Crop Monitoring System with Weather Simulation

Project Report

---

1. Cover Page

Project Title: Crop Monitoring System with Weather Simulation

Student Name: Rajveer Singh Solanki

Registration Number: 25BCE10591

Course: Computer Science

Faculty: D. Saravanan

Institution: VIT University

Date: November 2025

---

2. Introduction

The agricultural sector faces significant challenges due to unpredictable weather patterns and the need for precise crop management. Farmers often struggle to make timely decisions regarding irrigation, frost protection, and heat management, leading to crop losses and reduced yields.

This project presents a **Crop Monitoring System with Weather Simulation** that provides real-time weather analysis, crop-specific alerts, and actionable recommendations. The system simulates weather conditions and evaluates them against the optimal growing requirements of various crops, generating intelligent alerts for temperature extremes and irrigation needs.

The application features an intuitive Streamlit-based web interface that displays current weather conditions, a 7-day forecast, and crop-specific alerts, enabling farmers and agricultural professionals to make informed decisions.

---

3. Problem Statement

Problem

Farmers and agricultural managers lack access to integrated tools that combine weather monitoring with crop-specific recommendations. Existing solutions are either too complex, expensive, or not tailored to specific crop requirements.

Scope

This project provides a lightweight, web-based crop monitoring dashboard that:

- Simulates realistic weather data
- Supports 9 different crop types
- Generates temperature and irrigation alerts
- Displays a 7-day weather forecast

Target Users

- Small and medium-scale farmers
- Agricultural consultants
- Farming cooperatives
- Agricultural education institutions

High-Level Features

- Real-time weather simulation
- Multi-crop support with specific thresholds
- Temperature monitoring and frost/heat warnings
- Smart irrigation recommendations
- Interactive web-based dashboard

---

4. Functional Requirements

FR-01: Weather Data Simulation

The system shall generate simulated weather data including temperature, humidity, rainfall, wind speed, UV index, and soil moisture levels.

FR-02: 7-Day Forecast Generation

The system shall produce a 7-day weather forecast with daily temperature predictions and rain probability.

FR-03: Crop Database Management

The system shall maintain a database of crops with their optimal temperature ranges and water requirements.

FR-04: Temperature Alert Generation

The system shall analyze current temperature against crop-specific thresholds and generate appropriate alerts:

- Frost Warning (HIGH urgency)
- Low Temperature (MEDIUM urgency)
- Extreme Heat (HIGH urgency)
- High Temperature (MEDIUM urgency)

FR-05: Irrigation Alert Generation

The system shall evaluate soil moisture and upcoming rainfall to provide irrigation recommendations:

- Urgent Irrigation (HIGH urgency)
- Irrigation Needed (MEDIUM urgency)
- Rain Expected - Hold Irrigation (LOW urgency)

FR-06: Interactive Dashboard

The system shall provide a web-based interface displaying:

- Current weather metrics
- 7-day forecast visualization
- Crop selection dropdown
- Color-coded alert cards

FR-07: Weather Refresh Capability

The system shall allow users to refresh weather data on demand.

---

5. Non-Functional Requirements

NFR-01: Usability

The interface shall be intuitive and require no technical training. Users should be able to understand alerts within 5 seconds of viewing.

NFR-02: Performance

The system shall generate weather data and alerts within 500ms of user interaction.

NFR-03: Reliability

The system shall handle all edge cases without crashing, including extreme temperature values and missing data.

NFR-04: Maintainability

The codebase shall be modular with separate functions for weather simulation, temperature checking, and irrigation checking, allowing easy updates and extensions.

NFR-05: Scalability

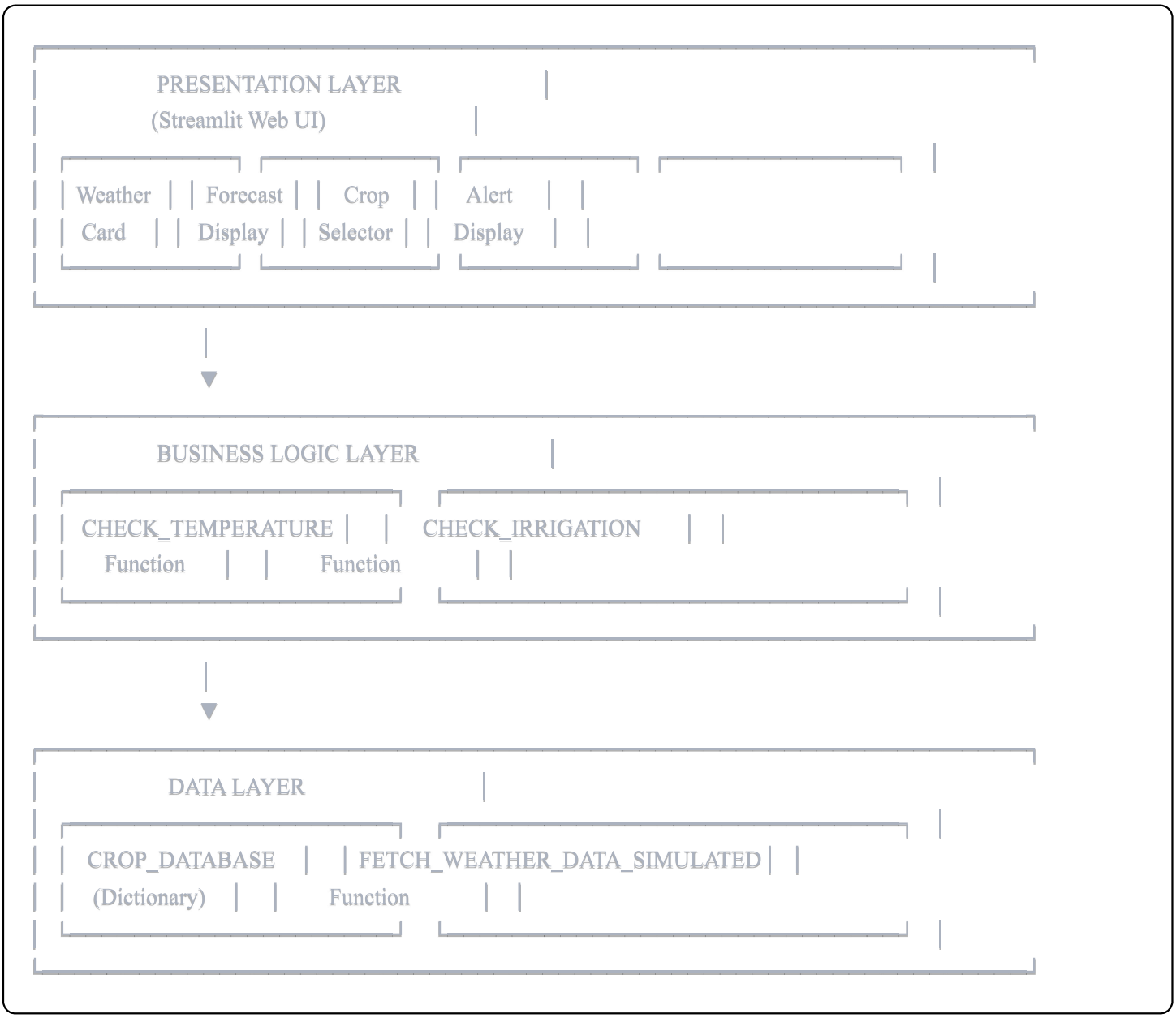
The crop database structure shall support easy addition of new crops without code modifications to alert logic.

NFR-06: Error Handling

The system shall gracefully handle invalid inputs and provide meaningful feedback to users.

---

6. System Architecture



Architecture Overview

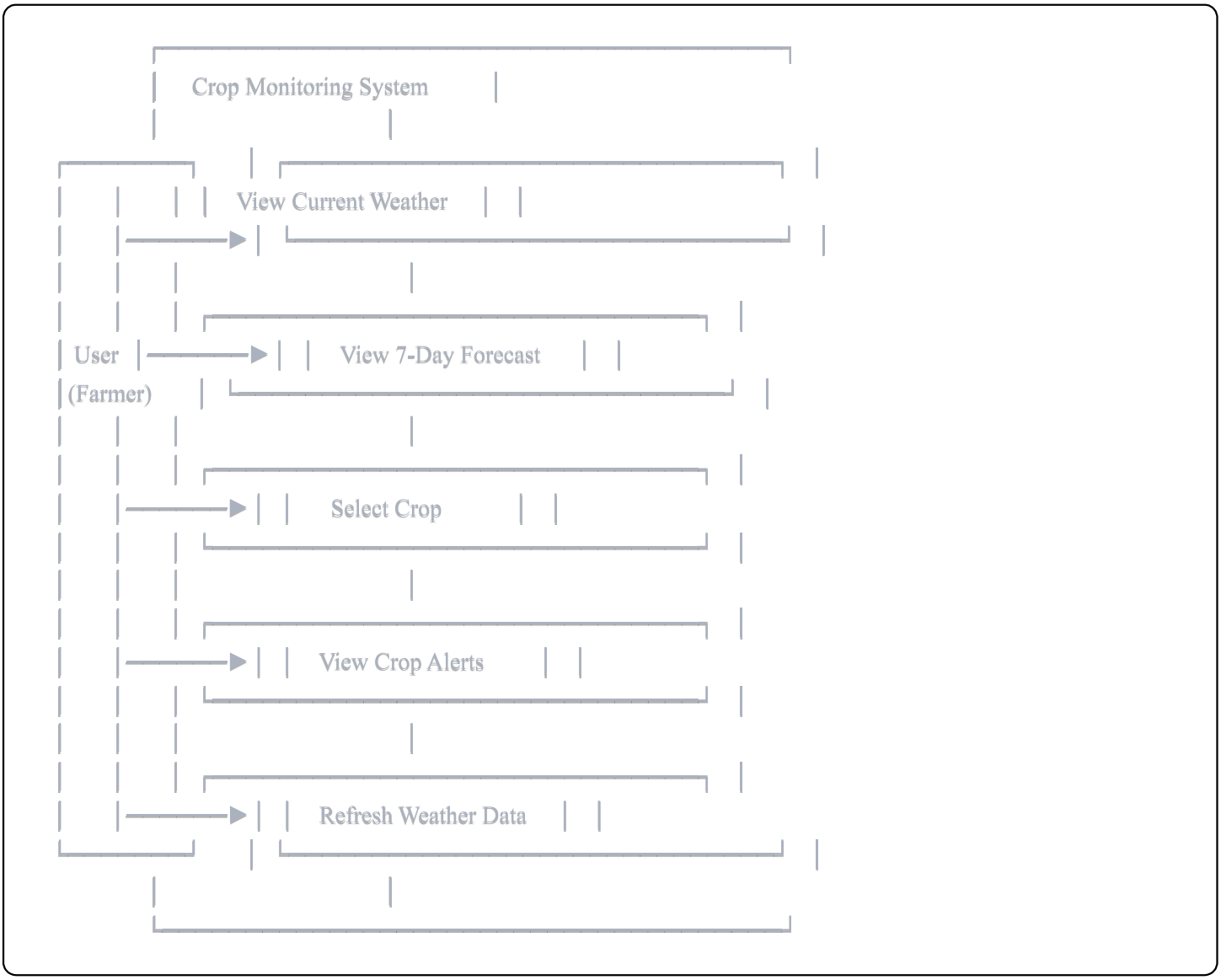
The system follows a **three-tier architecture**:

1. **Presentation Layer:** Streamlit-based web interface handling user interactions and data visualization.
2. **Business Logic Layer:** Core functions that process weather data against crop requirements to generate alerts.
3. **Data Layer:** Crop database and weather simulation engine providing the foundational data.

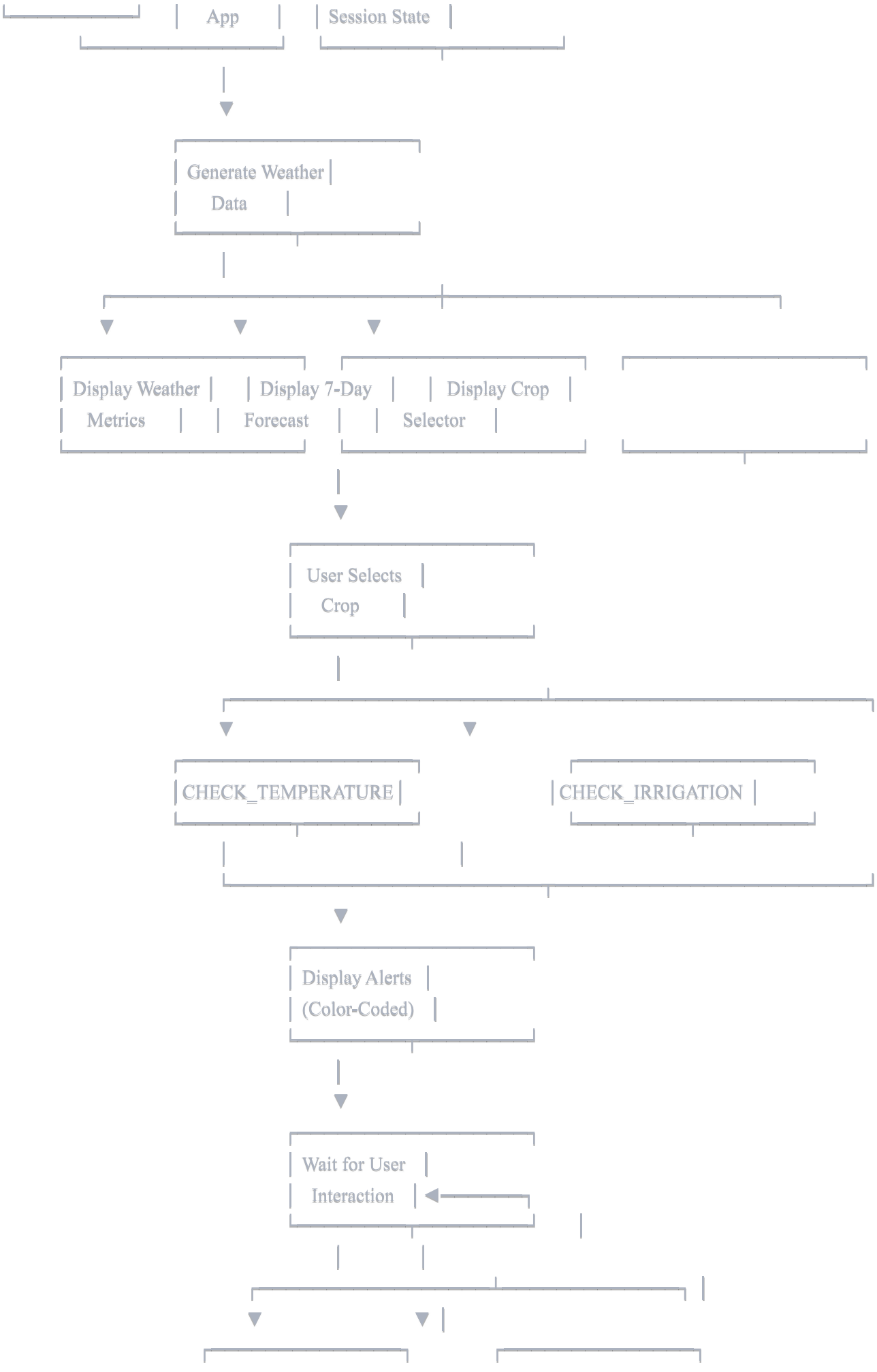
---

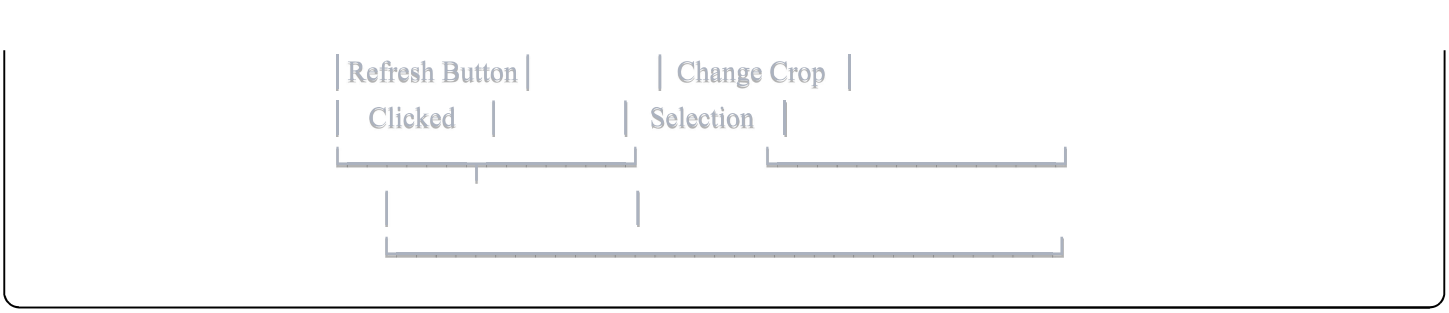
7. Design Diagrams

7.1 Use Case Diagram

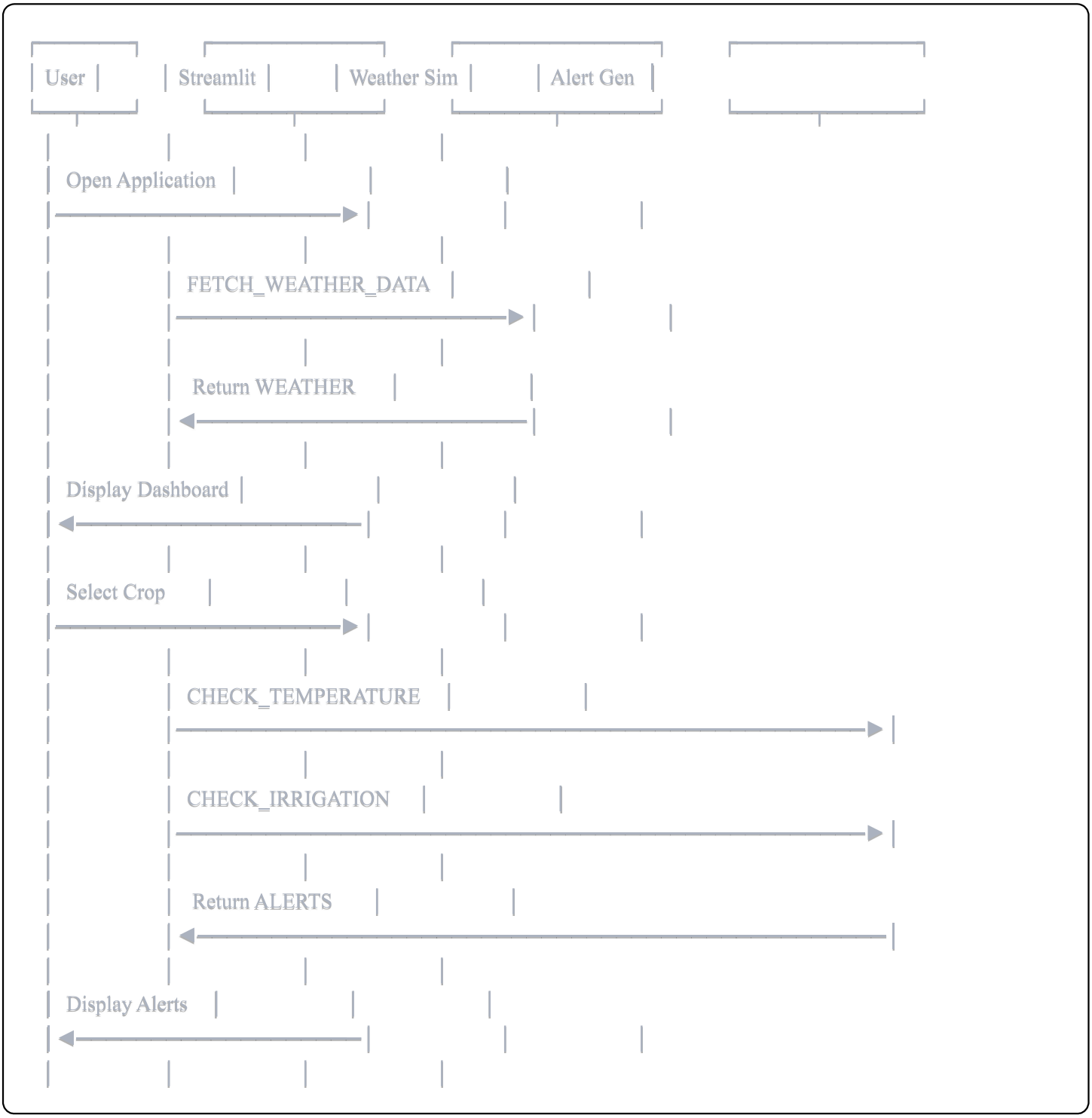


7.2 Workflow Diagram

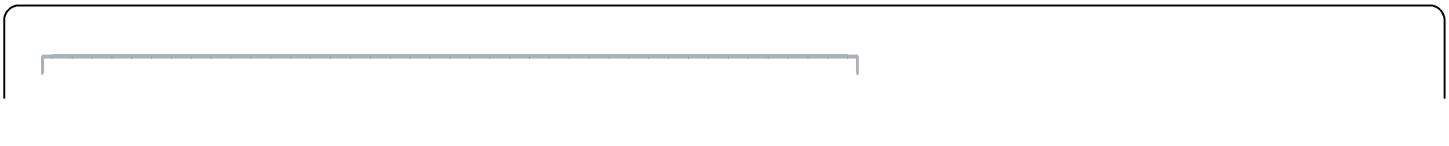

App
Session State
Generate Weather Data
Display Weather Metrics
Display 7-Day Forecast
Display Crop Selector
User Selects Crop
CHECK_TEMPERATURE
CHECK_IRRIGATION
Display Alerts (Color-Coded)
Wait for User Interaction

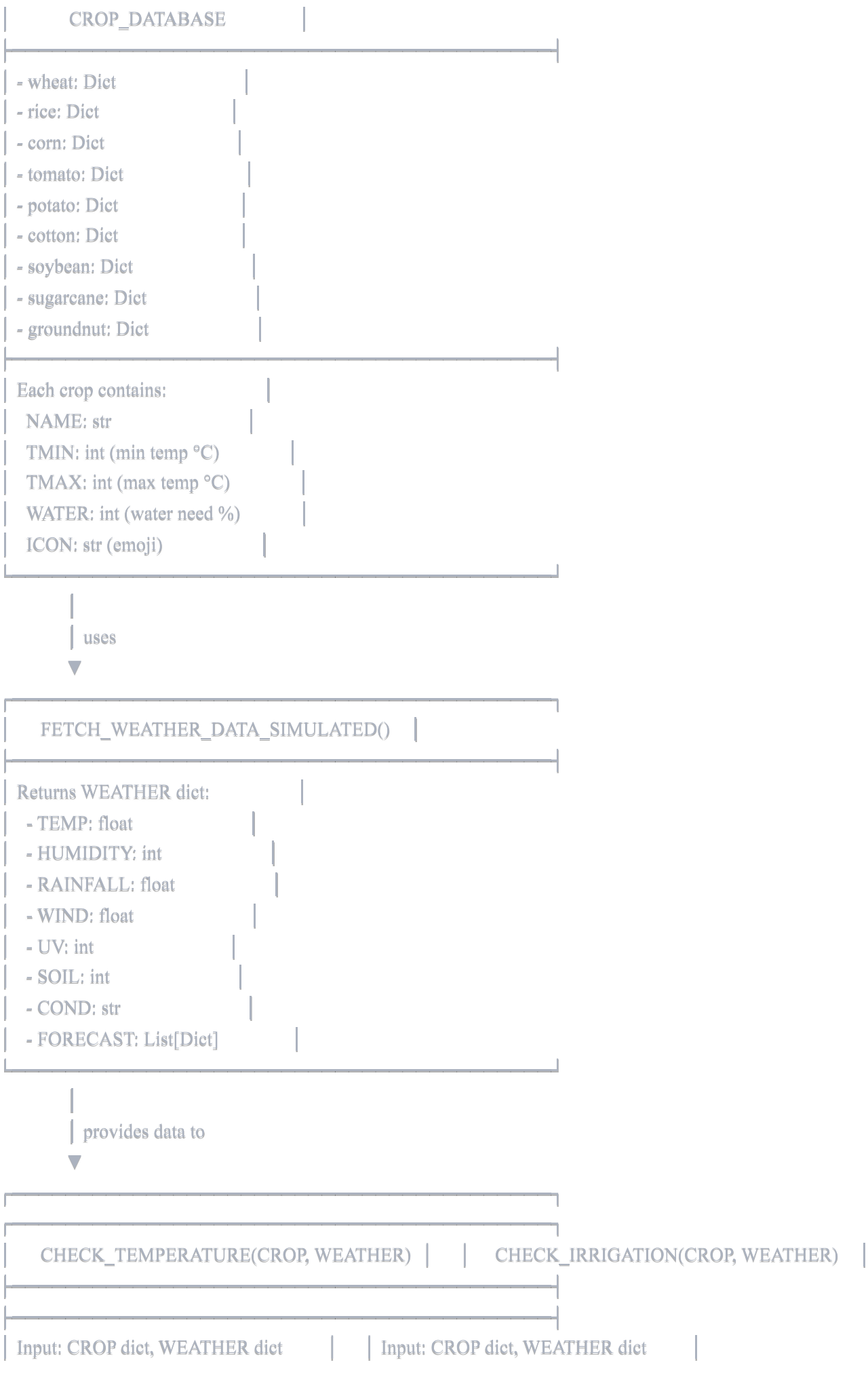


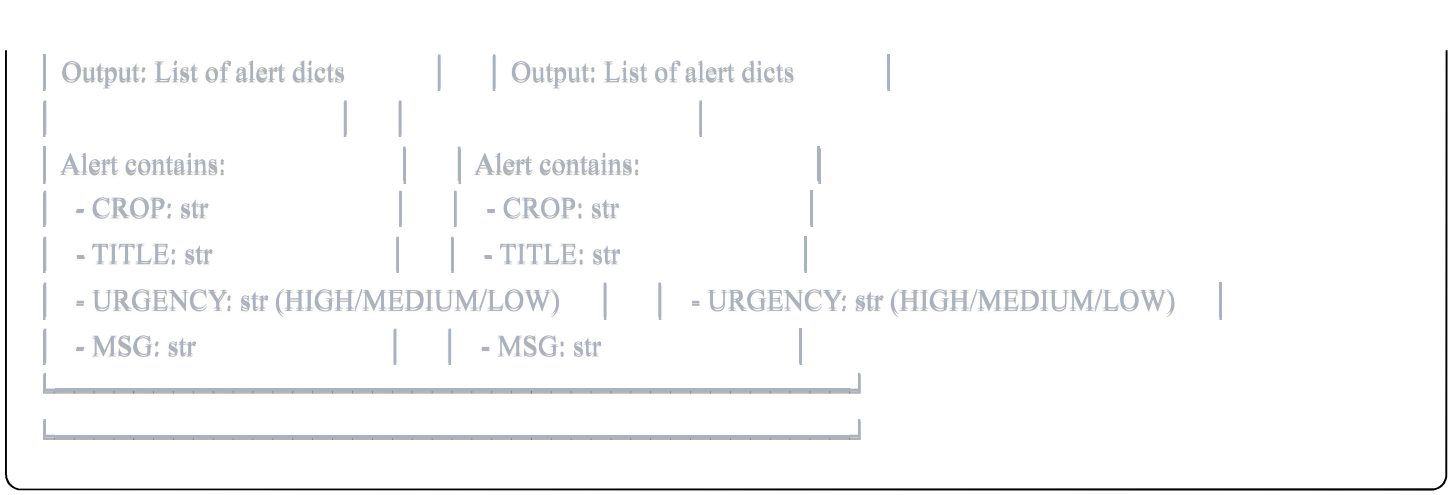
7.3 Sequence Diagram



7.4 Class/Component Diagram




CROP_DATABASE
- wheat: Dict
- rice: Dict
- corn: Dict
- tomato: Dict
- potato: Dict
- cotton: Dict
- soybean: Dict
- sugarcane: Dict
- groundnut: Dict
Each crop contains:
NAME: str
TMIN: int (min temp °C)
TMAX: int (max temp °C)
WATER: int (water need %)
ICON: str (emoji)
uses
FETCH_WEATHER_DATA_SIMULATED()
Returns WEATHER dict:
- TEMP: float
- HUMIDITY: int
- RAINFALL: float
- WIND: float
- UV: int
- SOIL: int
- COND: str
- FORECAST: List[Dict]
provides data to
CHECK_TEMPERATURE(CROP, WEATHER)
CHECK_IRRIGATION(CROP, WEATHER)
Input: CROP dict, WEATHER dict
Input: CROP dict, WEATHER dict



---

8. Design Decisions & Rationale

8.1 Technology Choice: Streamlit

Decision: Use Streamlit for the web interface. **Rationale:** Streamlit provides rapid prototyping capabilities with minimal frontend code. It integrates seamlessly with Python, allowing the existing logic to be used without modification.

8.2 Data Structure: Dictionaries

Decision: Use Python dictionaries for crop database and weather data. **Rationale:** Dictionaries provide O(1) lookup time and are easily extensible. New crops can be added without modifying any function logic.

8.3 Alert Urgency Levels

Decision: Implement three urgency levels (HIGH, MEDIUM, LOW). **Rationale:** This allows users to prioritize actions. High urgency alerts (frost, extreme heat, critical irrigation) require immediate attention, while low urgency (rain expected) is informational.

8.4 Weather Simulation vs API

Decision: Use simulated weather data instead of real API. **Rationale:** Simulation allows offline functionality, no API costs, and controlled testing. The architecture supports easy replacement with real weather APIs in future.

8.5 Session State for Weather Persistence

Decision: Store weather data in Streamlit session state. **Rationale:** Prevents weather regeneration on every interaction while allowing manual refresh when needed.

---

9. Implementation Details

9.1 Module Structure

```
crop_monitoring_system/
```

```
│
├── crop_dashboard.py      # Main application file
│
├── modules/
│   ├── __init__.py
│   ├── crop_database.py   # CROP_DATABASE dictionary
│   ├── weather_sim.py     # FETCH_WEATHER_DATA_SIMULATED()
│   ├── temp_check.py      # CHECK_TEMPERATURE()
│   └── irrigation.py      # CHECK_IRRIGATION()
│
├── requirements.txt       # Dependencies
├── README.md              # Project documentation
└── statement.md           # Problem statement
```

9.2 Key Functions

FETCH_WEATHER_DATA_SIMULATED():

- Generates random but realistic weather values
- Temperature: 18-38°C range
- Humidity: 40-90%
- Creates 7-day forecast with rain probability

CHECK_TEMPERATURE(CROP, WEATHER):

- Compares current temp against crop's TMIN and TMAX
- Generates frost warnings when temp < TMIN - 5
- Generates heat alerts when temp > TMAX + 5

CHECK_IRRIGATION(CROP, WEATHER):

- Evaluates soil moisture against crop water needs
- Checks 2-day forecast for upcoming rain
- Recommends holding irrigation if rain expected

9.3 Dependencies

```
streamlit>=1.28.0
```

---

10. Screenshots / Results

Dashboard Overview

The main dashboard displays current weather conditions with large, readable metrics and emoji icons for quick comprehension.

7-Day Forecast

A horizontal layout shows the weekly forecast with rain/sun icons and temperature predictions for each day.

Crop Selection

A dropdown menu allows selection from 9 supported crops, each displaying its optimal temperature range and water requirements.

Alert Display

Color-coded alert cards show:

- 🔴 Red for HIGH urgency (frost, extreme heat, urgent irrigation)
- 🟡 Yellow for MEDIUM urgency (low/high temp, irrigation needed)
- 🔵 Blue for LOW urgency (rain expected)
- 🟢 Green for optimal conditions

---

11. Testing Approach

11.1 Unit Testing

Test Case	Input	Expected Output	Status
Frost detection	TEMP=5, TMIN=15	Frost Warning alert	✓
Heat detection	TEMP=45, TMAX=35	Extreme Heat alert	✓
Normal temp	TEMP=20, TMIN=15, TMAX=25	No temp alerts	✓
Low soil moisture	SOIL=10, WATER=30	Urgent Irrigation	✓
Rain expected	SOIL=25, WATER=30, RAIN_SOON=True	Hold Irrigation	✓

11.2 Integration Testing

- Verified all functions work together in Streamlit
- Tested session state persistence across interactions
- Confirmed refresh button generates new weather data

11.3 UI Testing

- Tested on Chrome, Firefox, and Safari
- Verified responsive layout on different screen sizes
- Confirmed all interactive elements function correctly

---

12. Challenges Faced

Challenge 1: Incomplete Original Code

Problem: The initial CHECK_IRRIGATION function was incomplete. **Solution:** Analyzed the logic pattern from CHECK_TEMPERATURE and implemented the missing irrigation alert logic.

Challenge 2: Weather Data Persistence

Problem: Streamlit reruns the entire script on each interaction, regenerating weather data. **Solution:** Implemented session state to store weather data between interactions.

Challenge 3: Alert Priority Display

Problem: Needed to display alerts in a clear, prioritized manner. **Solution:** Used color-coded Streamlit components (st.error, st.warning, st.info, st.success) mapped to urgency levels.

---

13. Learnings & Key Takeaways

1. **Modular Design:** Separating weather simulation, alert logic, and UI made the code maintainable and testable.
2. **State Management:** Learned how Streamlit's session state handles data persistence in a stateless web environment.
3. **User-Centered Design:** Color-coded alerts and emoji icons significantly improve user experience and quick comprehension.
4. **Threshold-Based Systems:** Implementing multi-level thresholds (e.g., TMIN-5, TMIN, TMAX, TMAX+5) provides nuanced alerts rather than binary pass/fail.
5. **Agricultural Domain Knowledge:** Gained understanding of crop-specific requirements and how weather impacts farming decisions.

---

14. Future Enhancements

1. **Real Weather API Integration:** Connect to OpenWeatherMap or similar APIs for actual weather data

based on user location.

2. **Historical Data Analysis:** Store and visualize historical weather patterns and alert frequencies.
3. **Multi-Crop Monitoring:** Allow monitoring multiple crops simultaneously with a combined dashboard view.
4. **Push Notifications:** Implement email/SMS alerts for high-urgency situations.
5. **Machine Learning Predictions:** Use historical data to predict future irrigation needs and potential crop stress.
6. **Mobile Application:** Develop a mobile-friendly version using frameworks like Flutter or React Native.
7. **Database Integration:** Store crop data and user preferences in SQLite or PostgreSQL for persistence.
8. **Localization:** Support multiple languages for broader accessibility.

---

15. References

1. Streamlit Documentation - https://docs.streamlit.io/
2. Python Official Documentation - https://docs.python.org/3/
3. FAO Crop Water Requirements - http://www.fao.org/
4. Agricultural Temperature Guidelines - Various agricultural extension services
5. Weather Simulation Techniques - Python random module documentation

---

End of Report