Intelligent Crop Planning using AI

Harinder Kaur

Department of Computing Studies & Information System, Douglas College

CSIS 4495: Applied Research Project

Dr. Padmapriya Arasanipalai Kandhadai

April 13, 2025

Intelligent Crop Planning using AI

Introduction

Background and Context of Research

Agriculture remains the cornerstone of food security, economic development, and rural livelihood across the globe. In an era where climate variability, resource limitations, and sustainability concerns challenge conventional farming practices, integrating intelligent systems into agriculture has become increasingly vital. Precision agriculture and AI-driven decision support systems offer innovative solutions to optimizing crop selection, improving resource efficiency, and increasing yields.

This project, Intelligent Crop Planning using AI, aims to bridge the gap between real-time environmental conditions and data-driven farming decisions. By leveraging soil and weather data APIs, machine learning models, and dynamic user interaction, the system provides personalized crop recommendations and detailed weekly growth checklist which is generated specific to the geographic and environmental contexts.

Framing the Problem

The following questions are addressed by this research:

1. How can real-time soil and weather data be used to recommend the best crops based on their location's real-time weather and soil data?
2. How can we provide detailed, actionable weekly guidance for growing a chosen crop?
3. Can we create a simple, interactive platform that farmers or planners can use to simulate outcomes and make better planting decisions?

These questions are critically important because traditional crop planning often relies on static, generalized recommendations that do not reflect the constantly changing local conditions.

Literature Review and Knowledge Gaps

Previous research has demonstrated the use of remote sensing, soil analysis, and weather forecasting in precision agriculture. In addition to this, the potential of AI in agriculture—ranging from satellite-based yield prediction models to mobile-based farming assistance has been used before. These studies have explored various machine learning models for crop prediction, including Decision Trees, Random Forests, etc. However, many of these models or studies rely only on historical datasets and do not leverage real-time environmental data for continuous decision-making. Furthermore, existing solutions often require expensive IoT-based soil sensors, making them inaccessible to small-scale farmers. This research aims to bridge this gap by using real-time API-based soil and weather data to provide crop recommendations and without requiring physical sensor installations.

However, there exists a gap in delivering fully automated, real-time, and user-specific crop recommendations with continuous growth monitoring, PDF summaries, and interactive chat support. This project seeks to fill that gap.

Hypotheses, Assumptions, and Potential Benefits

The following are the assumptions made for this project:

1. Soil and weather data from external APIs are reliable and representative of actual field conditions.
2. Users know or can determine their land location.

The following are the potential benefits of this project:

1. Smarter crop choices based on live data help increase yield and efficiency.
2. Weekly water and sunlight recommendations reduce resource overuse and improve sustainability.
3. Interactive tools enable even novice users to make informed crop planning decisions.
4. The system can be expanded to include features like pest alerts, fertilizer tracking, and crop rotation planning.

Summary of the Research Project

This project developed an intelligent, AI-driven crop planning system that provides real-time crop recommendations and actionable growth plans based on live weather and soil data. On the homepage, the system automatically detects the user's location via IP and recommends three optimal crops suited for that region's current conditions, while also offering the flexibility to manually search other locations. Users can also choose both a crop and a location to receive a customized week-by-week crop management plan.

The platform displays one-month trends of soil moisture and temperature using interactive line graphs, enabling users to visualize environmental patterns that influence crop selection. After selecting a crop, users receive weekly plans that include irrigation schedules, temperature and pH targets, fertilizer guidance, and actionable summaries. These are presented through clear visuals and can be downloaded as a PDF.

Built using Flask, the system integrates real-time weather and soil APIs, applies reasoning and machine learning for personalized crop suggestions, and stores user interactions and plans using AWS services. An AI chatbot is also integrated to assist users in generating plans and answering questions. The system is designed to be user-friendly, insightful, and scalable—supporting smart farming decisions for both novice and experienced users.

Changes to the Proposal

1. ***Crop Recommendation Source Shift:*** The original plan was to use historical Kaggle-based datasets to train a machine learning model for crop recommendation. In the final implementation I shifted to using OpenAI's Mistral 7B Instruct model for generating crop recommendations based on real-time weather and soil data.

    Justification: The publicly available datasets were insufficient in both size and regional coverage. Accurate, location-specific crop recommendations required more dynamic and context-aware insights than static datasets could offer. Using OpenAI's model allowed the system to leverage generalized intelligence with real-time environmental inputs for better recommendations.

2. ***Location Detection via IP***: At first the user manually input their location. In the final implementation the project automatically detect the user's location through IP address.

    Justification: This reduced friction in the user experience and allowed the homepage to provide instant recommendations without requiring user input. It also enhanced accessibility for less tech-savvy users.

3. ***Soil Data Collection:*** Initially, only AgroAPI from OpenWeather for weather data soil data was used. Now, additional data is collected with Open-Meteo and for soil data.

    Justification: AgroAPI had limitations in coverage and access. Open-Meteo provided a more robust and open-access source of historical and real-time soil parameters. Selenium was additionally used to extract supplementary data when not available via APIs.

4. ***Removal of Historical Dataset-Based Predictions:*** Originally, I used a dataset to generate a plan. The plan was made on the basis of the data in the columns rather than a machine learning model and it was not dynamic. It is now replaced by AI model

which provides the sowing to harvest time using environmental context, with no dataset-based ML training.

Justification: Due to the lack of high-quality, labelled crop lifecycle datasets, the project could not train a reliable supervised model. Leveraging the chatbot to estimate duration based on real-time environmental data offered a more scalable solution.

5. ***Timeline and Hosting:*** In the proposal, the system was to be deployed to AWS EC2 with complete integration of login, chatbot, and cloud storage. The final implementation is hosted locally. The login/register system is built using DynamoDB, but chat history is not backed up. AWS S3 is used for some storage features.

   Justification: Due to time constraints and integration challenges, full cloud deployment was deprioritized. The focus was shifted to ensuring functional features were working correctly on localhost.
6. ***Displaying Trend Graphs on Homepage:*** In the original proposal, there was no mention of visual soil trend analytics. The homepage, in the final implementation, now includes a one-month trend line graph showing soil temperature and moisture.

   Justification: Adding visualizations allows users to understand environmental patterns more clearly and helps in making informed decisions. It enhances transparency and educational value for users.

Project Completion Timeline

Phase 1: Initial Research, Dataset Exploration, and Planning (Week 4-5)

During this period, the initial machine learning model was developed to recommend crops based on user-specific environmental conditions. This phase also included building the logic to compare user-provided or detected soil and weather conditions to ideal crop parameters.

In parallel, the design of the user interface also started. A preliminary layout of the homepage was constructed, focusing on usability and clarity of inputs and outputs. Since the intended soil data API lacked full global coverage, an alternative method was implemented using a web automation tool to fetch soil data from online sources.

Phase 2: Feature Development and Frontend Integration (Week 5-7)

The second development phase focused on integrating new features into the frontend and expanding the application's functionality. A systematic approach was followed to implement a crop growth duration feature, weekly crop schedules, and recommendations for modifying local soil or weather conditions to support better yields. These features were implemented using a dataset that had all the required conditions.

The frontend, developed using a Python-based web framework, was updated to visually display the generated weekly recommendations. The crop planning page was designed and refined during this phase. Runtime errors caused by missing or inconsistent API responses were identified and resolved. This debugging stage was essential for ensuring that the platform could provide reliable recommendations under varied data conditions. Additionally, some of the elements of the user interface were improved through CSS.

Phase 3: AI Integration and Backend Infrastructure (Week 8-10)

In the third phase, the focus shifted to enhancing the platform's intelligence and interactivity. A conversational chatbot was implemented to answer user queries about agriculture, irrigation, and crop recommendations. The chatbot was powered by Open AI's language model. When usage limitations were encountered with the initial provider, the system was migrated to a different API that allowed for uninterrupted development and improved flexibility.

The chatbot's interface was redesigned to reflect a natural conversation flow, with retained message history and consistent formatting of the bot's responses. Several attempts were made to refine the response output and eliminate formatting issues. Concurrently, user authentication features, including login and registration, were implemented. This was followed by the integration of a scalable cloud-based database solution, where user credentials were saved. An unsuccessful attempt to save chatbot interactions was made. However, the chats were retained for the active session.

Phase 4: Real-time Data Handling and Visualization (Week 11- 13)

In this phase, the system was enhanced with real-time environmental data handling capabilities. The code was configured to automatically detect a user's geographical location using their IP address. This enabled the system to dynamically fetch location-specific soil and weather data, eliminating the need for manual input. A robust weather API was integrated to retrieve soil data for the past 30 days, including soil moisture and temperature trends.

These datasets were processed and visualized using dynamic line graphs on the frontend, allowing users to identify whether their environmental conditions were stable, improving, or degrading. Trend analysis was incorporated to assess the detected changes. This visualization added a layer of transparency and interactivity to the recommendation engine, helping users make informed decisions based on real data trends.

Phase 5: Final Enhancements and Deliverables (Last week)

The final phase involved fine-tuning the intelligence of the system and packaging key outputs for user accessibility. The recommendations were formatted into structured weekly plans, which could be downloaded in document format for offline use. This feature provided a practical output that users could retain for long-term planning and implementation.

The completion of this phase marked the successful integration of all core components, including real-time data collection, machine learning-based prediction, chatbot interactivity, cloud-based user management, and interactive frontend visualizations. The system was then tested for performance and usability to ensure it met the functional and technical objectives defined in the early stages of the project.

Implemented Features

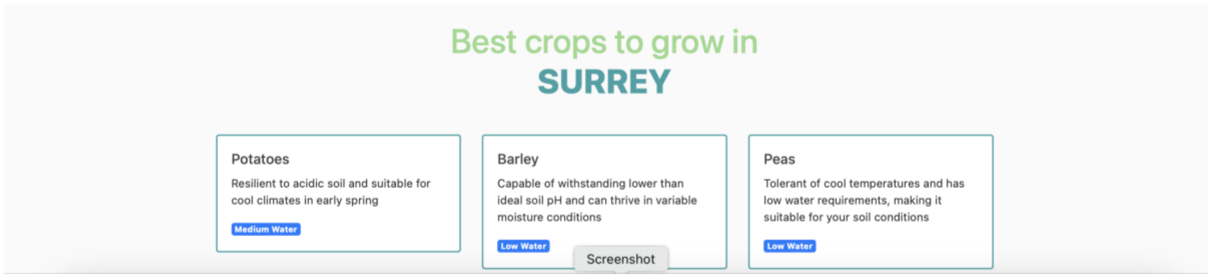
Feature 1: Crop Recommendation System

The crop recommendation system was developed to suggest the best crops for a user to plant based on their local soil and weather conditions. The objective was to provide accurate, data-driven recommendations that adapt to different environments.

Implementation Details:

1. A dataset containing ideal growing conditions for various crops was curated and cleaned.
2. Each user’s soil data (temperature, moisture) and weather parameters (humidity, rainfall, wind) were fetched using APIs and web scraping methods.
3. A filtering algorithm was implemented in Python to match the user's environmental conditions with the optimal requirements of each crop.
4. The final list of recommended crops was generated and displayed on the frontend.

App Screenshots:



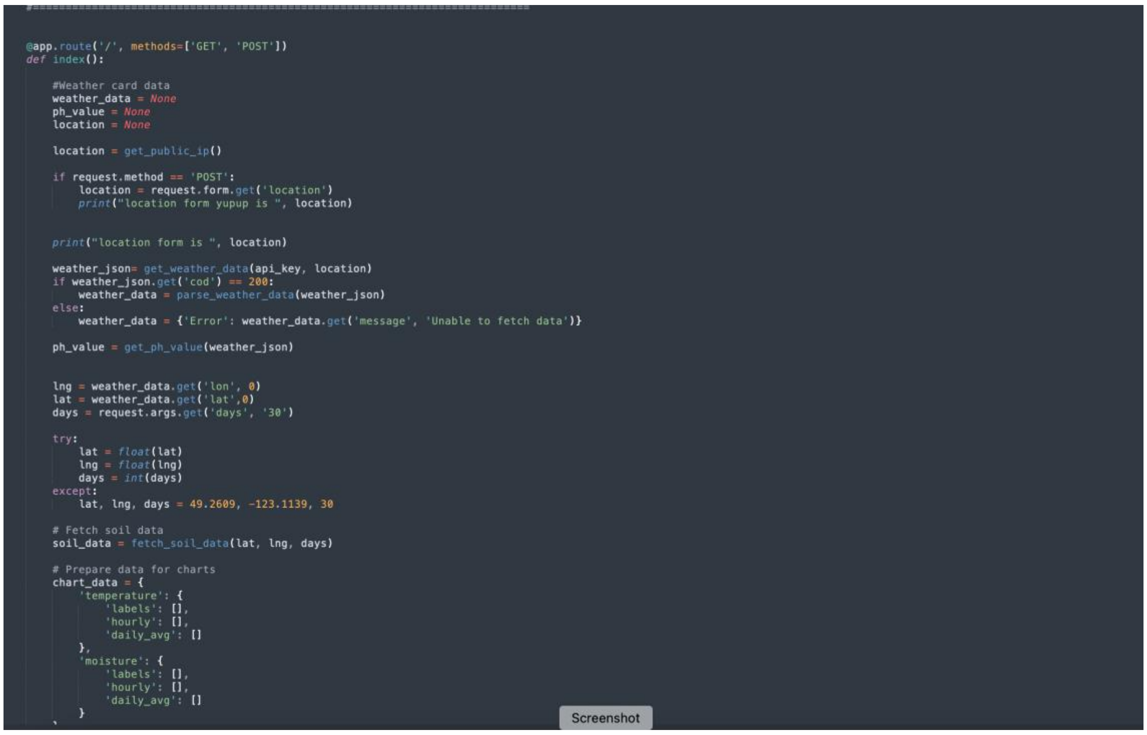
Best crops to grow in
SURREY

Potatoes
Resilient to acidic soil and suitable for cool climates in early spring
Medium Water

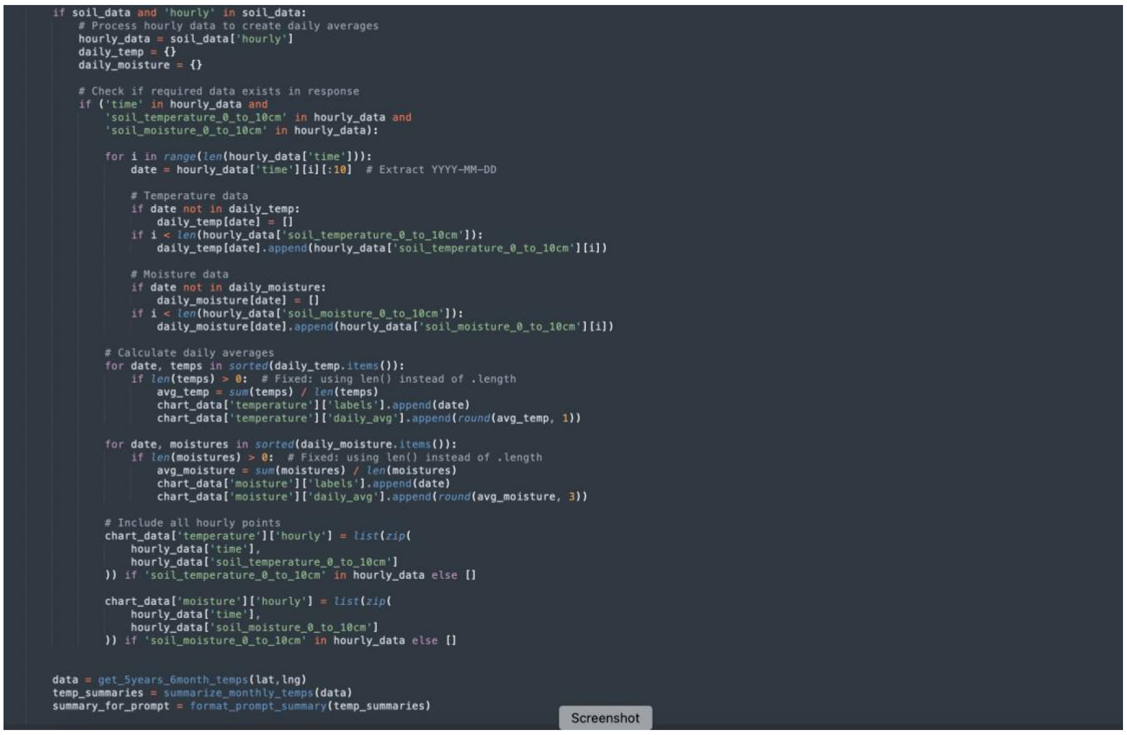
Barley
Capable of withstanding lower than ideal soil pH and can thrive in variable moisture conditions
Low Water

Peas
Tolerant of cool temperatures and has low water requirements, making it suitable for your soil conditions
Low Water

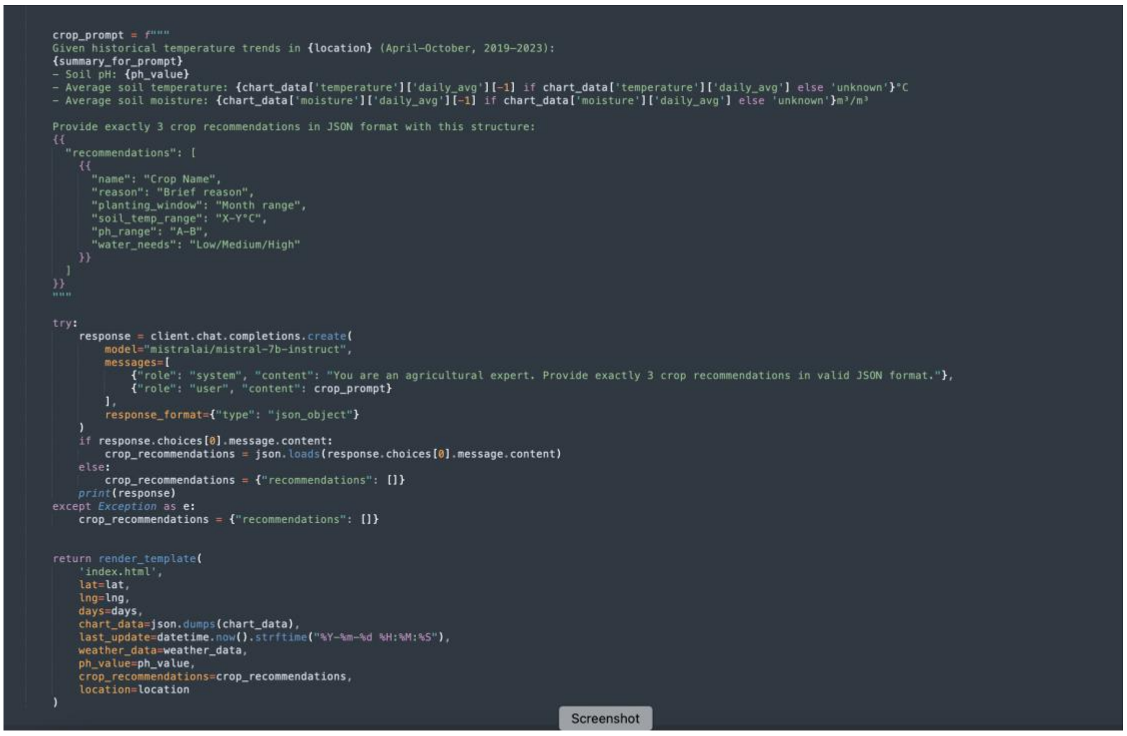
Code Screenshots:



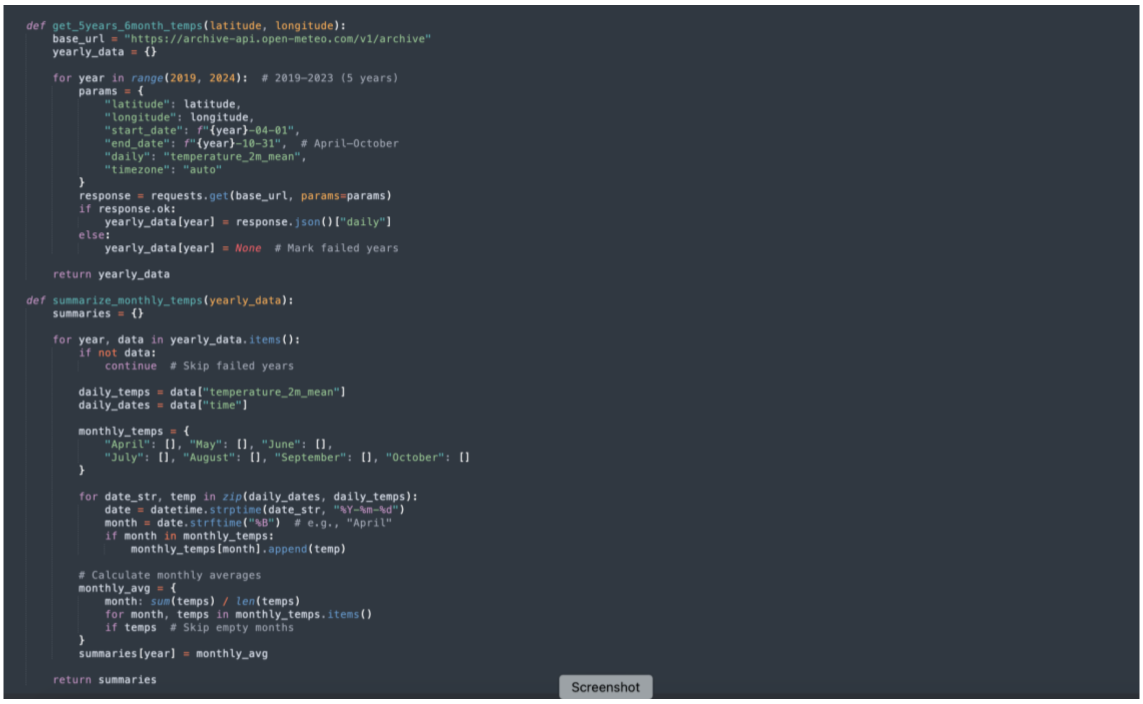
```
#==========================================================================

@app.route('/', methods=['GET', 'POST'])
def index():

    #Weather card data
    weather_data = None
    ph_value = None
    location = None

    location = get_public_ip()

    if request.method == 'POST':
        location = request.form.get('location')
        print("location form yupup is ", location)


    print("location form is ", location)

    weather_json= get_weather_data(api_key, location)
    if weather_json.get('cod') == 200:
        weather_data = parse_weather_data(weather_json)
    else:
        weather_data = {'Error': weather_data.get('message', 'Unable to fetch data')}

    ph_value = get_ph_value(weather_json)


    lng = weather_data.get('lon', 0)
    lat = weather_data.get('lat',0)
    days = request.args.get('days', '30')

    try:
        lat = float(lat)
        lng = float(lng)
        days = int(days)
    except:
        lat, lng, days = 49.2609, -123.1139, 30

    # Fetch soil data
    soil_data = fetch_soil_data(lat, lng, days)

    # Prepare data for charts
    chart_data = {
        'temperature': {
            'labels': [],
            'hourly': [],
            'daily_avg': []
        },
        'moisture': {
            'labels': [],
            'hourly': [],
            'daily_avg': []
        }
```



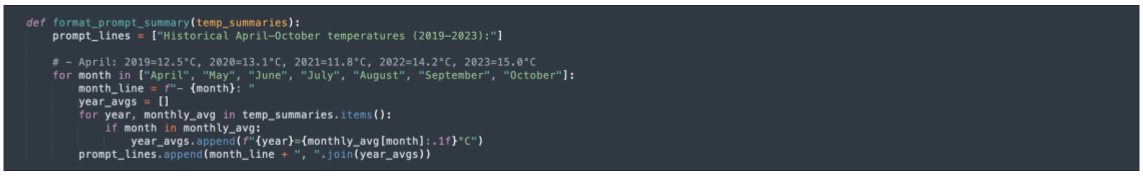
```
if soil_data and 'hourly' in soil_data:
    # Process hourly data to create daily averages
    hourly_data = soil_data['hourly']
    daily_temp = {}
    daily_moisture = {}

    # Check if required data exists in response
    if ('time' in hourly_data and
        'soil_temperature_0_to_10cm' in hourly_data and
        'soil_moisture_0_to_10cm' in hourly_data):

        for i in range(len(hourly_data['time'])):
            date = hourly_data['time'][i][:10]  # Extract YYYY-MM-DD

            # Temperature data
            if date not in daily_temp:
                daily_temp[date] = []
            if i < len(hourly_data['soil_temperature_0_to_10cm']):
                daily_temp[date].append(hourly_data['soil_temperature_0_to_10cm'][i])

            # Moisture data
            if date not in daily_moisture:
                daily_moisture[date] = []
            if i < len(hourly_data['soil_moisture_0_to_10cm']):
                daily_moisture[date].append(hourly_data['soil_moisture_0_to_10cm'][i])

        # Calculate daily averages
        for date, temps in sorted(daily_temp.items()):
            if len(temps) > 0:  # Fixed: using len() instead of .length
                avg_temp = sum(temps) / len(temps)
                chart_data['temperature']['labels'].append(date)
                chart_data['temperature']['daily_avg'].append(round(avg_temp, 1))

        for date, moistures in sorted(daily_moisture.items()):
            if len(moistures) > 0:  # Fixed: using len() instead of .length
                avg_moisture = sum(moistures) / len(moistures)
                chart_data['moisture']['labels'].append(date)
                chart_data['moisture']['daily_avg'].append(round(avg_moisture, 3))

        # Include all hourly points
        chart_data['temperature']['hourly'] = list(zip(
            hourly_data['time'],
            hourly_data['soil_temperature_0_to_10cm']
        )) if 'soil_temperature_0_to_10cm' in hourly_data else []

        chart_data['moisture']['hourly'] = list(zip(
            hourly_data['time'],
            hourly_data['soil_moisture_0_to_10cm']
        )) if 'soil_moisture_0_to_10cm' in hourly_data else []


data = get_5years_6month_temps(lat,lng)
temp_summaries = summarize_monthly_temps(data)
summary_for_prompt = format_prompt_summary(temp_summaries)
```



```
crop_prompt = f"""
Given historical temperature trends in {location} (April–October, 2019–2023):
{summary_for_prompt}
- Soil pH: {ph_value}
- Average soil temperature: {chart_data['temperature']['daily_avg'][-1] if chart_data['temperature']['daily_avg'] else 'unknown'}°C
- Average soil moisture: {chart_data['moisture']['daily_avg'][-1] if chart_data['moisture']['daily_avg'] else 'unknown'}m³/m³

Provide exactly 3 crop recommendations in JSON format with this structure:
{{
  "recommendations": [
    {{
      "name": "Crop Name",
      "reason": "Brief reason",
      "planting_window": "Month range",
      "soil_temp_range": "X-Y°C",
      "ph_range": "A-B",
      "water_needs": "Low/Medium/High"
    }}
  ]
}}
"""

try:
    response = client.chat.completions.create(
        model="mistralai/mistral-7b-instruct",
        messages=[
            {"role": "system", "content": "You are an agricultural expert. Provide exactly 3 crop recommendations in valid JSON format."},
            {"role": "user", "content": crop_prompt}
        ],
        response_format={"type": "json_object"}
    )
    if response.choices[0].message.content:
        crop_recommendations = json.loads(response.choices[0].message.content)
    else:
        crop_recommendations = {"recommendations": []}
    print(response)
except Exception as e:
    crop_recommendations = {"recommendations": []}


return render_template(
    'index.html',
    lat=lat,
    lng=lng,
    days=days,
    chart_data=json.dumps(chart_data),
    last_update=datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
    weather_data=weather_data,
    ph_value=ph_value,
    crop_recommendations=crop_recommendations,
    location=location
)
```



```python
def get_5years_6month_temps(latitude, longitude):
    base_url = "https://archive-api.open-meteo.com/v1/archive"
    yearly_data = {}

    for year in range(2019, 2024):  # 2019–2023 (5 years)
        params = {
            "latitude": latitude,
            "longitude": longitude,
            "start_date": f"{year}-04-01",
            "end_date": f"{year}-10-31",  # April–October
            "daily": "temperature_2m_mean",
            "timezone": "auto"
        }
        response = requests.get(base_url, params=params)
        if response.ok:
            yearly_data[year] = response.json()["daily"]
        else:
            yearly_data[year] = None  # Mark failed years

    return yearly_data

def summarize_monthly_temps(yearly_data):
    summaries = {}

    for year, data in yearly_data.items():
        if not data:
            continue  # Skip failed years

        daily_temps = data["temperature_2m_mean"]
        daily_dates = data["time"]

        monthly_temps = {
            "April": [], "May": [], "June": [],
            "July": [], "August": [], "September": [], "October": []
        }

        for date_str, temp in zip(daily_dates, daily_temps):
            date = datetime.strptime(date_str, "%Y-%m-%d")
            month = date.strftime("%B")  # e.g., "April"
            if month in monthly_temps:
                monthly_temps[month].append(temp)

        # Calculate monthly averages
        monthly_avg = {
            month: sum(temps) / len(temps)
            for month, temps in monthly_temps.items()
            if temps  # Skip empty months
        }
        summaries[year] = monthly_avg

    return summaries
```



```python
def format_prompt_summary(temp_summaries):
    prompt_lines = ["Historical April–October temperatures (2019–2023):"]

    # - April: 2019=12.5°C, 2020=13.1°C, 2021=11.8°C, 2022=14.2°C, 2023=15.0°C
    for month in ["April", "May", "June", "July", "August", "September", "October"]:
        month_line = f"- {month}: "
        year_avgs = []
        for year, monthly_avg in temp_summaries.items():
            if month in monthly_avg:
                year_avgs.append(f"{year}={monthly_avg[month]:.1f}°C")
        prompt_lines.append(month_line + ", ".join(year_avgs))
```

Feature 2: Weekly Crop Schedule and Growth Timeline

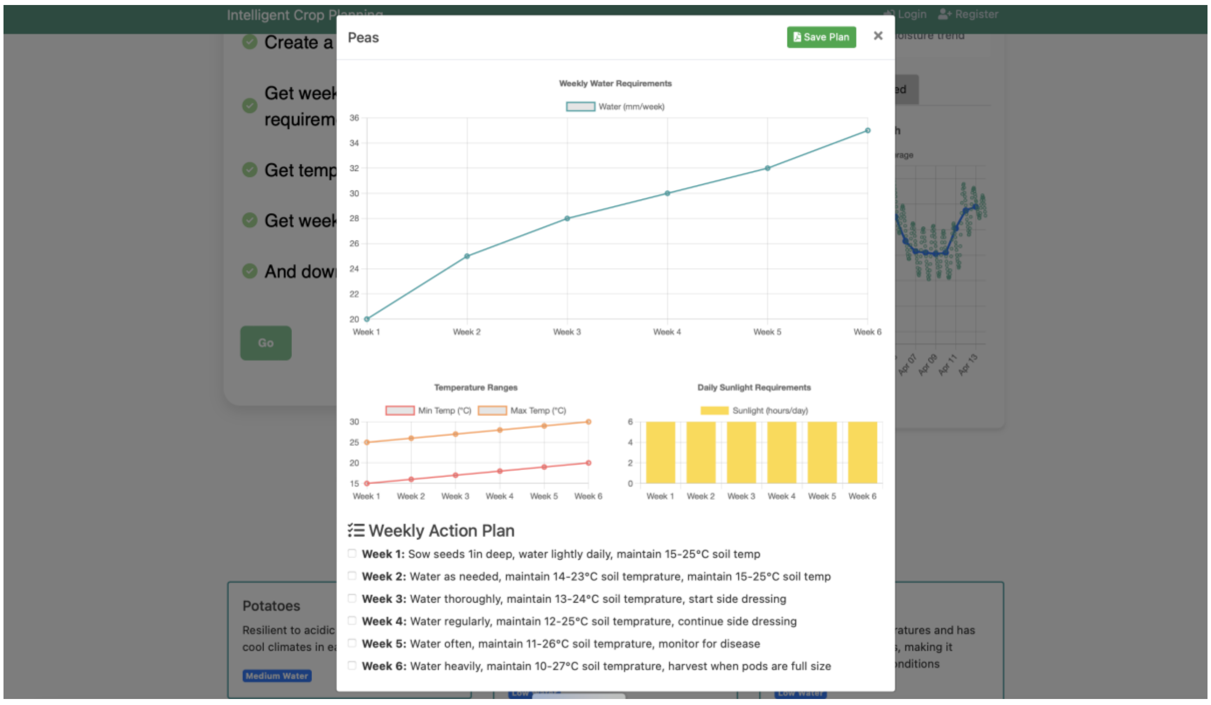
This feature aimed to assist users by providing them with a weekly plan for the crops they choose to plant. It includes ideal growth durations, temperature, water and sunlight requierements, and irrigation or management tips for each stage.

Implementation Details:

1. A crop growth timeline dataset was created, specifying the duration and care instructions for different crops.
2. Based on the recommended crops and the current date, the system generated a weekly schedule leading up to harvesting.

3. Each week contained specific instructions related to watering, temperatures tolerated and sunlight hours.
4. The schedule was formatted in tabular form and rendered using HTML and Bootstrap for visual clarity.

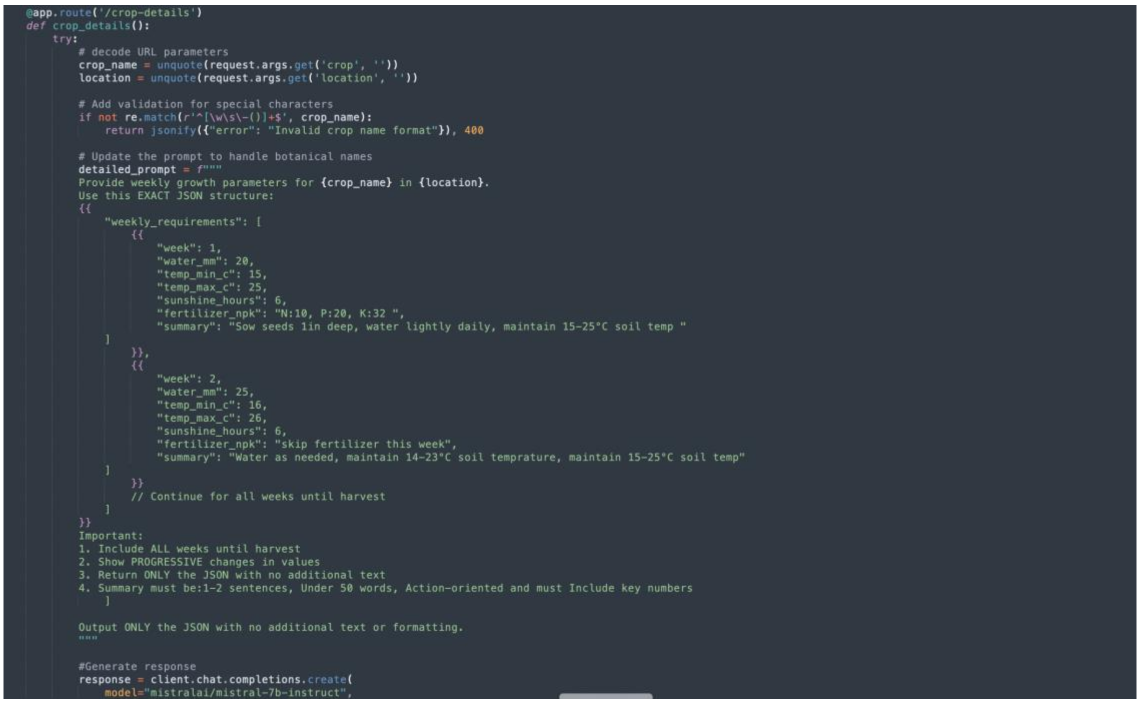
App Screenshots:



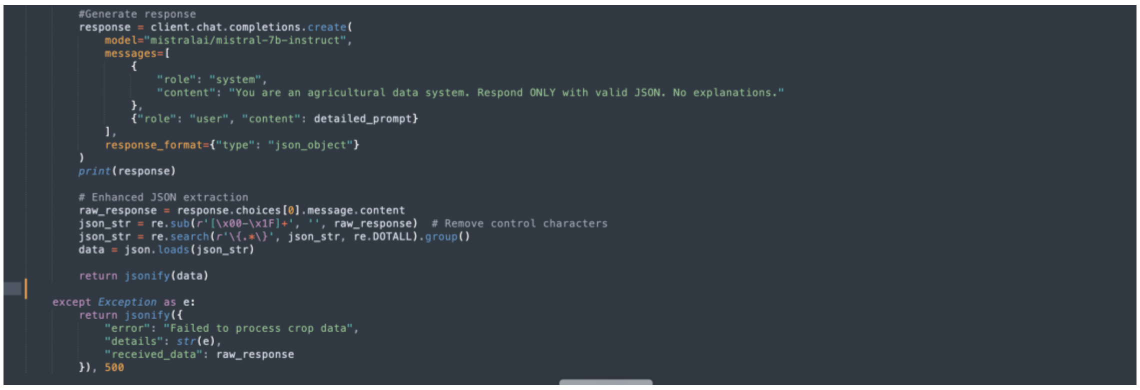
Weekly Action Plan

- ☐ **Week 1:** Sow seeds 1in deep, water lightly daily, maintain 15-25°C soil temp
- ☐ **Week 2:** Water as needed, maintain 14-23°C soil temprature, maintain 15-25°C soil temp
- ☐ **Week 3:** Water thoroughly, maintain 13-24°C soil temprature, start side dressing
- ☐ **Week 4:** Water regularly, maintain 12-25°C soil temprature, continue side dressing
- ☐ **Week 5:** Water often, maintain 11-26°C soil temprature, monitor for disease
- ☐ **Week 6:** Water heavily, maintain 10-27°C soil temprature, harvest when pods are full size

Code Screenshots:



```python
@app.route('/crop-details')
def crop_details():
    try:
        # decode URL parameters
        crop_name = unquote(request.args.get('crop', ''))
        location = unquote(request.args.get('location', ''))

        # Add validation for special characters
        if not re.match(r'^[\w\s\-()]+$', crop_name):
            return jsonify({"error": "Invalid crop name format"}), 400

        # Update the prompt to handle botanical names
        detailed_prompt = f"""
        Provide weekly growth parameters for {crop_name} in {location}.
        Use this EXACT JSON structure:
        {{
            "weekly_requirements": [
                {{
                    "week": 1,
                    "water_mm": 20,
                    "temp_min_c": 15,
                    "temp_max_c": 25,
                    "sunshine_hours": 6,
                    "fertilizer_npk": "N:10, P:20, K:32 ",
                    "summary": "Sow seeds 1in deep, water lightly daily, maintain 15-25°C soil temp "
            ]
                }},
                {{
                    "week": 2,
                    "water_mm": 25,
                    "temp_min_c": 16,
                    "temp_max_c": 26,
                    "sunshine_hours": 6,
                    "fertilizer_npk": "skip fertilizer this week",
                    "summary": "Water as needed, maintain 14-23°C soil temprature, maintain 15-25°C soil temp"
            ]
                }}
                // Continue for all weeks until harvest
            ]
        }}
        Important:
        1. Include ALL weeks until harvest
        2. Show PROGRESSIVE changes in values
        3. Return ONLY the JSON with no additional text
        4. Summary must be:1-2 sentences, Under 50 words, Action-oriented and must Include key numbers
            ]

        Output ONLY the JSON with no additional text or formatting.
        """

        #Generate response
        response = client.chat.completions.create(
            model="mistralai/mistral-7b-instruct",
```



```python
        #Generate response
        response = client.chat.completions.create(
            model="mistralai/mistral-7b-instruct",
            messages=[
                {
                    "role": "system",
                    "content": "You are an agricultural data system. Respond ONLY with valid JSON. No explanations."
                },
                {"role": "user", "content": detailed_prompt}
            ],
            response_format={"type": "json_object"}
        )
        print(response)

        # Enhanced JSON extraction
        raw_response = response.choices[0].message.content
        json_str = re.sub(r'[\x00-\x1F]+', '', raw_response)  # Remove control characters
        json_str = re.search(r'\{.*\}', json_str, re.DOTALL).group()
        data = json.loads(json_str)

        return jsonify(data)

    except Exception as e:
        return jsonify({
            "error": "Failed to process crop data",
            "details": str(e),
            "received_data": raw_response
        }), 500
```

Feature 3: Chatbot for Agricultural Assistance

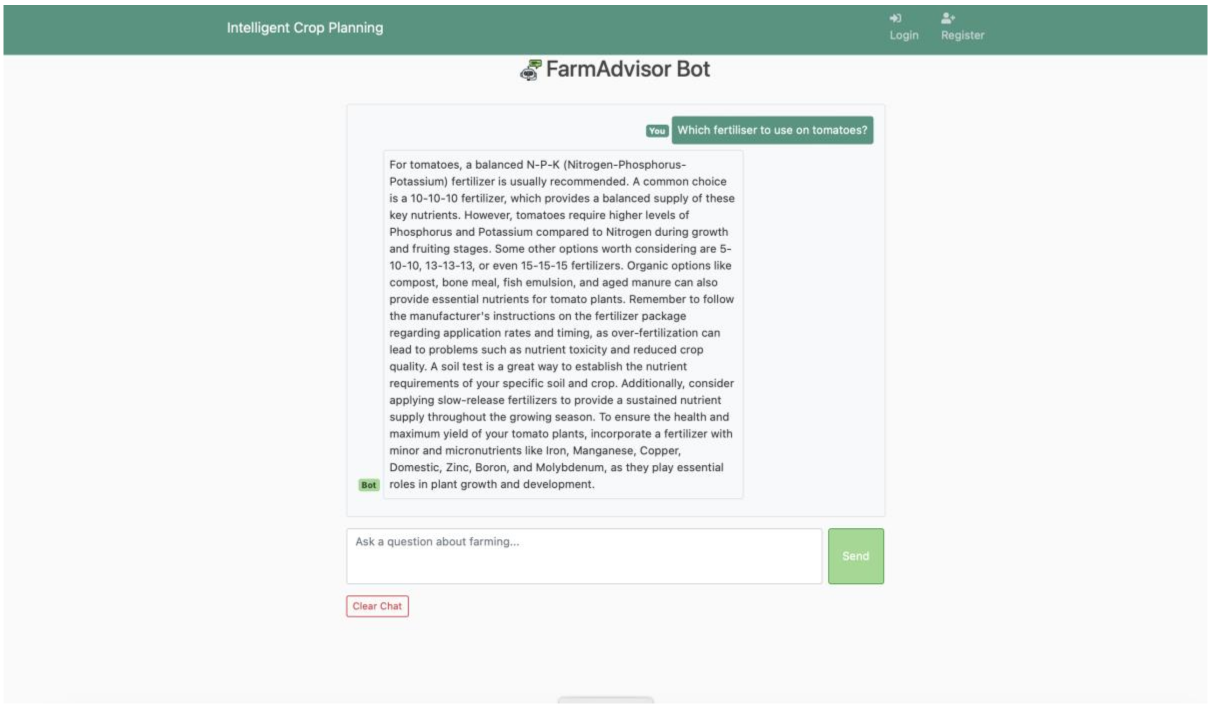
To enhance user engagement and provide dynamic responses to agricultural queries, a chatbot was developed using a language model API. The chatbot was designed to simulate a natural conversational flow, allowing users to ask about irrigation, crops, soil, and more.

Implementation Details:

1. Initially integrated using OpenAI's API, the chatbot was later migrated to OpenRouter API for extended usage.

2. The chatbot was designed using a Flask backend, and its responses were rendered in a chat-style interface on the frontend.
3. Chat history was retained in AWS DynamoDB, allowing users to revisit previous interactions.
4. The interface was developed using Bootstrap, and JavaScript was used to dynamically update the chat window.

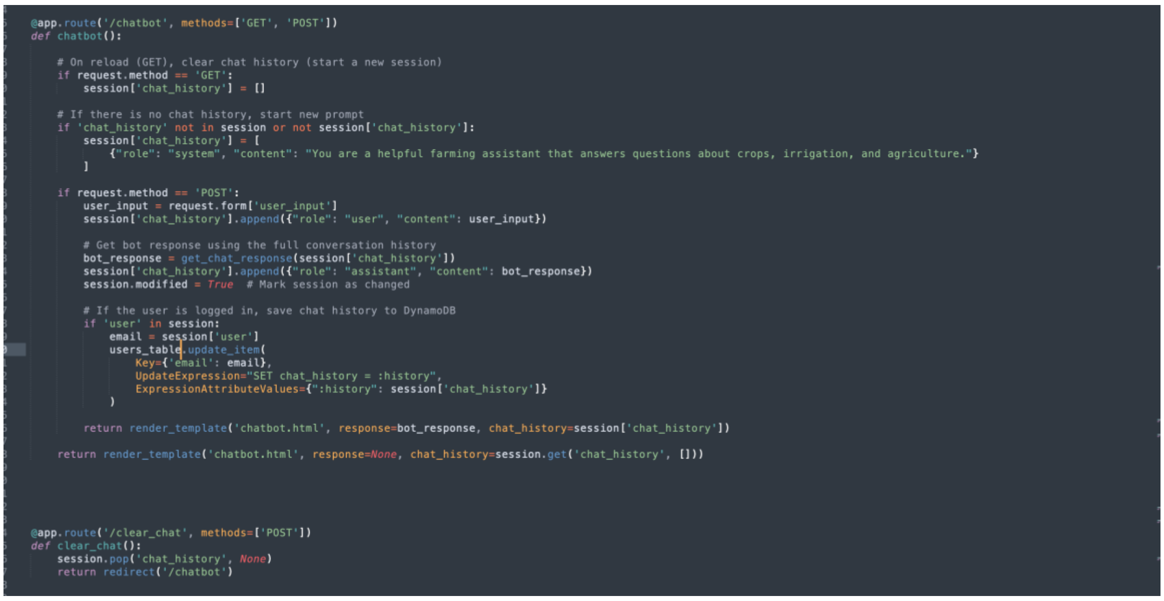
App Screenshots:



Which fertiliser to use on tomatoes?

For tomatoes, a balanced N-P-K (Nitrogen-Phosphorus-Potassium) fertilizer is usually recommended. A common choice is a 10-10-10 fertilizer, which provides a balanced supply of these key nutrients. However, tomatoes require higher levels of Phosphorus and Potassium compared to Nitrogen during growth and fruiting stages. Some other options worth considering are 5-10-10, 13-13-13, or even 15-15-15 fertilizers. Organic options like compost, bone meal, fish emulsion, and aged manure can also provide essential nutrients for tomato plants. Remember to follow the manufacturer's instructions on the fertilizer package regarding application rates and timing, as over-fertilization can lead to problems such as nutrient toxicity and reduced crop quality. A soil test is a great way to establish the nutrient requirements of your specific soil and crop. Additionally, consider applying slow-release fertilizers to provide a sustained nutrient supply throughout the growing season. To ensure the health and maximum yield of your tomato plants, incorporate a fertilizer with minor and micronutrients like Iron, Manganese, Copper, Domestic, Zinc, Boron, and Molybdenum, as they play essential roles in plant growth and development.

Code Screenshots:



```python
@app.route('/chatbot', methods=['GET', 'POST'])
def chatbot():

    # On reload (GET), clear chat history (start a new session)
    if request.method == 'GET':
        session['chat_history'] = []

    # If there is no chat history, start new prompt
    if 'chat_history' not in session or not session['chat_history']:
        session['chat_history'] = [
            {"role": "system", "content": "You are a helpful farming assistant that answers questions about crops, irrigation, and agriculture."}
        ]

    if request.method == 'POST':
        user_input = request.form['user_input']
        session['chat_history'].append({"role": "user", "content": user_input})

        # Get bot response using the full conversation history
        bot_response = get_chat_response(session['chat_history'])
        session['chat_history'].append({"role": "assistant", "content": bot_response})
        session.modified = True  # Mark session as changed

        # If the user is logged in, save chat history to DynamoDB
        if 'user' in session:
            email = session['user']
            users_table.update_item(
                Key={'email': email},
                UpdateExpression="SET chat_history = :history",
                ExpressionAttributeValues={":history": session['chat_history']}
            )

        return render_template('chatbot.html', response=bot_response, chat_history=session['chat_history'])

    return render_template('chatbot.html', response=None, chat_history=session.get('chat_history', []))



@app.route('/clear_chat', methods=['POST'])
def clear_chat():
    session.pop('chat_history', None)
    return redirect('/chatbot')
```

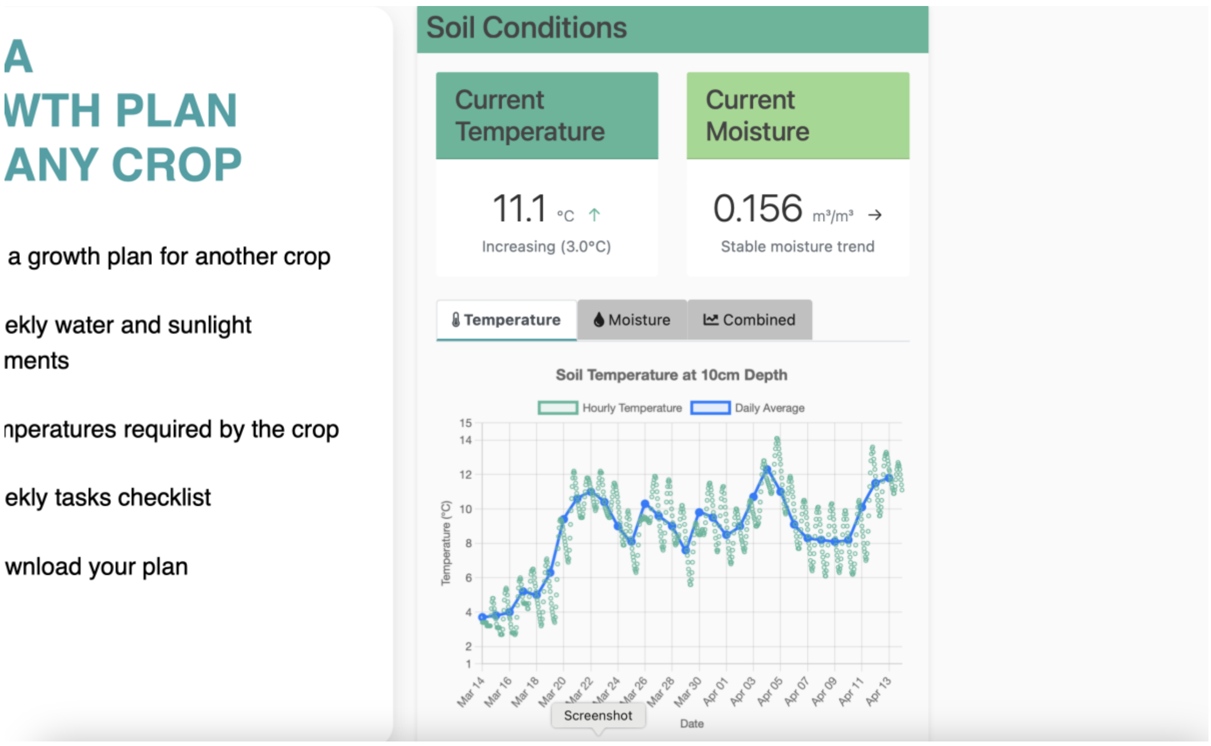
Feature 4: Real-Time Soil Condition Trend Analysis

This feature was designed to analyse and visualize soil condition trends over a 30-day period, allowing users to understand if their land was improving, degrading, or stable in terms of moisture and temperature.

Implementation Details:

1. The Open-Meteo API was used to collect soil data for the past 30 days based on the user's location.
2. JavaScript and Chart.js were used to create dynamic trend line graphs for soil moisture and temperature.
3. Calculated whether the trends were increasing, decreasing, or stable using regression techniques.
4. The results of this analysis were summarized in the dashboard and linked with the crop recommendations for context.

App Screenshots:



A
WTH PLAN
ANY CROP

a growth plan for another crop

ekly water and sunlight
ments

nperatures required by the crop

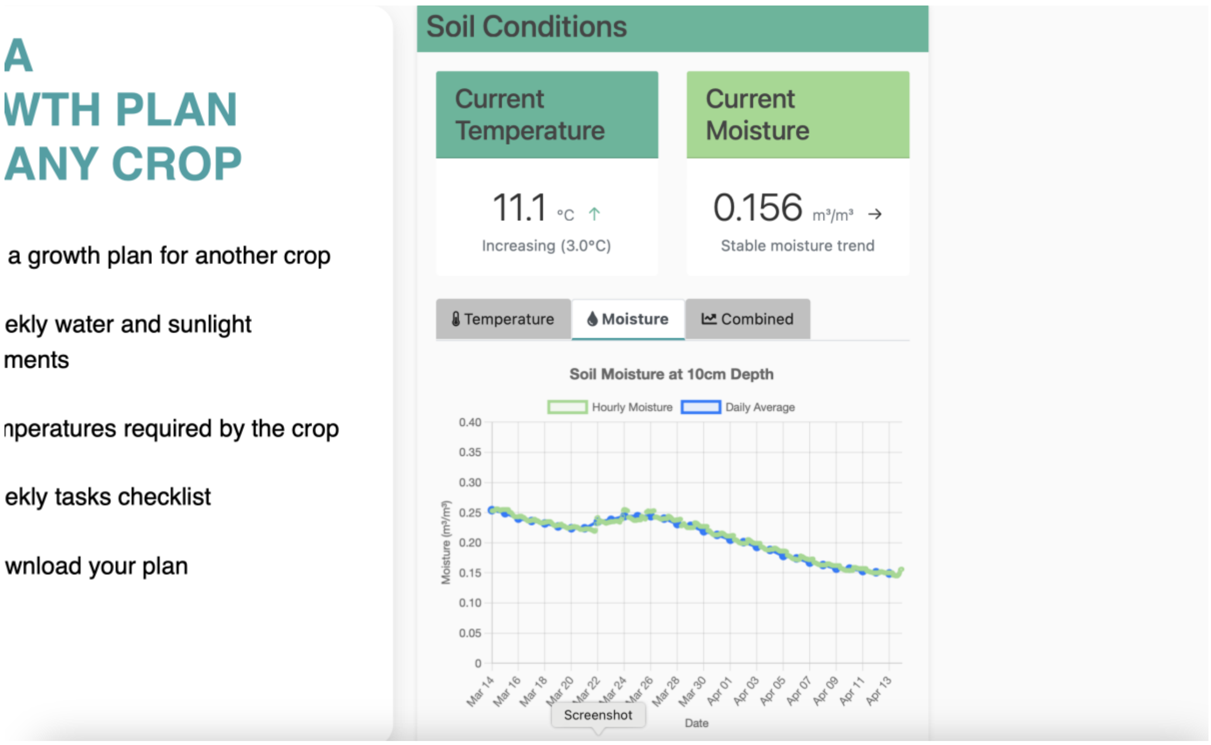
ekly tasks checklist

wnload your plan

Soil Conditions

Current Temperature
11.1 °C ↑
Increasing (3.0°C)

Current Moisture
0.156 m³/m³ →
Stable moisture trend



A
WTH PLAN
ANY CROP

a growth plan for another crop

ekly water and sunlight
ments

nperatures required by the crop

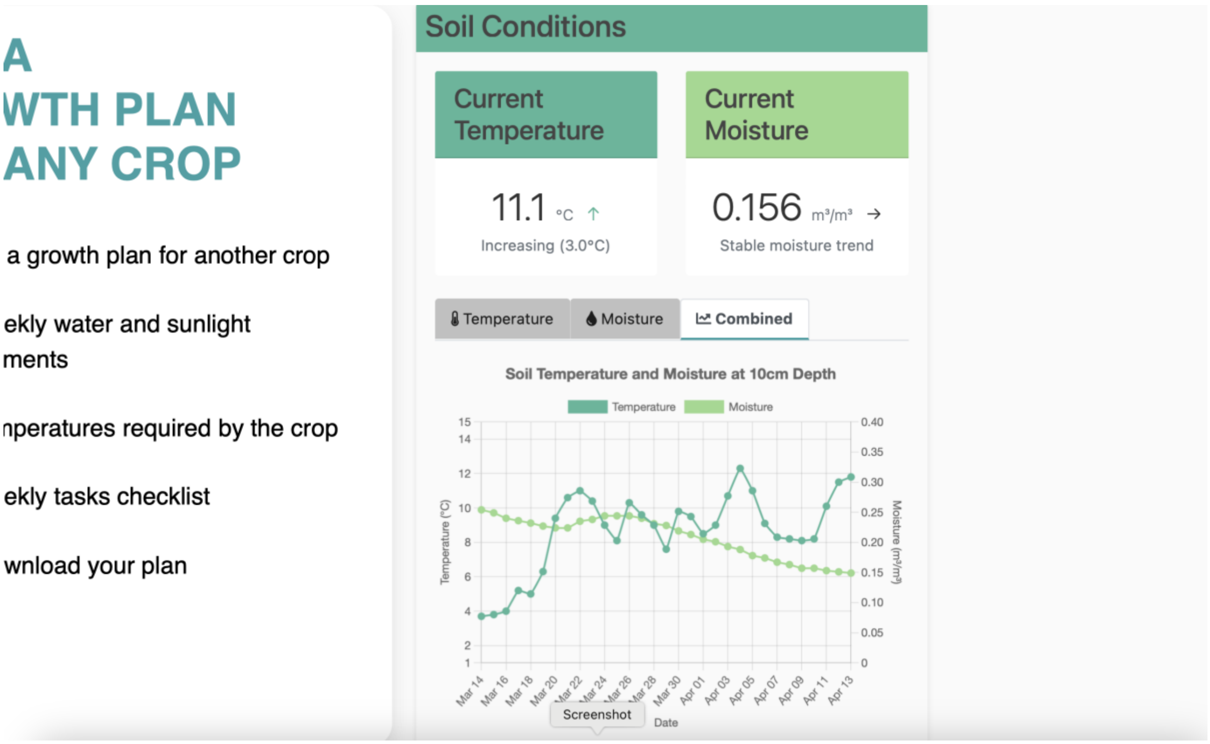
ekly tasks checklist

wnload your plan

Soil Conditions

Current Temperature
11.1 °C ↑
Increasing (3.0°C)

Current Moisture
0.156 m³/m³ →
Stable moisture trend



A
WTH PLAN
ANY CROP

a growth plan for another crop

ekly water and sunlight
ments

nperatures required by the crop

ekly tasks checklist

wnload your plan

Soil Conditions

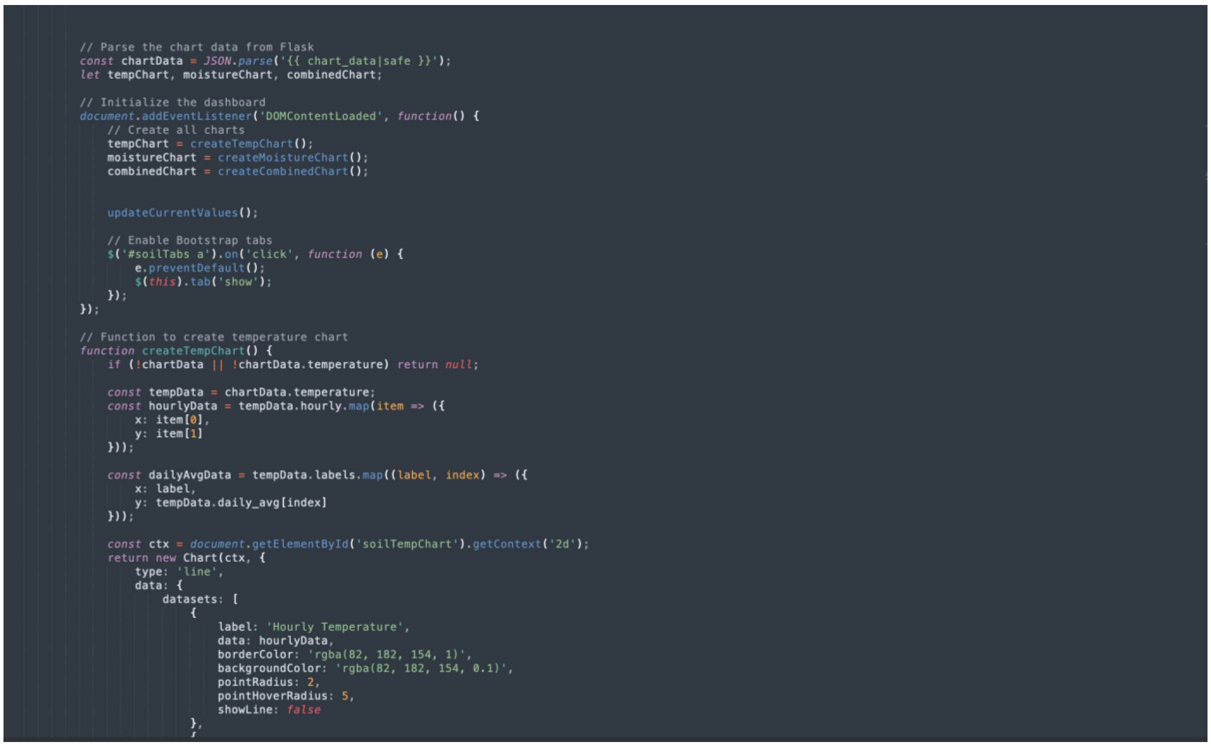
Current Temperature

11.1 °C ↑
Increasing (3.0°C)

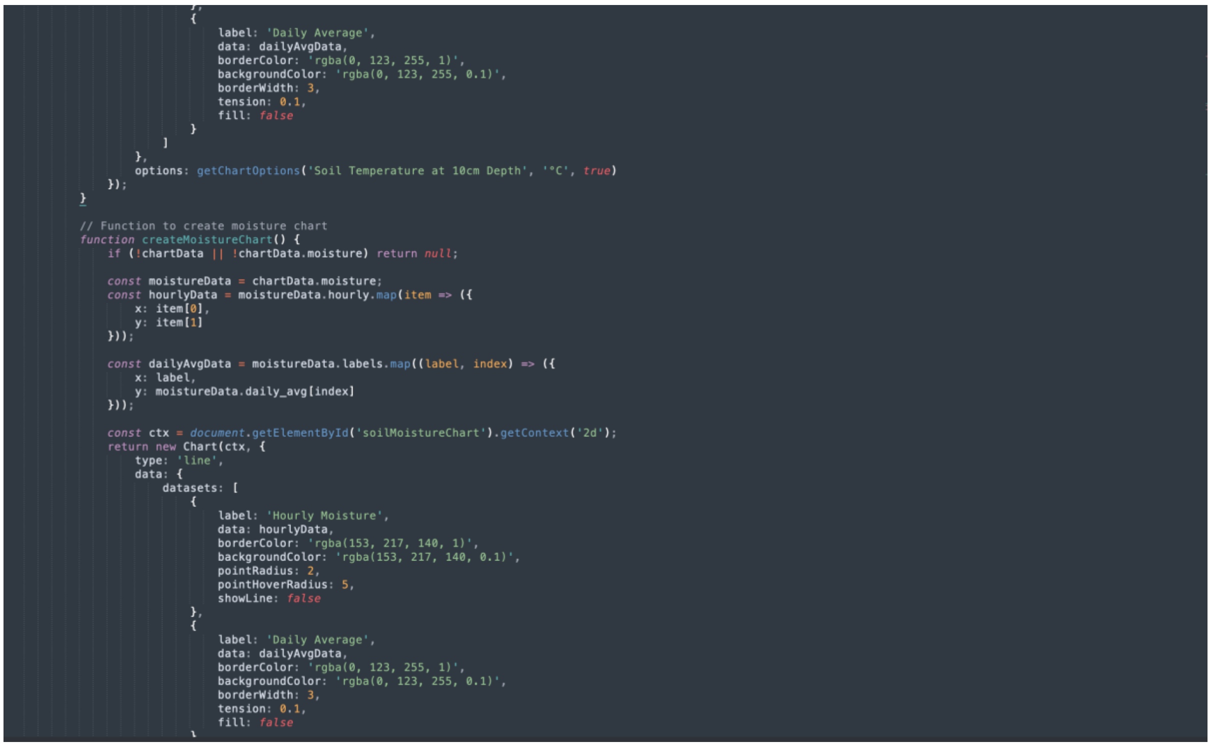
Current Moisture

0.156 m³/m³ →
Stable moisture trend

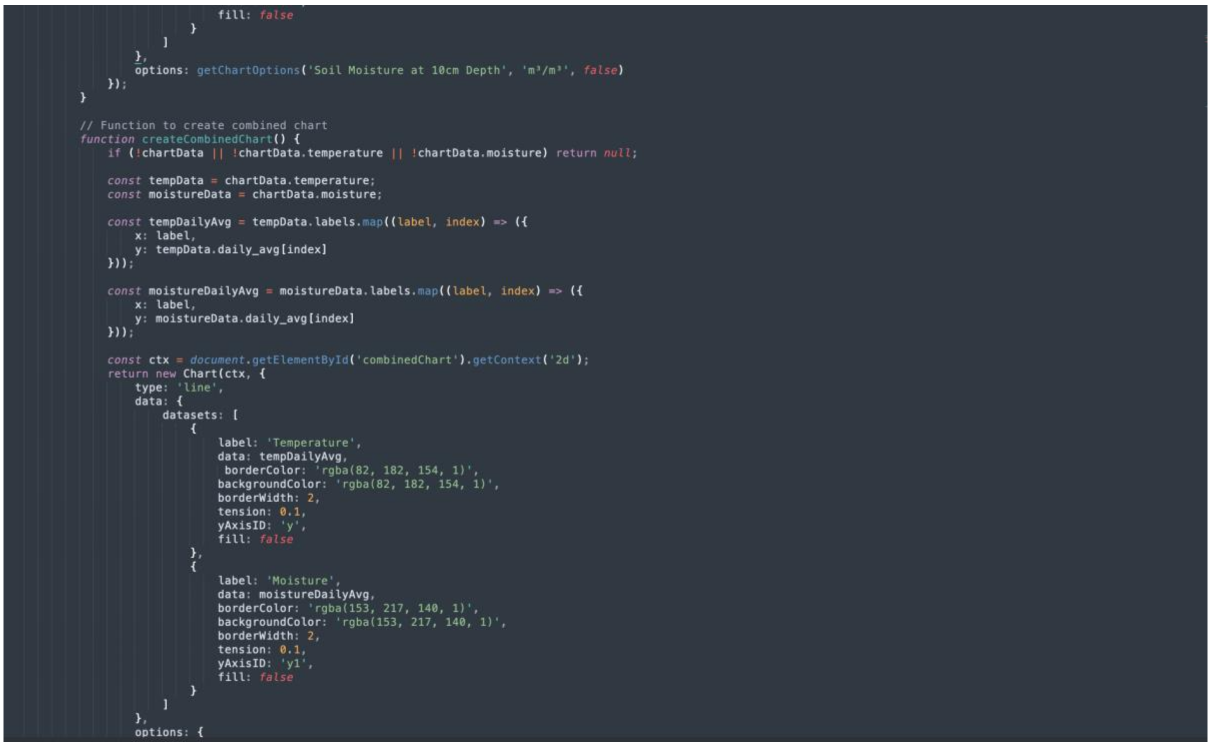
Code Screenshots:



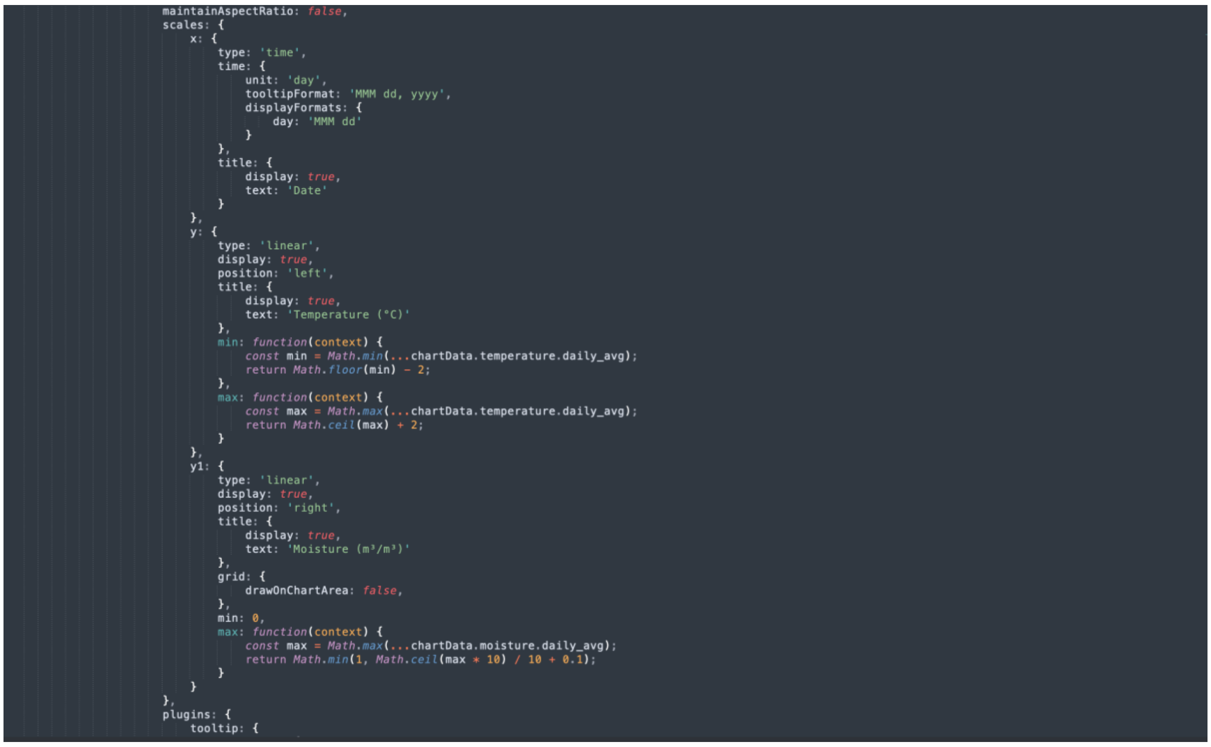
```javascript
// Parse the chart data from Flask
const chartData = JSON.parse('{{ chart_data|safe }}');
let tempChart, moistureChart, combinedChart;

// Initialize the dashboard
document.addEventListener('DOMContentLoaded', function() {
    // Create all charts
    tempChart = createTempChart();
    moistureChart = createMoistureChart();
    combinedChart = createCombinedChart();


    updateCurrentValues();

    // Enable Bootstrap tabs
    $('#soilTabs a').on('click', function (e) {
        e.preventDefault();
        $(this).tab('show');
    });
});

// Function to create temperature chart
function createTempChart() {
    if (!chartData || !chartData.temperature) return null;

    const tempData = chartData.temperature;
    const hourlyData = tempData.hourly.map(item => ({
        x: item[0],
        y: item[1]
    }));

    const dailyAvgData = tempData.labels.map((label, index) => ({
        x: label,
        y: tempData.daily_avg[index]
    }));

    const ctx = document.getElementById('soilTempChart').getContext('2d');
    return new Chart(ctx, {
        type: 'line',
        data: {
            datasets: [
                {
                    label: 'Hourly Temperature',
                    data: hourlyData,
                    borderColor: 'rgba(82, 182, 154, 1)',
                    backgroundColor: 'rgba(82, 182, 154, 0.1)',
                    pointRadius: 2,
                    pointHoverRadius: 5,
                    showLine: false
                },
```



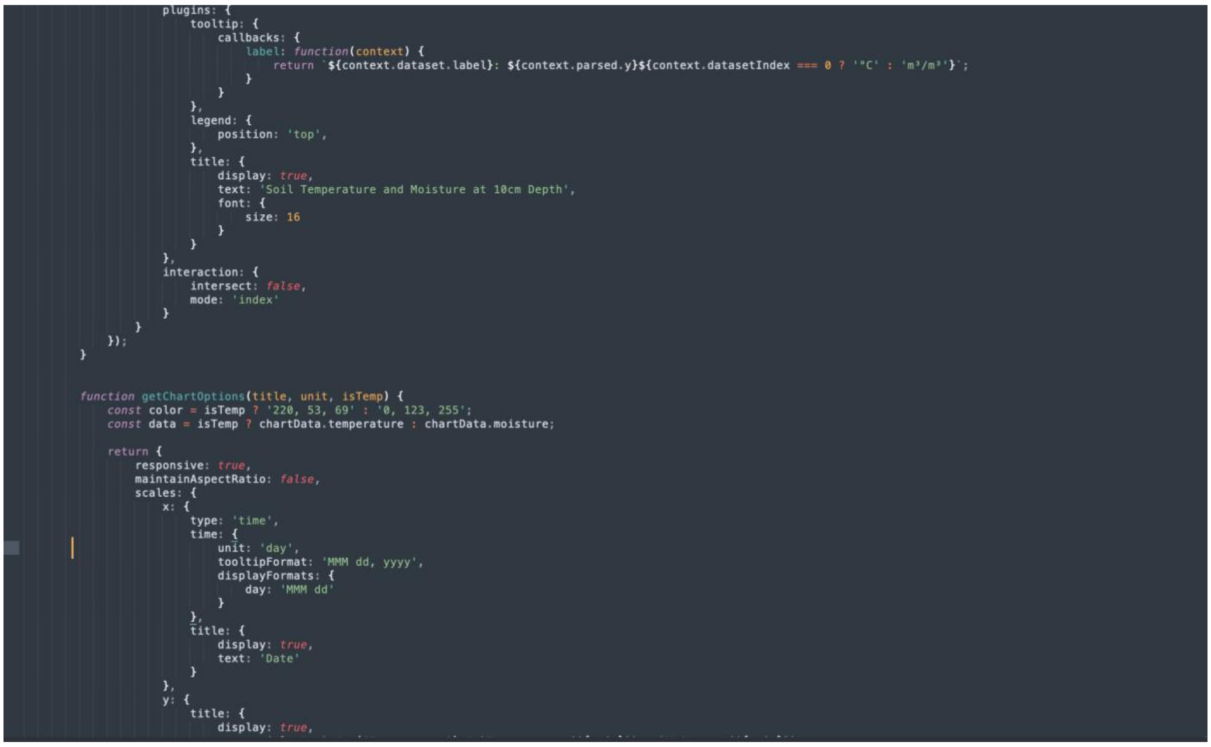
```javascript
            {
                label: 'Daily Average',
                data: dailyAvgData,
                borderColor: 'rgba(0, 123, 255, 1)',
                backgroundColor: 'rgba(0, 123, 255, 0.1)',
                borderWidth: 3,
                tension: 0.1,
                fill: false
            }
        ]
    },
    options: getChartOptions('Soil Temperature at 10cm Depth', '°C', true)
});
}

// Function to create moisture chart
function createMoistureChart() {
    if (!chartData || !chartData.moisture) return null;

    const moistureData = chartData.moisture;
    const hourlyData = moistureData.hourly.map(item => ({
        x: item[0],
        y: item[1]
    }));

    const dailyAvgData = moistureData.labels.map((label, index) => ({
        x: label,
        y: moistureData.daily_avg[index]
    }));

    const ctx = document.getElementById('soilMoistureChart').getContext('2d');
    return new Chart(ctx, {
        type: 'line',
        data: {
            datasets: [
                {
                    label: 'Hourly Moisture',
                    data: hourlyData,
                    borderColor: 'rgba(153, 217, 140, 1)',
                    backgroundColor: 'rgba(153, 217, 140, 0.1)',
                    pointRadius: 2,
                    pointHoverRadius: 5,
                    showLine: false
                },
                {
                    label: 'Daily Average',
                    data: dailyAvgData,
                    borderColor: 'rgba(0, 123, 255, 1)',
                    backgroundColor: 'rgba(0, 123, 255, 0.1)',
                    borderWidth: 3,
                    tension: 0.1,
                    fill: false
```



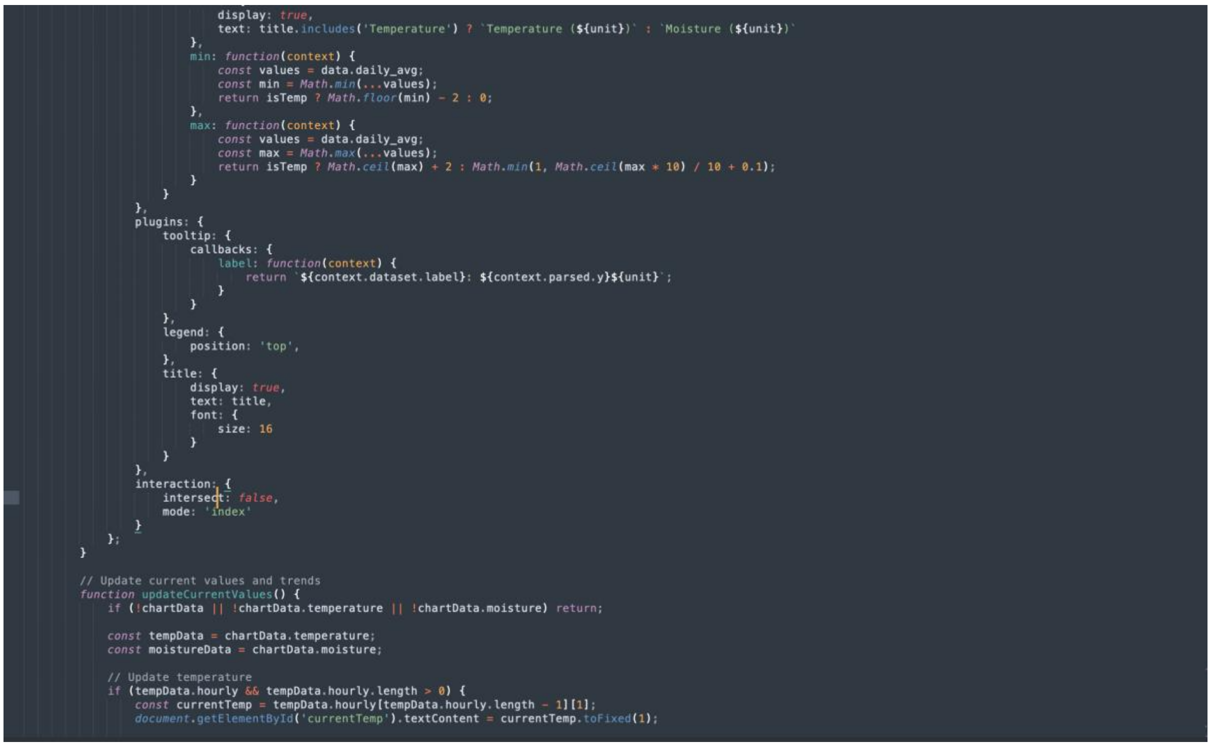
```javascript
                    fill: false
                }
            ]
        },
        options: getChartOptions('Soil Moisture at 10cm Depth', 'm³/m³', false)
    });
}

// Function to create combined chart
function createCombinedChart() {
    if (!chartData || !chartData.temperature || !chartData.moisture) return null;

    const tempData = chartData.temperature;
    const moistureData = chartData.moisture;

    const tempDailyAvg = tempData.labels.map((label, index) => ({
        x: label,
        y: tempData.daily_avg[index]
    }));

    const moistureDailyAvg = moistureData.labels.map((label, index) => ({
        x: label,
        y: moistureData.daily_avg[index]
    }));

    const ctx = document.getElementById('combinedChart').getContext('2d');
    return new Chart(ctx, {
        type: 'line',
        data: {
            datasets: [
                {
                    label: 'Temperature',
                    data: tempDailyAvg,
                     borderColor: 'rgba(82, 182, 154, 1)',
                    backgroundColor: 'rgba(82, 182, 154, 1)',
                    borderWidth: 2,
                    tension: 0.1,
                    yAxisID: 'y',
                    fill: false
                },
                {
                    label: 'Moisture',
                    data: moistureDailyAvg,
                    borderColor: 'rgba(153, 217, 140, 1)',
                    backgroundColor: 'rgba(153, 217, 140, 1)',
                    borderWidth: 2,
                    tension: 0.1,
                    yAxisID: 'y1',
                    fill: false
                }
            ]
        },
        options: {
```



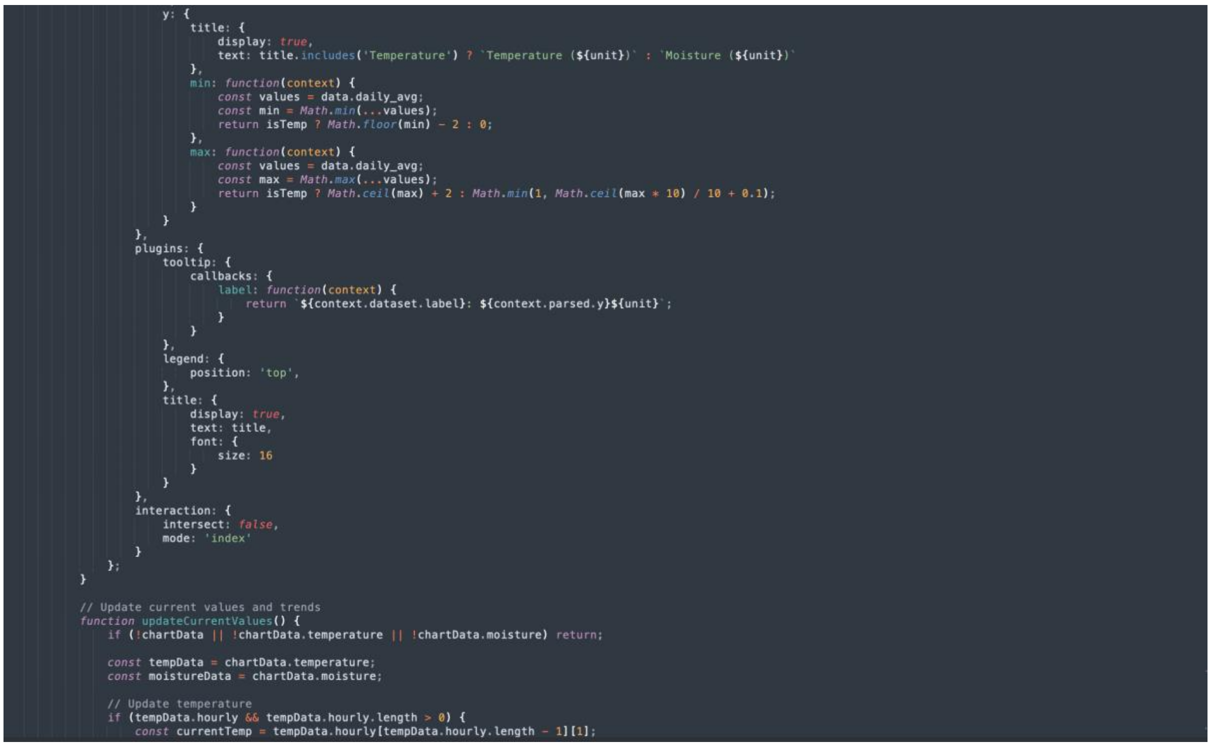
```
            maintainAspectRatio: false,
            scales: {
                x: {
                    type: 'time',
                    time: {
                        unit: 'day',
                        tooltipFormat: 'MMM dd, yyyy',
                        displayFormats: {
                            day: 'MMM dd'
                        }
                    },
                    title: {
                        display: true,
                        text: 'Date'
                    }
                },
                y: {
                    type: 'linear',
                    display: true,
                    position: 'left',
                    title: {
                        display: true,
                        text: 'Temperature (°C)'
                    },
                    min: function(context) {
                        const min = Math.min(...chartData.temperature.daily_avg);
                        return Math.floor(min) - 2;
                    },
                    max: function(context) {
                        const max = Math.max(...chartData.temperature.daily_avg);
                        return Math.ceil(max) + 2;
                    }
                },
                y1: {
                    type: 'linear',
                    display: true,
                    position: 'right',
                    title: {
                        display: true,
                        text: 'Moisture (m³/m³)'
                    },
                    grid: {
                        drawOnChartArea: false,
                    },
                    min: 0,
                    max: function(context) {
                        const max = Math.max(...chartData.moisture.daily_avg);
                        return Math.min(1, Math.ceil(max * 10) / 10 + 0.1);
                    }
                }
            },
            plugins: {
                tooltip: {
```



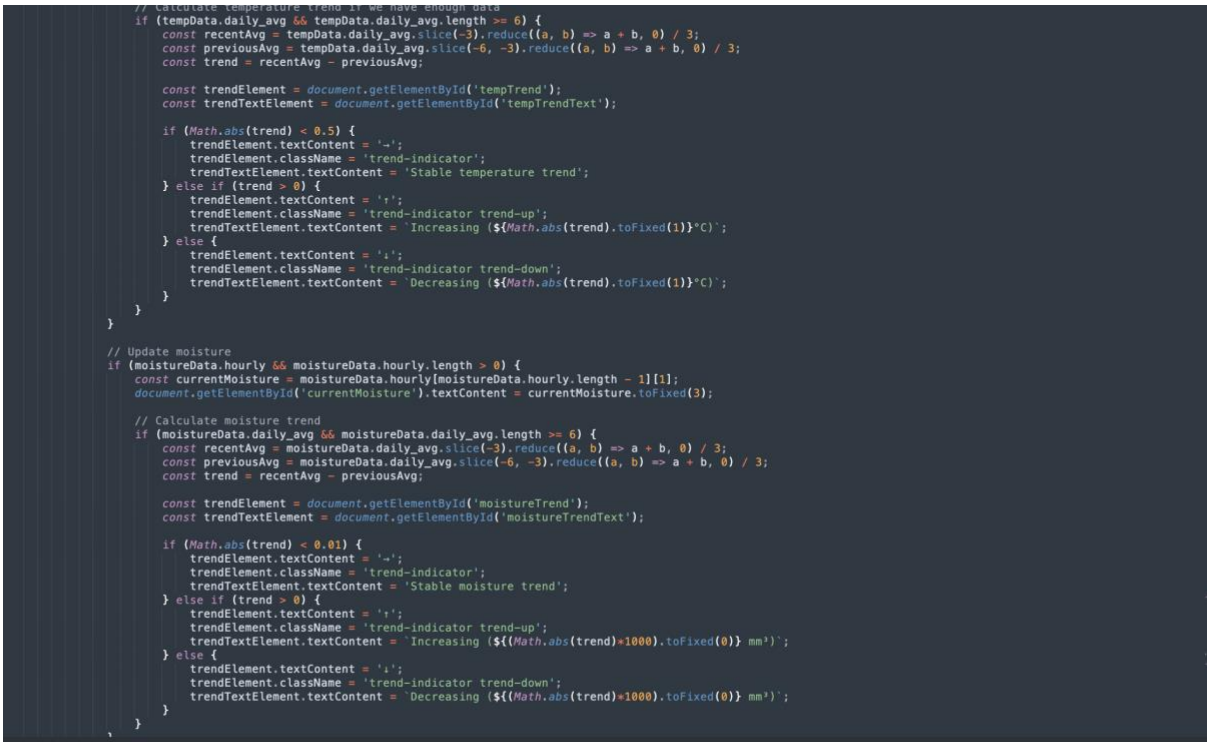
```
            plugins: {
                tooltip: {
                    callbacks: {
                        label: function(context) {
                            return `${context.dataset.label}: ${context.parsed.y}${context.datasetIndex === 0 ? '°C' : 'm³/m³'}`;
                        }
                    }
                },
                legend: {
                    position: 'top',
                },
                title: {
                    display: true,
                    text: 'Soil Temperature and Moisture at 10cm Depth',
                    font: {
                        size: 16
                    }
                }
            },
            interaction: {
                intersect: false,
                mode: 'index'
            }
        }
    });
}


function getChartOptions(title, unit, isTemp) {
    const color = isTemp ? '220, 53, 69' : '0, 123, 255';
    const data = isTemp ? chartData.temperature : chartData.moisture;

    return {
        responsive: true,
        maintainAspectRatio: false,
        scales: {
            x: {
                type: 'time',
                time: {
                    unit: 'day',
                    tooltipFormat: 'MMM dd, yyyy',
                    displayFormats: {
                        day: 'MMM dd'
                    }
                },
                title: {
                    display: true,
                    text: 'Date'
                }
            },
            y: {
                title: {
                    display: true,
```



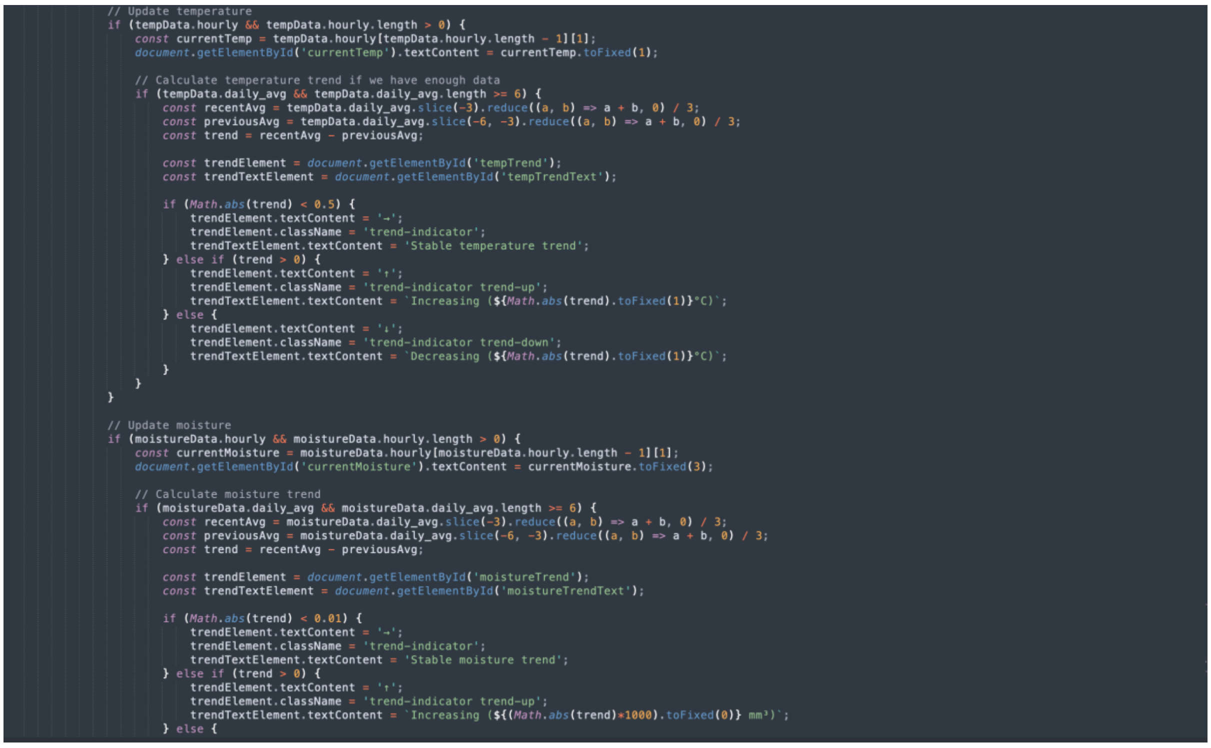
```
                    display: true,
                    text: title.includes('Temperature') ? `Temperature (${unit})` : `Moisture (${unit})`
                },
                min: function(context) {
                    const values = data.daily_avg;
                    const min = Math.min(...values);
                    return isTemp ? Math.floor(min) - 2 : 0;
                },
                max: function(context) {
                    const values = data.daily_avg;
                    const max = Math.max(...values);
                    return isTemp ? Math.ceil(max) + 2 : Math.min(1, Math.ceil(max * 10) / 10 + 0.1);
                }
            }
        },
        plugins: {
            tooltip: {
                callbacks: {
                    label: function(context) {
                        return `${context.dataset.label}: ${context.parsed.y}${unit}`;
                    }
                }
            },
            legend: {
                position: 'top',
            },
            title: {
                display: true,
                text: title,
                font: {
                    size: 16
                }
            }
        },
        interaction: {
            intersect: false,
            mode: 'index'
        }
    };
}

// Update current values and trends
function updateCurrentValues() {
    if (!chartData || !chartData.temperature || !chartData.moisture) return;

    const tempData = chartData.temperature;
    const moistureData = chartData.moisture;

    // Update temperature
    if (tempData.hourly && tempData.hourly.length > 0) {
        const currentTemp = tempData.hourly[tempData.hourly.length - 1][1];
        document.getElementById('currentTemp').textContent = currentTemp.toFixed(1);
```



```
            y: {
                title: {
                    display: true,
                    text: title.includes('Temperature') ? `Temperature (${unit})` : `Moisture (${unit})`
                },
                min: function(context) {
                    const values = data.daily_avg;
                    const min = Math.min(...values);
                    return isTemp ? Math.floor(min) - 2 : 0;
                },
                max: function(context) {
                    const values = data.daily_avg;
                    const max = Math.max(...values);
                    return isTemp ? Math.ceil(max) + 2 : Math.min(1, Math.ceil(max * 10) / 10 + 0.1);
                }
            }
        },
        plugins: {
            tooltip: {
                callbacks: {
                    label: function(context) {
                        return `${context.dataset.label}: ${context.parsed.y}${unit}`;
                    }
                }
            },
            legend: {
                position: 'top',
            },
            title: {
                display: true,
                text: title,
                font: {
                    size: 16
                }
            }
        },
        interaction: {
            intersect: false,
            mode: 'index'
        }
    };
}

// Update current values and trends
function updateCurrentValues() {
    if (!chartData || !chartData.temperature || !chartData.moisture) return;

    const tempData = chartData.temperature;
    const moistureData = chartData.moisture;

    // Update temperature
    if (tempData.hourly && tempData.hourly.length > 0) {
        const currentTemp = tempData.hourly[tempData.hourly.length - 1][1];
```



```
    // Calculate temperature trend if we have enough data
    if (tempData.daily_avg && tempData.daily_avg.length >= 6) {
        const recentAvg = tempData.daily_avg.slice(-3).reduce((a, b) => a + b, 0) / 3;
        const previousAvg = tempData.daily_avg.slice(-6, -3).reduce((a, b) => a + b, 0) / 3;
        const trend = recentAvg - previousAvg;

        const trendElement = document.getElementById('tempTrend');
        const trendTextElement = document.getElementById('tempTrendText');

        if (Math.abs(trend) < 0.5) {
            trendElement.textContent = '→';
            trendElement.className = 'trend-indicator';
            trendTextElement.textContent = 'Stable temperature trend';
        } else if (trend > 0) {
            trendElement.textContent = '↑';
            trendElement.className = 'trend-indicator trend-up';
            trendTextElement.textContent = `Increasing (${Math.abs(trend).toFixed(1)}°C)`;
        } else {
            trendElement.textContent = '↓';
            trendElement.className = 'trend-indicator trend-down';
            trendTextElement.textContent = `Decreasing (${Math.abs(trend).toFixed(1)}°C)`;
        }
    }
}

// Update moisture
if (moistureData.hourly && moistureData.hourly.length > 0) {
    const currentMoisture = moistureData.hourly[moistureData.hourly.length - 1][1];
    document.getElementById('currentMoisture').textContent = currentMoisture.toFixed(3);

    // Calculate moisture trend
    if (moistureData.daily_avg && moistureData.daily_avg.length >= 6) {
        const recentAvg = moistureData.daily_avg.slice(-3).reduce((a, b) => a + b, 0) / 3;
        const previousAvg = moistureData.daily_avg.slice(-6, -3).reduce((a, b) => a + b, 0) / 3;
        const trend = recentAvg - previousAvg;

        const trendElement = document.getElementById('moistureTrend');
        const trendTextElement = document.getElementById('moistureTrendText');

        if (Math.abs(trend) < 0.01) {
            trendElement.textContent = '→';
            trendElement.className = 'trend-indicator';
            trendTextElement.textContent = 'Stable moisture trend';
        } else if (trend > 0) {
            trendElement.textContent = '↑';
            trendElement.className = 'trend-indicator trend-up';
            trendTextElement.textContent = `Increasing (${(Math.abs(trend)*1000).toFixed(0)} mm³)`;
        } else {
            trendElement.textContent = '↓';
            trendElement.className = 'trend-indicator trend-down';
            trendTextElement.textContent = `Decreasing (${(Math.abs(trend)*1000).toFixed(0)} mm³)`;
        }
    }
```



```
// Update temperature
if (tempData.hourly && tempData.hourly.length > 0) {
    const currentTemp = tempData.hourly[tempData.hourly.length - 1][1];
    document.getElementById('currentTemp').textContent = currentTemp.toFixed(1);

    // Calculate temperature trend if we have enough data
    if (tempData.daily_avg && tempData.daily_avg.length >= 6) {
        const recentAvg = tempData.daily_avg.slice(-3).reduce((a, b) => a + b, 0) / 3;
        const previousAvg = tempData.daily_avg.slice(-6, -3).reduce((a, b) => a + b, 0) / 3;
        const trend = recentAvg - previousAvg;

        const trendElement = document.getElementById('tempTrend');
        const trendTextElement = document.getElementById('tempTrendText');

        if (Math.abs(trend) < 0.5) {
            trendElement.textContent = '→';
            trendElement.className = 'trend-indicator';
            trendTextElement.textContent = 'Stable temperature trend';
        } else if (trend > 0) {
            trendElement.textContent = '↑';
            trendElement.className = 'trend-indicator trend-up';
            trendTextElement.textContent = `Increasing (${Math.abs(trend).toFixed(1)}°C)`;
        } else {
            trendElement.textContent = '↓';
            trendElement.className = 'trend-indicator trend-down';
            trendTextElement.textContent = `Decreasing (${Math.abs(trend).toFixed(1)}°C)`;
        }
    }
}

// Update moisture
if (moistureData.hourly && moistureData.hourly.length > 0) {
    const currentMoisture = moistureData.hourly[moistureData.hourly.length - 1][1];
    document.getElementById('currentMoisture').textContent = currentMoisture.toFixed(3);

    // Calculate moisture trend
    if (moistureData.daily_avg && moistureData.daily_avg.length >= 6) {
        const recentAvg = moistureData.daily_avg.slice(-3).reduce((a, b) => a + b, 0) / 3;
        const previousAvg = moistureData.daily_avg.slice(-6, -3).reduce((a, b) => a + b, 0) / 3;
        const trend = recentAvg - previousAvg;

        const trendElement = document.getElementById('moistureTrend');
        const trendTextElement = document.getElementById('moistureTrendText');

        if (Math.abs(trend) < 0.01) {
            trendElement.textContent = '→';
            trendElement.className = 'trend-indicator';
            trendTextElement.textContent = 'Stable moisture trend';
        } else if (trend > 0) {
            trendElement.textContent = '↑';
            trendElement.className = 'trend-indicator trend-up';
            trendTextElement.textContent = `Increasing (${(Math.abs(trend)*1000).toFixed(0)} mm³)`;
        } else {
```

Evaluation Techniques

To evaluate the effectiveness of the solution, three users were invited to participate in structured interviews. Each user was presented with a demonstration of the system, including the chatbot interface, crop recommendation engine, weekly schedule generation, and trend analysis dashboard. Their feedback was collected through open-ended questions and used to identify areas for enhancement.

Evaluation Methodology

1. Method: One-on-one user interviews
2. Participants: Three users with basic to moderate knowledge of farming and digital tools
3. Format: Live demonstration followed by a structured feedback session

Interview Questions and User Responses

1. What was your overall experience using the intelligent crop planning system?

Response Summary:

All three users expressed a positive overall experience. They found the system intuitive and appreciated how it simplified complex agricultural decision-making through automation. One of them, however, found it “less expressive” in terms of the design. One user stated, "The recommendations came up as soon as without selecting location, which is good because not everyone knows how to manually input data."

2. What did you think about the AI chatbot's responses to your questions?

Response Summary:

The AI chatbot was a standout feature. All users praised its natural language responses and ability to interpret questions related to irrigation, crop care, and scheduling. Two users commented that it felt like talking to an agricultural assistant, which helped them get quick answers without searching through technical documents. Since, the chatbot's responses were not structured they wished the appearance of the responses was more like Chat GPT like including tables in responses.

3. Did you find the recommendations and schedules useful for your planning? Why or why not?

Response Summary:

Users responded positively to the recommendation engine and the weekly crop schedule feature which included the water, temperature and sunlight requirements. They found the suggestions practical and location-specific, which added to their trust in the system.

4. What additional features would make this system more useful for you?

Response Summary:

Several suggestions were made during this part of the evaluation:

1. Users wanted the ability to download their weekly plan as a checklist or report, so they could refer to it offline or share it with others. This feature was implemented shortly after the interview process.
2. Users requested real-time sensor integration, expressing interest in comparing their own soil sensor data with the system's predictions. This feedback is being considered for future integration with compatible IoT devices.
3. Users asked for extreme weather alerts, particularly warnings for hail, heatwaves, or storms. Unfortunately, the open-meteo API used in the current system only provides 16-day weather forecasts and does not include real-time or alert-level

weather data. Therefore, this feature could not be implemented due to technical limitations.

Reflections and Discussions

One of the most significant challenges of this project was to find reliable datasets which unfortunately I could not find. Therefore, I relied on APIs for this project. I continuously switched between APIs, when I found a better API or when the free limits exhausted. The most challenging part of all was implementing JavaScript. I was able to code in Python, but significant part of my work was spending on JavaScript.

This project strengthened my understanding of how different technologies interact, from backend Flask development to frontend rendering, API integration and AWS DynamoDB for data persistence.

I learned that planning is important, but flexibility is also important. Nearly every phase of this project required me to change my approach, whether due to technical limitations, feedback from users, or unexpected errors. I also learned that I should be able to explore a new idea and make an attempt to implement it, because I might actually be able to do it.

Work Log:

Date	**Description of work done**	**Number of hours**
March 31, 2025	Fetch data based on user’s IP address	1
April 1, 2025	Changed the layout of home page	1.5
April 2, 2025	Used open-meteo API to fetch 30-day soil data	2

April 3, 2025	Used JavaScript to display the trends of soil in using trend line graphs	3
April 4, 2025	Calculated increase, decrease or stability in the trend lines	0.5
April 6, 2025	Created trend line graphs for the weekly recommendation dashboard	1.5
April 7, 2025	Made trend line graph for daily water and sunlight requirements	2.5
April 8, 2025	Made a graph for maximum and minimum temperatures tolerated at each stage	2
April 9, 2025	Added the features that were requested in user evaluation, i.e., weekly schedule	1
April 11, 2025	Finished the remaining features like download pdf of the plan	2
April 12, 2025	Cleaned the code	1

Conclusion

This project has been a challenging yet deeply rewarding journey that combined real-time data, AI, and user-focused design to support farmers in making informed crop decisions. From integrating APIs and building an intelligent chatbot to implementing feedback-based features like downloadable plans, every step involved learning, problem-solving, and adapting. Despite limitations such as unavailable extreme weather data and real-time sensor integration, the final product successfully delivers personalized, accessible, and meaningful recommendations.

References

Contributors, M. O. J. T. a. B. (n.d.). Get started with Bootstrap. https://getbootstrap.com/docs/5.3/getting-started/introduction/

Docs | Open-Meteo.com. (n.d.). https://open-meteo.com/en/docs

Gardens BC. (2025, March 1). Gardens BC. https://gardensbc.com/

Mistral 7B Instruct. (n.d.). https://openrouter.ai/mistralai/mistral-7b-instruct

OpenWeatherMap.org. (n.d.). Current weather data - OpenWeatherMap. https://openweathermap.org/current

Soil temperature and moisture. (n.d.). https://soiltemperature.app/

AI Tools

1. Chat GPT: I used this tool to generate project ideas, report creation and grammar. I also used this tool to debug errors in my code.
2. DeepSeek: I used this tool to debug errors in my code.

Installation Guide:

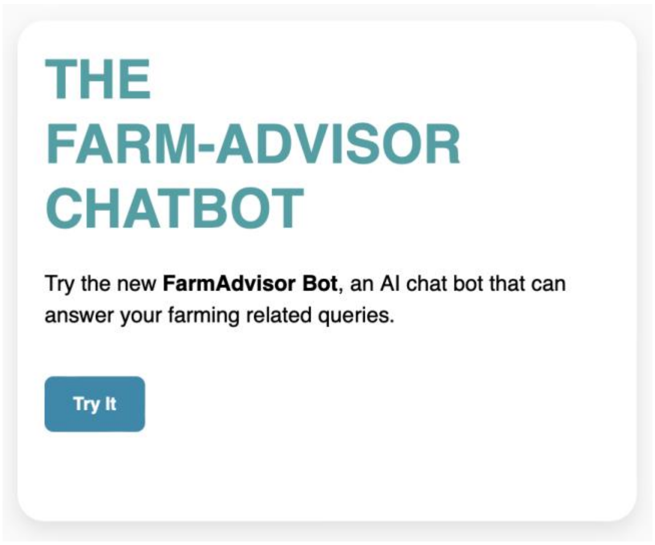
1. The user should have python installed on their device.
2. The user should install the required libraries by running following commands in the terminal:

```
pip install flask
pip install requests
pip install httpx
pip install selenium

pip install dotenv

pip install openai

pip install uuid
```

3. The project runs at the root folder by running the command, “python app.py”
4. The project run at http://127.0.0.1:5000

User Guide:

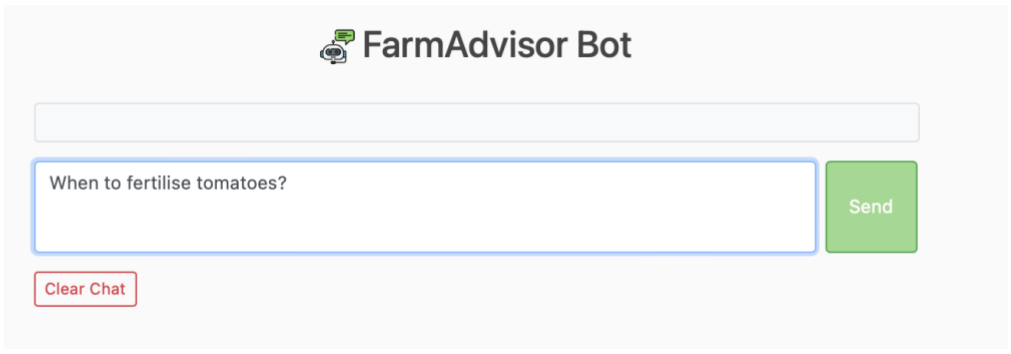
1. After going to the UR : http://127.0.0.1:5000, the user can see the recommended crops and soil conditions of their location.
2. To go to the chatbot the user must click on the “Try it” button as shown below:



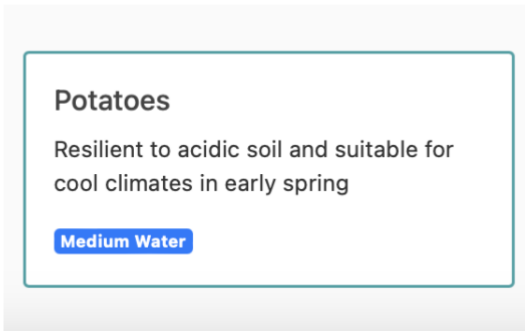
THE FARM-ADVISOR CHATBOT

Try the new **FarmAdvisor Bot**, an AI chat bot that can answer your farming related queries.

3. Type in the text box any query and hit “Send”.



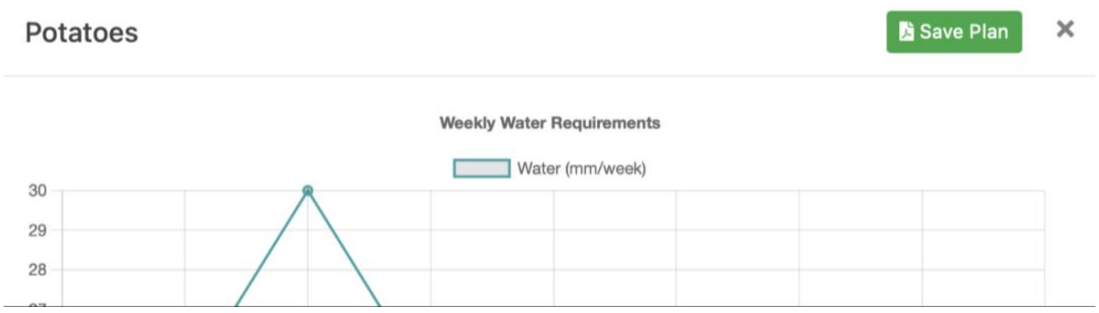
4. To see each plan user must click on the recommended crop card.



Potatoes

Resilient to acidic soil and suitable for cool climates in early spring

5. To download the plan the user must clikc on “Save Plan”


Potatoes
Save Plan
×
Weekly Water Requirements
Water (mm/week)
30
29
28