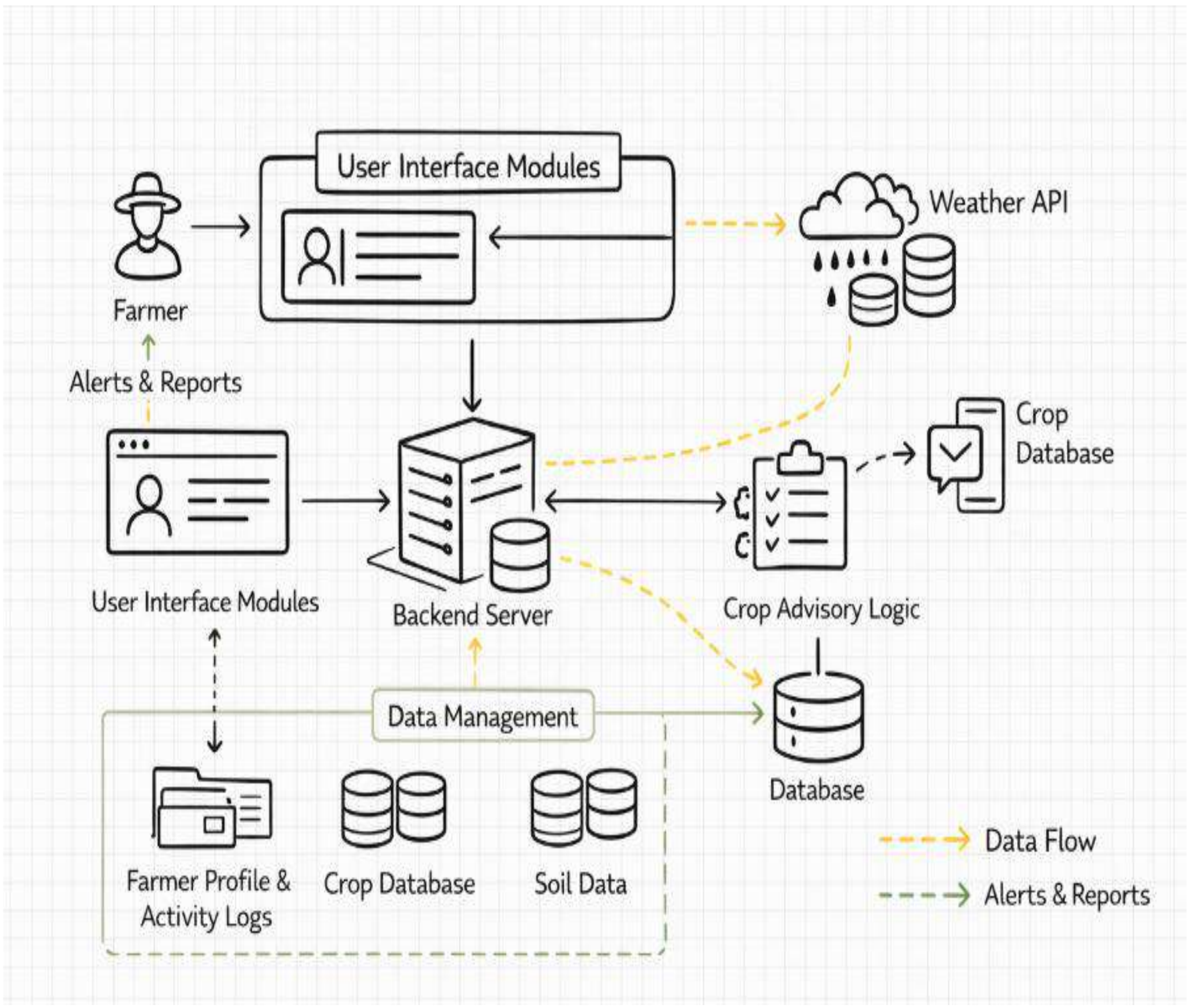
Project Title: AgriBuddy – Smart Crop Scheduling & Advisory Web Application

NAME: Roshini Pininti

REGISTRATION NUMBER: 23BCE5129

Architecture Diagram



Software Requirements Specification (SRS)

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to outline the functional and non-functional requirements for the **AgriBuddy** platform — a responsive web application aimed at assisting small and medium-scale farmers with intelligent crop scheduling and agricultural decision-making. The application provides personalised irrigation and fertilisation alerts based on weather forecasts, crop advisories based on local conditions, and a crop diary to log and track farming activities. This SRS will serve as a reference for developers, testers, and stakeholders
throughout the development lifecycle.

1.2 Document Conventions

- **UI**: User Interface
- **DB**: Database
- **API**: Application Programming Interface
- **CSV**: Comma-Separated Values
- **CRUD**: Create, Read, Update, Delete

1.3 Intended Audience and Reading Suggestions

This document is intended for the following stakeholders involved in the development and deployment of AgriBuddy:

- Software Developers: To design the frontend, backend, APIs, and database schemas according to the defined requirements.
- Project Managers and Clients: To understand the scope, timelines, and expectations from the system.
- Test Engineers: To derive functional and non-functional test cases directly from defined use cases and specifications.
- Agricultural Domain Experts: To validate whether the advisory logic and data are aligned with current agricultural standards.
- End-Users (Farmers): To eventually use the system based on their daily agricultural needs (not to read the document, but to benefit from the outcomes defined here).

1.4 Project Scope

AgriBuddy is a smart crop scheduling and advisory **web-based platform** designed to make agriculture smarter, especially for rural and semi-urban farmers with access to basic smartphones or computers. The platform offers an intuitive interface that connects farmers with data-driven agricultural decisions by combining the following:

- **Real-time Weather Data Integration**: Farmers receive timely alerts for irrigation, fertilization, and climate risks using APIs like OpenWeatherMap.
- **Soil Condition Analysis**: Local or uploaded soil data helps tailor crop and input recommendations.
- **Smart Crop Calendar**: A personalized calendar for each user detailing crop stages, recommended practices, and resource inputs.
- **Crop Diary Logging**: Farmers can track their own actions (e.g., sowing, watering, harvesting), creating a digital logbook.
- **Localized Crop Recommendations**: Based on district/state, rainfall, and seasonal patterns.
- **Analytics & Reports**: Visualizations and charts for trends in yield, input efficiency, and planning.

1.5 References

- OpenWeatherMap API Documentation
- Indian Council of Agricultural Research (ICAR) Guidelines
- Fundamentals of Web Development – Randy Connolly, Ricardo Hoar

2. OVERALL DESCRIPTION

2.1 Product Perspective

AgriBuddy is a responsive, web-based agricultural advisory system developed as a standalone but modular platform that integrates external data sources like weather APIs, soil databases, and crop advisory datasets. The system aims to assist farmers in managing crop cycles, scheduling irrigation/fertilization tasks, and improving yield efficiency based on real-time and historical data.

The platform comprises the following major components:

- **Farmer Profile Module:** Stores user-specific information such as farmer ID, name, location (village/district), land area, and preferred language. This data is used to personalize advice and alerts.
- **Crop Logbook (Diary):** Enables farmers to record crop-specific activities such as sowing date, fertilizer applied, irrigation events, and pest treatments. This helps track crop progress and automate recommendations.
- **Advisory & Scheduling Module:** Provides intelligent alerts and suggestions based on:

1. Weather forecasts from integrated APIs
2. Soil data (type, moisture)
3. Crop type and calendar

This module helps schedule irrigation and fertilization activities for optimal yield.

- **Weather & Soil Integration:** Real-time weather data is fetched using APIs (e.g., OpenWeather). Soil data can either be uploaded manually (CSV) or pulled from government APIs where available.
- **Notification System:** Sends advisory alerts through:

1. On-site notifications
2. Email or optional SMS integration

- **Analytics Dashboard:** Presents visual insights such as:
    1. Weekly weather summary
    2. Crop growth timeline
    3. Fertilizer/water usage tracking
    4. Predicted harvest windows

Each component is connected via RESTful APIs and relies on a centralized backend database to maintain user sessions, crop data, and historical logs.

2.2 Product Features

The major features of the AgriBuddy platform, as reflected in its conceptual Entity–Relationship (ER) Model, include the following:

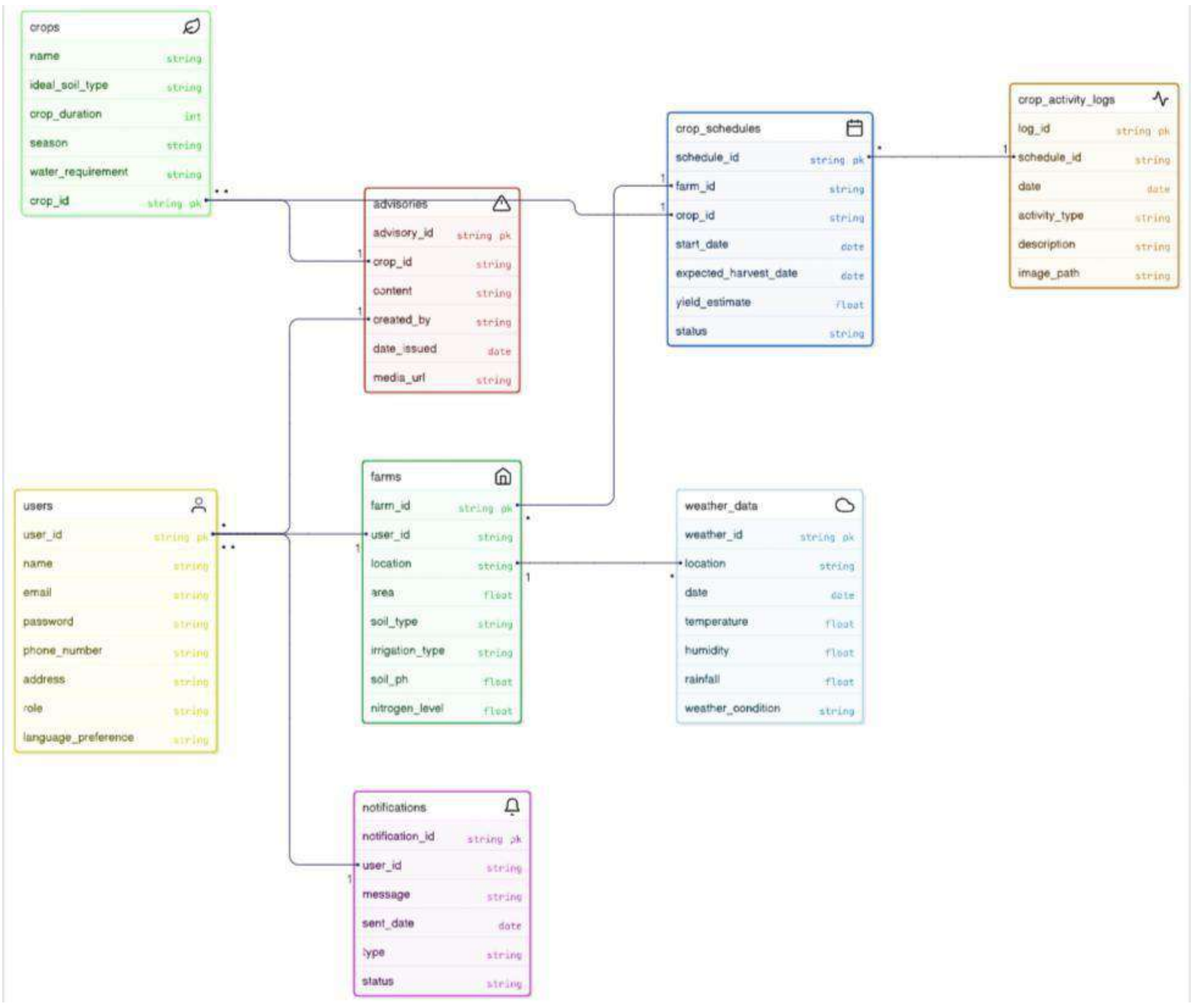


FIGURE-1 E-R DIAGRAM

2.3 User Classes and Characteristics

The AgriBuddy platform supports two primary types of users based on privilege levels:

1. **Farmer (End-User)**

Farmers are the primary users of the system and will have access to personalized crop planning and tracking functionalities. The system interface will be designed with simplicity and rural usability in mind (multilingual options, minimal input complexity). Key functions for farmers include:

- Create/Update Farm Profile: Register farm land location, size, and cropping details
- View Crop Suggestions: Based on current season, soil type, and weather data
- Receive Smart Alerts: Irrigation and fertilization notifications, pest/disease warnings

- Maintain Crop Diary: Log sowing, irrigation, fertilizer/pesticide usage, harvest dates
- Access Weather Forecasts: 7-day forecast integrated with crop advisory logic
- Receive Yield Predictions: Based on timeline, activity logs, and environmental conditions

2. **Agricultural Expert / Admin**

Agricultural experts or system administrators will have additional privileges to manage the backend content, maintain crop recommendation rules, and monitor system usage.

- Manage Crop Database: Add/update crop requirements, ideal conditions, and scheduling logic
- Manage User Reports and Feedback: View user-submitted issues, send targeted advisories or alerts
- Moderate Advisory Engine Rules: Calibrate thresholds for alerts based on feedback or policy changes
- Oversee Soil and Weather API Integrations: Ensure data freshness and source reliability
- Analyze Usage Metrics: View user activity patterns, success of crop recommendations, etc.

Additional Characteristics:

- Connectivity: Users may access the system from low-bandwidth areas. Hence, offline caching and SMS/email-based alerts will be considered.
- Device: Designed to be responsive for mobile browsers, tablets, and desktop usage.
- Multilingual Support: Core regions may use Hindi, Tamil, Telugu, or other regional languages alongside English.

2.4 Operating Environment

- OS: Windows/Linux/Mac
- Browsers: Chrome, Firefox, Edge (Latest Versions)
- Languages: HTML, CSS, JavaScript (React or Vue), Python (Flask/Django)
- Database: MySQL/PostgreSQL
- API: Weather and Soil Data APIs

2.5 Design and Implementation Constraints

- Must work offline for basic features with sync when reconnected
- Should support multilingual interfaces
- API limits on free weather/soil data access
- Should use RESTful architecture for backend communication

2.6 Assumptions and Dependencies

- The system assumes internet connectivity, but low-bandwidth support will be available through offline caching and SMS alerts.
- Weather and soil data depend on reliable APIs and government datasets.
- The web application will be accessed primarily from rural regions, so UI is optimized for mobile browsers.
- The system assumes crop advisory rules to be deterministic and updated periodically by admin/agri experts.
- In future scope, the architecture may shift to a distributed database with regional nodes across states (e.g., Maharashtra, Tamil Nadu, Punjab, Assam).
- The system may integrate with government DBs like *Soil Health Card*, *Fasal Bima Yojana*, or *eNAM* for broader functionality.

3. SYSTEM FEATURES

- **Description and Priority**

AgriBuddy maintains critical agricultural data such as crop types, farming activities, weather conditions, and user preferences. As food production and rural livelihood depend heavily on timely farming decisions, this system has a high priority in digital agriculture initiatives.

- **Stimulus/Response:**

1. *User adds a crop* → System fetches weather → Shows watering schedule.
2. *Rain forecasted* → Sends SMS/email to farmers with irrigation advice.

- **Functional Requirements**

Other system features include:

1. **Distributed Database**

- AgriBuddy's data is stored across distributed databases to ensure high availability and low-latency access in rural regions.
- Ensures that advisory and alert services are uninterrupted, even during connectivity issues in specific areas.

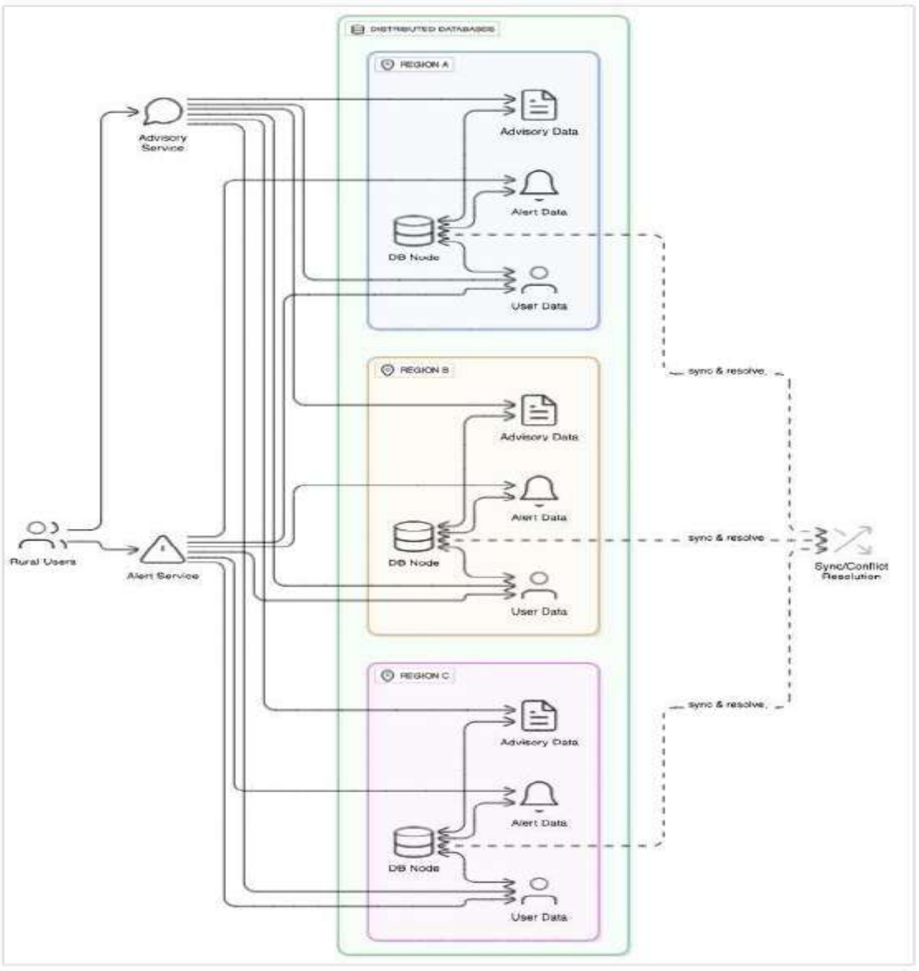


FIGURE-2 DATABASE DISTRIBUTION

2. **Client/Server System**

- Frontend (client): The farmer interface accessed via mobile/desktop browsers.
- Backend (server): Cloud-hosted database and business logic engine.
- All agricultural data resides at the server site, while interactions and advisories occur on the client side.

4. EXTERNAL INTERFACE REQUIREMENTS

4.1 User Interfaces

- Mobile-friendly responsive design
- Form-based input for logs
- Graphical dashboard with filters and legends

4.2 Hardware Interfaces

- Runs on any modern device with Windows/Linux/macOS and browser compatibility
- Browser support includes Chrome, Firefox, Edge, Safari supporting HTML5, CSS3, and JavaScript

4.3 Software Interfaces

Software Used	**Description**
Operating System	Windows/Linux for flexibility and wide hardware support
Database	PostgreSQL/MySQL used for storing user profiles, farm logs, and crop data
Frontend Framework	React.js for interactive and fast user experience
Backend Language	Python (Django/Flask) or Node.js to handle API logic and business rules

4.4 Communications Interfaces

- HTTPS protocol for secure data transfer between client and server
- RESTful APIs for communication with weather and soil databases
- JSON format used for data exchange
- Support for email/SMS notifications through SMTP and third-party gateway APIs

5. NON-FUNCTIONAL REQUIREMENTS

5.1 Performance Requirements

The implementation of the AgriBuddy platform involves the creation and optimization of an agricultural database. This includes:

A) E-R Diagram

The E-R Diagram represents the logical structure of the database in a pictorial format. It aids in:

- Organizing agricultural data as entities like User, Farm, Crop, Advisory, etc.
- Establishing attributes such as soil type, crop duration, location.
- Defining relationships such as "farmer owns farm", "farm grows crop", and "crop has advisory".

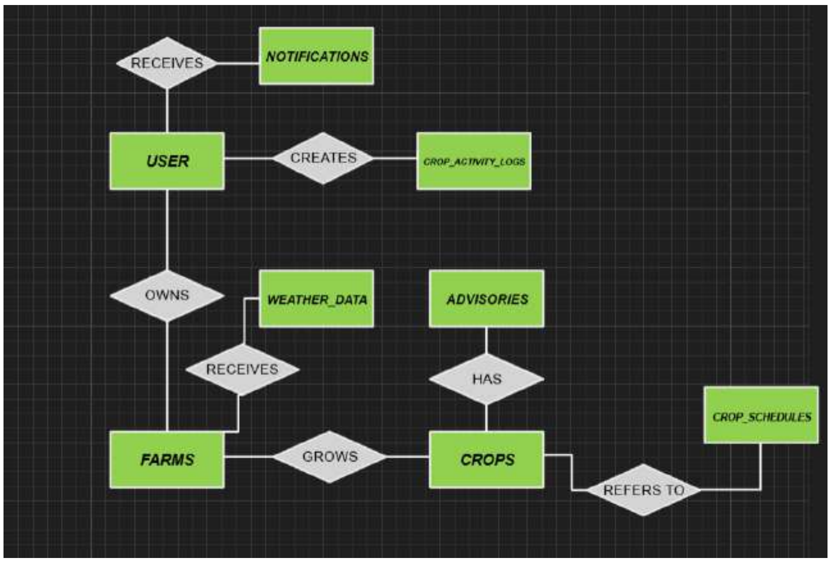


FIGURE-3 E-R DIAGRAM

B) Normalization

Normalization is employed to:

- Eliminate redundancy by ensuring data is stored only once.
- Reduce anomalies during data insertion, deletion, or updates.
- Break down the system into manageable, logically related tables.
- The system adheres to at least Third Normal Form (3NF) for relational integrity.

5.2 Safety Requirements

In the event of catastrophic failure (e.g., server crash), the system should:

- Restore the last backup of the database from secure cloud archival.
- Reapply all logged changes from the backup point to current using transaction logs.
- Ensure data integrity and minimal downtime through recovery scripts and alerts.

5.3 Security Requirements

To ensure confidentiality and data protection:

- All user credentials and sensitive data will be encrypted.
- Role-based access control (RBAC) will be implemented.
- HTTPS will be enforced for all communication.
- Firewalls and periodic vulnerability assessments will be conducted.

5.4 Software Quality Attributes

- Availability: The advisory and dashboard services should be accessible 24/7, even under low-bandwidth conditions.

- Correctness: Recommendations must align with scientific agricultural practices and validated data sources.
- Maintainability: The system should be modular, with clear codebase separation to allow easy upgrades.
- Usability: The platform must be intuitive, multilingual, and farmer-friendly with icon-driven interfaces.

WORK BREAKDOWN STRUCTURE

Deliverable-Based Work Breakdown Structure

In AgriBuddy, each deliverable is a critical component of the complete system and represents a part of the working application. This structure helps in decomposing the full application scope into manageable submodules and ensures that each module aligns with the functional goals of the platform — such as crop recommendation, schedule generation, weather-based alerts, and user advisory.

The Level 1 deliverables of this WBS include:

- Requirement Specification Document
- Frontend Design (Web UI)
- Backend API Services
- Crop Schedule Logic Engine
- Weather & Advisory Module
- Database Design & Integration
- Notification System
- Testing & Validation
- Final Report and Submission

Each of these deliverables is further broken down:

- The Frontend Design deliverable includes wireframes, HTML/CSS layout, input forms for farm and crop details, and responsive design testing.
- The Backend API Services deliverable involves building Flask/Node.js APIs for CRUD operations, connecting farm/crop data to logic engines, and secure authentication handling.
- The Crop Schedule Logic Engine implements algorithmic recommendations based on crop type, region, and seasonal data.
- The Weather & Advisory Module connects to external APIs to fetch real-time weather data and generate dynamic advisory alerts.
- The Database Design & Integration task includes defining schemas for User, Farm, Crop, ActivityLog, Notification, and connecting them to frontend and backend modules.
- The Notification System includes scheduled and event-triggered alert features via SMS/email or in-app.

This structure supports proper task ownership, ensures modular testing, and provides a clear breakdown of what the project will deliver. It also makes report writing and viva preparation easier as every component is documented as an individual output.

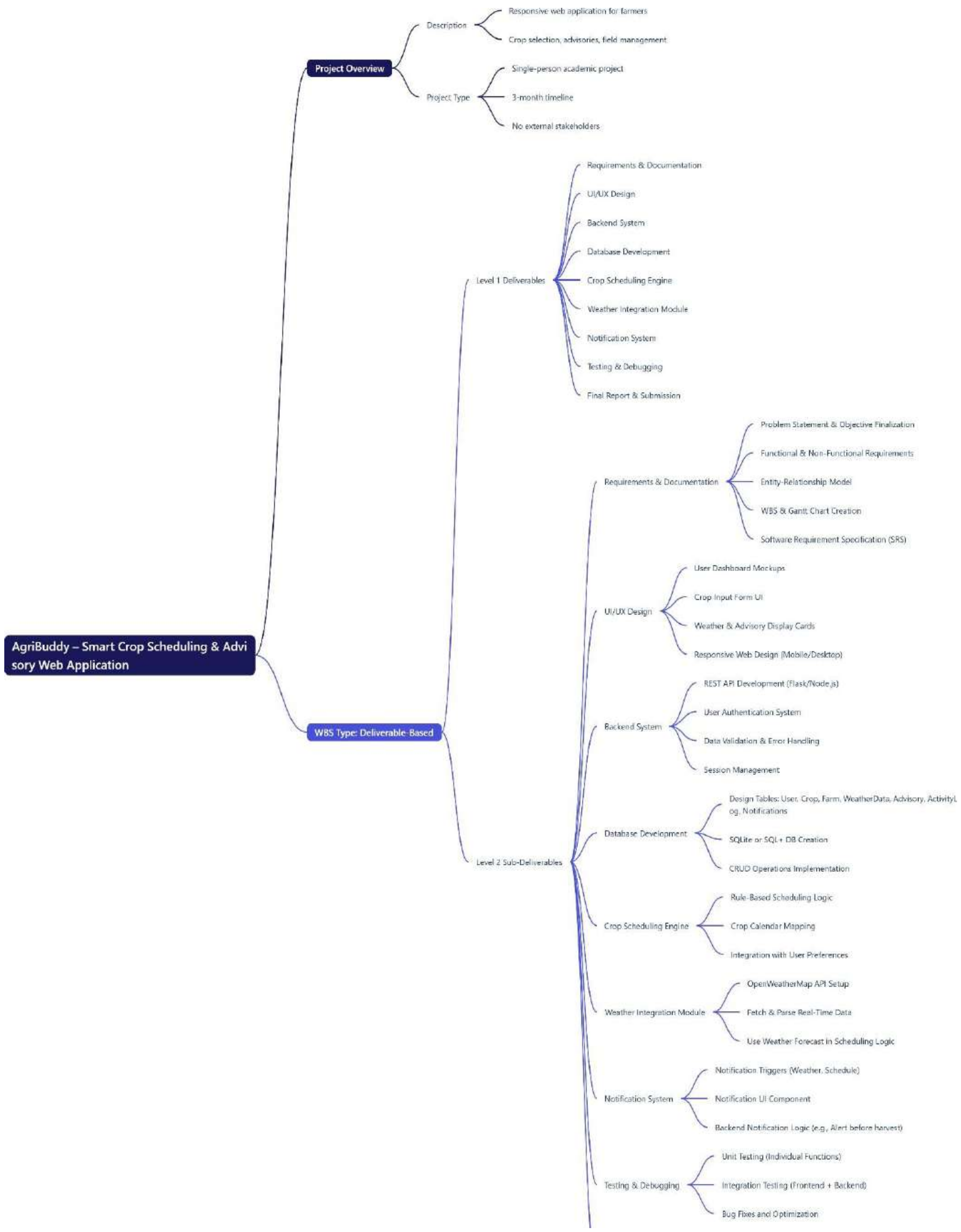

AgriBuddy – Smart Crop Scheduling & Advisory Web Application
Project Overview
Description
Responsive web application for farmers
Crop selection, advisories, field management
Project Type
Single-person academic project
3-month timeline
No external stakeholders
WBS Type: Deliverable-Based
Level 1 Deliverables
Requirements & Documentation
UI/UX Design
Backend System
Database Development
Crop Scheduling Engine
Weather Integration Module
Notification System
Testing & Debugging
Final Report & Submission
Level 2 Sub-Deliverables
Requirements & Documentation
Problem Statement & Objective Finalization
Functional & Non-Functional Requirements
Entity-Relationship Model
WBS & Gantt Chart Creation
Software Requirement Specification (SRS)
UI/UX Design
User Dashboard Mockups
Crop Input Form UI
Weather & Advisory Display Cards
Responsive Web Design (Mobile/Desktop)
Backend System
REST API Development (Flask/Node.js)
User Authentication System
Data Validation & Error Handling
Session Management
Database Development
Design Tables: User, Crop, Farm, WeatherData, Advisory, ActivityLog, Notifications
SQLite or SQL + DB Creation
CRUD Operations Implementation
Crop Scheduling Engine
Rule-Based Scheduling Logic
Crop Calendar Mapping
Integration with User Preferences
Weather Integration Module
OpenWeatherMap API Setup
Fetch & Parse Real-Time Data
Use Weather Forecast in Scheduling Logic
Notification System
Notification Triggers (Weather, Schedule)
Notification UI Component
Backend Notification Logic (e.g., Alert before harvest)
Testing & Debugging
Unit Testing (Individual Functions)
Integration Testing (Frontend + Backend)
Bug Fixes and Optimization

FIGURE 1 – DELIVERABLE BASED WORK BREAKDOWN STRUCTURE

Phase-Based Work Breakdown Structure

In a Phase-Based Work Breakdown Structure for the AgriBuddy project, the entire development cycle is broken into logical phases based on time and progression of work, typically followed in a structured academic project with a 3-month timeline. This structure emphasizes "when and how" the work is done, with deliverables grouped under each phase according to when they are to be completed.

For AgriBuddy, the Level 1 phases and their associated deliverables (Level 2) are:

🔷 **Phase 1: Planning & Requirement Analysis**

- Defining the project scope and objective
- Functional and non-functional requirement collection
- Preparing SRS document
- ER diagram and use-case modeling

🔷 **Phase 2: System & UI Design**

- Low-fidelity wireframes for user flows (farm entry, schedule view, etc.)
- Final UI/UX design of pages: Home, Login, Dashboard, Crop Advisory
- Frontend tech selection (HTML, CSS, JS, Bootstrap/Tailwind)
- Database schema design for entities: User, Farm, Crop, etc.

🔷 **Phase 3: Module Development**

- Implementing user login and authentication
- Building Farm, Crop, and ActivityLog CRUD features
- Integrating crop schedule logic (based on sowing/harvest rules)
- Developing advisory logic linked to weather APIs
- Backend and database connection setup (Flask + SQLite/PostgreSQL)

🔷 **Phase 4: Testing and Debugging**

- Frontend unit testing (form validation, alerts)
- Backend route/API testing using Postman
- Database integrity check (foreign key relations, data flows)
- Fixing UI/logic bugs
- Test run of crop schedule and alert notification triggers

🔷 **Phase 5: Deployment and Documentation**

- Deploying frontend and backend locally or on Heroku/Render

- Preparing final report: screenshots, outcomes, WBS & Gantt

This phase-wise structure makes it easier to monitor progress at each milestone, aligns with academic project reporting cycles, and simplifies risk management in case any module is delayed.

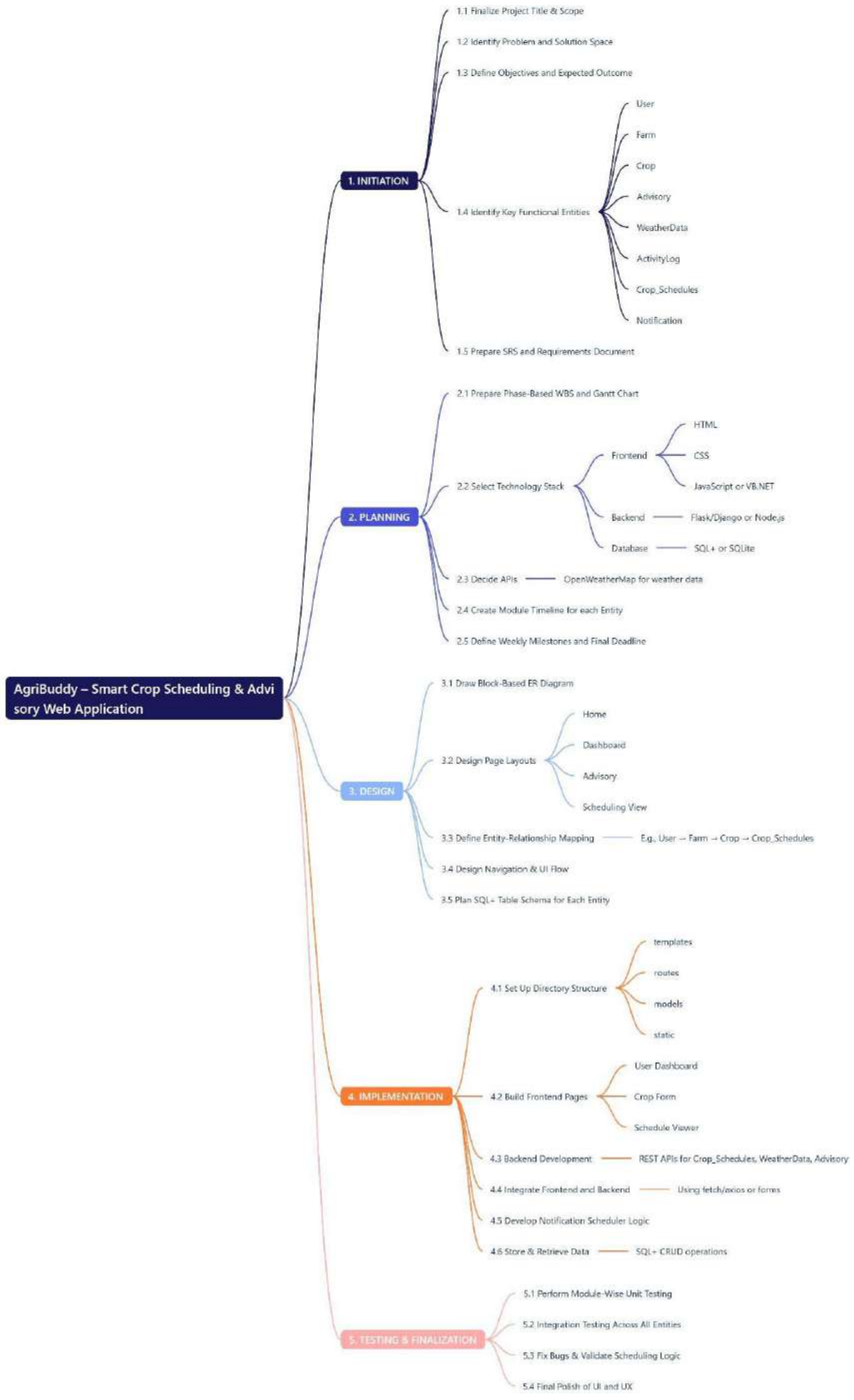
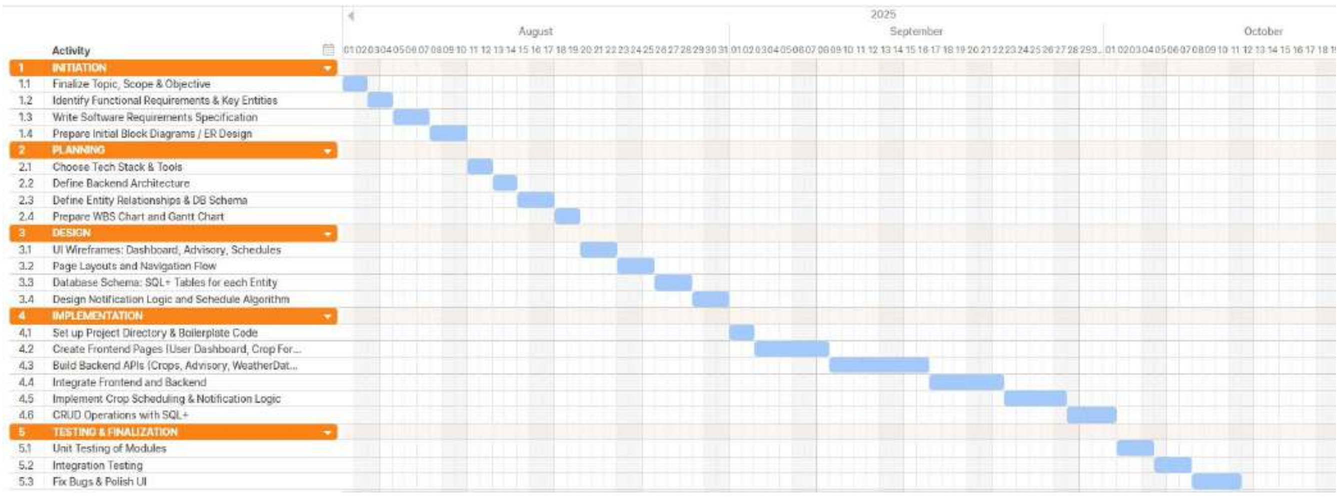

AgriBuddy – Smart Crop Scheduling & Advi sory Web Application
1. INITIATION
1.1 Finalize Project Title & Scope
1.2 Identify Problem and Solution Space
1.3 Define Objectives and Expected Outcome
1.4 Identify Key Functional Entities
User
Farm
Crop
Advisory
WeatherData
ActivityLog
Crop_Schedules
Notification
1.5 Prepare SRS and Requirements Document
2. PLANNING
2.1 Prepare Phase-Based WBS and Gantt Chart
2.2 Select Technology Stack
Frontend
HTML
CSS
JavaScript or VB.NET
Backend
Flask/Django or Node.js
Database
SQL+ or SQLite
2.3 Decide APIs
OpenWeatherMap for weather data
2.4 Create Module Timeline for each Entity
2.5 Define Weekly Milestones and Final Deadline
3. DESIGN
3.1 Draw Block-Based ER Diagram
3.2 Design Page Layouts
Home
Dashboard
Advisory
Scheduling View
3.3 Define Entity-Relationship Mapping
E.g., User → Farm → Crop → Crop_Schedules
3.4 Design Navigation & UI Flow
3.5 Plan SQL+ Table Schema for Each Entity
4. IMPLEMENTATION
4.1 Set Up Directory Structure
templates
routes
models
static
4.2 Build Frontend Pages
User Dashboard
Crop Form
Schedule Viewer
4.3 Backend Development
REST APIs for Crop_Schedules, WeatherData, Advisory
4.4 Integrate Frontend and Backend
Using fetch/axios or forms
4.5 Develop Notification Scheduler Logic
4.6 Store & Retrieve Data
SQL+ CRUD operations
5. TESTING & FINALIZATION
5.1 Perform Module-Wise Unit Testing
5.2 Integration Testing Across All Entities
5.3 Fix Bugs & Validate Scheduling Logic
5.4 Final Polish of UI and UX

FIGURE 2 - PHASE BASED WORK BREAKDOWN STRUCTURE

Gantt Chart:

The Gantt Chart for the AgriBuddy project outlines the structured timeline for all major activities over a three-month development period, organized into sequential and overlapping phases. The chart begins with the Planning and Requirement Analysis Phase in Week 1, where project scope, objectives, and required functionalities such as crop scheduling, farm management, weather integration, and user advisory systems are identified and documented. This is followed by the UI & Database Design Phase, where low-fidelity wireframes are created for all screens including user dashboards, crop inputs, farm profiles, and weather data display, while in parallel, the database schema is finalized with entities such as User, Farm, Crop, WeatherData, Advisory, and their relationships. By Week 5, the Frontend Development Phase begins, focusing on implementing a responsive user interface using HTML, CSS, and JavaScript frameworks like React or Vue.js. Simultaneously, the Backend Development Phase is initiated in Week 6, where RESTful APIs are developed using technologies such as Node.js or Django to manage data operations, scheduling logic, and advisory generation. These phases run concurrently until Week 8 to maintain timeline efficiency. The Integration & Testing Phase spans Week 9 and Week 10, where unit testing, bug fixing, and frontend-backend integration are performed to ensure stable functionality. The Final Deployment & Documentation Phase occupies the last two weeks, where the application is hosted on a local server or cloud platform (e.g., Firebase or Heroku), and technical documentation, user manual, and final project reports are prepared. Each task in the Gantt chart is represented as a horizontal bar against a time axis, visually depicting dependencies, overlaps, and durations, offering a clear and efficient roadmap for single-developer academic execution.



	Activity
1	INITIATION
1.1	Finalize Topic, Scope & Objective
1.2	Identify Functional Requirements & Key Entities
1.3	Write Software Requirements Specification
1.4	Prepare Initial Block Diagrams / ER Design
2	PLANNING
2.1	Choose Tech Stack & Tools
2.2	Define Backend Architecture
2.3	Define Entity Relationships & DB Schema
2.4	Prepare WBS Chart and Gantt Chart
3	DESIGN
3.1	UI Wireframes: Dashboard, Advisory, Schedules
3.2	Page Layouts and Navigation Flow
3.3	Database Schema: SQL+ Tables for each Entity
3.4	Design Notification Logic and Schedule Algorithm
4	IMPLEMENTATION
4.1	Set up Project Directory & Boilerplate Code
4.2	Create Frontend Pages (User Dashboard, Crop For...
4.3	Build Backend APIs (Crops, Advisory, WeatherDat...
4.4	Integrate Frontend and Backend
4.5	Implement Crop Scheduling & Notification Logic
4.6	CRUD Operations with SQL+
5	TESTING & FINALIZATION
5.1	Unit Testing of Modules
5.2	Integration Testing
5.3	Fix Bugs & Polish UI

FIGURE-3 GANTT CHART