the backend.
Firebase	A Backend-as-a-Service (BaaS) platform used in VertiGrow for authentication, real-time databases, Firestore storage, and analytic
Firestore	A flexible, scalable NoSQL cloud database used by VertiGrow for storing structured app data such as users, farms, and logs.
Fuzzy Logic	A decision-making method that mimics human reasoning by handling uncertain or imprecise data. Used in VertiGrow for automated irrigation control.
Google Sign-In	An OAuth 2.0-based authentication system that allows users to sign in to VertiGrow using their Google account.
IoT (Internet of Things)	A network of physical devices connected to the internet that collect and exchange data. VertiGrow uses IoT via sensors and ESP32 modules.
JSON (JavaScript Object Notation)	A lightweight data-interchange format used in VertiGrow to structure and exchange data, particularly in API responses.

LEACH (Low-Energy Adaptive Clustering Hierarchy)	An algorithm for wireless sensor networks that helps reduce energy usage. It’s used in VertiGrow to enhance sensor efficiency.
MVVM (Model-View-ViewModel)	A design pattern that separates data (Model), UI (View), and logic (ViewModel) to make the app easier to manage and maintain.
MPAndroidChart	An open-source charting library used to create interactive visualizations of data like water and energy consumption in VertiGrow.
OkHttp	An HTTP client for Android and Java used in VertiGrow to handle API requests, such as weather data from OpenWeatherMap.
OpenWeatherMap API	A service providing weather data via API, integrated into VertiGrow for real-time environmental context.
Play Integrity API	A security feature used to verify the authenticity of the app and prevent abuse.
Real-time Database	A Firebase database service that syncs data between users and devices in real time, used in VertiGrow for live sensor updates.
Sensor	A device that measures physical input from the environment. VertiGrow uses sensors for temperature, humidity, soil moisture, water level, and light.
UI/UX (User Interface/User Experience)	Refers to the design and interaction elements of the app. VertiGrow follows Material Design principles for better usability.

REFERENCES & SOURCES

Fundamentals. (n.d.). Firebase. https://firebase.google.com/docs/guides

Guide to app architecture. (n.d.). Android Developers. https://developer.android.com/topic/architecture

OpenWeatherMap.org. (n.d.). *Weather API - OpenWeatherMap*. https://openweathermap.org/api

Play Integrity API. (n.d.). Android Developers. https://developer.android.com/google/play/integrity

Using OAuth 2.0 to access Google APIs. (n.d.). Google for Developers. https://developers.google.com/identity/protocols/oauth2

What is the best way to document architecture for mobile applications? (2023, September 26). https://www.linkedin.com/advice/0/what-best-way-document-architecture-mobile