CODE:

```
import streamlit as st
import pandas as pd
import base64
import streamlit as st
from shapely.geometry import Polygon, shape
from geopy.distance import geodesic
from streamlit_folium import folium_static
import folium
from folium import plugins
import json
import base64
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read())
    st.markdown(
    f"""
    <style>
    .stApp {{
        background-image:
url({encoded_string.decode()});
```

```
        background-size: cover
    }}
    </style>
    """,
    unsafe_allow_html=True
    )
add_bg_from_local('bg.jpg')
def display_crop_history(crop_data):
    st.subheader("Crop History")
    st.write(crop_data)
    st.subheader("Input Usage")
    input_data = crop_data[['Year', 'Crop', 'Fertilizers', 'Pesticides', 'Water']]
    st.write(input_data)
    st.subheader("Graphs")
    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))
    axes[0, 0].bar(input_data['Year'], input_data['Fertilizers'])
    axes[0, 0].set_title('Fertilizer Usage')
    axes[0, 1].bar(input_data['Year'], input_data['Pesticides'])
    axes[0, 1].set_title('Pesticide Usage')
    axes[1, 0].plot(input_data['Year'], input_data['Water'])
    axes[1, 0].set_title('Water Usage')
    axes[1, 1].plot(input_data['Year'], input_data['Fertilizers'], marker='o',
label='Fertilizers')
    axes[1, 1].plot(input_data['Year'], input_data['Pesticides'], marker='o',
label='Pesticides')
    axes[1, 1].plot(input_data['Year'], input_data['Water'], marker='o',
label='Water')
```

```
    axes[1, 1].set_title('Input Usage Comparison')
    axes[1, 1].legend()
    plt.tight_layout()
    st.pyplot(fig)

def home():
     # Center-align all elements
    # Logo
    st.markdown("<div class='st-centered_img'><img src='https://www.freeiconspng.com/uploads/green-leaf-png-9.png' width='80'></div>", unsafe_allow_html=True)

    # Text with size 100
    st.markdown("<h2 class='st-centered' style='color: #5BEAB1;'>Farm Management System</h2>", unsafe_allow_html=True)

    # Heading
    st.markdown("<h1 class='st-centered' style='color: #5BEAB1;'></h1>", unsafe_allow_html=True)

    # Simple center-aligned text
    st.markdown("<p class='st-centered' style='color: #5BEAB1;'> A comprehensive solution to streamline all farming procedures and activities. Optimize productivity, make informed decisions, and embrace the future of farming.</p>", unsafe_allow_html=True)

    # Button to display a paragraph
```

```
col4, col5,col6 =st.columns(3)
with col5:
    if st.button("About the application"):
        st.header("Field Mapping and Delineation")
        st.write("Effortlessly map and delineate your fields using Farm_era's intuitive interface. Visualize your land boundaries and effectively manage your fields.")
        st.header("Crop Planning and Rotation Management")
        st.write("Plan your crop planting schedules and manage rotation cycles with ease. Keep track of different crop varieties and maintain optimal soil health for improved yields.")
        st.header("Input Usage and Crop History")
        st.write("Record and track the usage of fertilizers, pesticides, water, and other inputs. Maintain a detailed crop history for compliance and better resource management.")
        st.header("Pest and Disease Management")
        st.write("Receive timely alerts and recommendations for pest and disease control. Access integrated databases to identify and address issues promptly, safeguarding your crops.")
        st.header("Weather Tracking")
        st.write("Stay informed about changing weather conditions with real-time updates and forecasts. Utilize historical weather data to make informed decisions for your farming activities.")
        st.header("Irrigation Scheduling and Monitoring")
        st.write("Optimize water usage and enhance irrigation practices. Set up customized schedules based on crop needs and monitor water usage to promote sustainable farming.")
```

```
    st.write("Farm_era is your reliable companion, providing seamless integration, comprehensive features, and user-friendly analytics. Simplify your crop and field management, increase efficiency, and embrace the future of farming with Farm_era.")
    # Heading in a different color
    st.markdown("<h2 class='st-centered' style='color: white;'>Monitor Your Farm</h2>", unsafe_allow_html=True)
    # Boxes with hover effect
    col1, col2, col3 = st.columns(3)
    with col1:
# Field mapping and delineation
def field_mapping():
    st.header("MAPPINGT")
    # Sidebar
    st.sidebar.title("Enter GPS Coordinates")
    # Input GPS coordinates
    latitude = st.sidebar.number_input('Enter latitude', -90.0, 90.0, 0.0)
    longitude = st.sidebar.number_input('Enter longitude', -180.0, 180.0, 0.0)
    # Create a map centered at the input coordinates
    m = folium.Map(location=[latitude, longitude], zoom_start=13)
    # Add drawing tool to the map
    draw = plugins.Draw(export=True)
    draw.add_to(m)
    # Display the map in the Streamlit app
    folium_static(m)
    # File uploader for the GeoJSON file
    st.sidebar.title("Upload GeoJSON")
```

```
    uploaded_file = st.sidebar.file_uploader("Upload the GeoJSON file")
    if uploaded_file is not None:
        # Load the GeoJSON file
        geojson_data = json.load(uploaded_file)
        i=0
        # Get the coordinates of the polygon
        for feature in geojson_data['features']:
            i+=1
            if feature['geometry']['type'] == 'Polygon':
                # Get the vertices
                vertices = feature['geometry']['coordinates'][0]
                vertices = [(lon, lat) for lon, lat in vertices]  # Flip coordinates
                # Calculate and display the area if there are enough vertices
                if len(vertices) >= 3:
                    polygon = Polygon(vertices)
                    area = polygon.area
                    st.header(f"Area of the Field{i}: \n{area} square units")
                # Calculate and display the distances between the vertices
                for i in range(len(vertices) - 1):
                    distance = geodesic(vertices[i], vertices[i+1]).miles
                    st.header("boundary distance")
                    st.write(f"Distance between point {i+1} and point {i+2}:
{distance} miles")
def crop_planning():
    # Implement crop planning logic
    st.header("Crop Planning")
    # Sidebar inputs
```

```
    st.sidebar.title("Inputs")
    # Get number of crops from user
    num_crops = st.sidebar.number_input("Number of Crops", min_value=1, step=1, value=3)
    # Create lists to store crop data
    crops = []
    areas = []
    pod_counts = []
    grain_counts = []
    grain_weights = []
    rainfall_data_per_crop = []
    # Collect crop data from user inputs
    for i in range(num_crops):
        crop = st.sidebar.text_input(f"Crop {i+1} Name")
        area = st.sidebar.number_input(f"Crop {i+1} Area (in acres)", min_value=1, step=1, value=10)
        pod_count = st.sidebar.number_input(f"Crop {i+1} Pod Count (Average of 5 measurements)", min_value=1, step=1, value=10)
        grain_count = st.sidebar.number_input(f"Crop {i+1} Grain Count (Average of 20 measurements)", min_value=1, step=1, value=50)
        grain_weight = st.sidebar.number_input(f"Crop {i+1} Grain Weight (per grain)", min_value=1, step=1, value=10)
        crops.append(crop)
        areas.append(area)
        pod_counts.append(pod_count)
        grain_counts.append(grain_count)
        grain_weights.append(grain_weight)
```

```
        # Collect rainfall data for each crop
        months = st.sidebar.multiselect(f"Select months for Crop {i+1}", ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"])
        rainfall_data = []
        for month in months:
            month_rainfall = st.sidebar.number_input(f"Crop {i+1} - {month} Rainfall (in mm)", min_value=0, step=1, value=0, key=f"rainfall_{i}_{month}")
            rainfall_data.append(month_rainfall)
        rainfall_data_per_crop.append(rainfall_data)
    # Perform crop planning calculations
    crop_yields = [(areas[i] * pod_counts[i] * grain_counts[i] * grain_weights[i]) / 10000 for i in range(num_crops)]  # Placeholder calculation
    # Display crop planning results
    st.header("Crop Planning Results")
    # Display crop yields in a table
    df_yields = pd.DataFrame({"Crop": crops, "Yield": crop_yields})
    st.subheader("Crop Yields")
    st.dataframe(df_yields)
    # Display pie chart to compare crop yields
    st.subheader("Crop Yield Comparison")
    fig, ax = plt.subplots()
    ax.pie(crop_yields, labels=crops, autopct='%1.1f%%', startangle=90)
    ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    ax.set_title("Crop Yield Comparison")
    st.pyplot(fig)
    # Display graphs
```

```
    st.header("Graphs")
    # Bar chart for individual crop yields
    fig, ax = plt.subplots()
    sns.barplot(data=df_yields, x="Crop", y="Yield", ax=ax)
    ax.set_title("Crop Yields")
    st.pyplot(fig)
    # Line chart for monthly rainfall
    fig, ax = plt.subplots()
    for i, crop in enumerate(crops):
        data = pd.DataFrame({"Month": [month for month, rainfall in zip(months,
rainfall_data_per_crop[i]) if rainfall != 0],
                             "Rainfall": [rainfall for rainfall in rainfall_data_per_crop[i] if
rainfall != 0]})
        sns.lineplot(data=data, x="Month", y="Rainfall", ax=ax, label=crop)
    ax.set_title("Monthly Rainfall")
    ax.set_ylabel("Rainfall (mm)")
    ax.legend()
    st.pyplot(fig)
    # Display images
    st.header("Images")
    uploaded_images_per_crop = []
    for i in range(num_crops):
        uploaded_images = st.sidebar.file_uploader(f"Upload images for Crop
{i+1}", accept_multiple_files=True)
        uploaded_images_per_crop.extend(uploaded_images)
    num_images_per_row = 3
    num_images = len(uploaded_images_per_crop)
```

```
    num_rows = (num_images + num_images_per_row - 1) // num_images_per_row
    for row in range(num_rows):
        cols = st.columns(num_images_per_row)
        for col in range(num_images_per_row):
            index = row * num_images_per_row + col
            if index < num_images:
                cols[col].image(uploaded_images_per_crop[index], caption=f"Image {index+1}", use_column_width=True)
# Recording crop history and input usage
def crop_history():
    # Implement crop history recording logic
    st.header("Crop History and Input Usage")
    # Load existing data from CSV or create an empty dataframe
    try:
        crop_data = pd.read_csv('crop_data.csv')
    except FileNotFoundError:
        crop_data = pd.DataFrame(columns=['Year', 'Crop', 'Fertilizers', 'Pesticides', 'Water']
    # Display existing crop data
    if not crop_data.empty:
        display_crop_history(crop_data)
    # Collect new data from user inputs
    st.subheader("Enter Crop History and Input Usage")
    year = st.number_input("Year", min_value=1900, max_value=2100, value=2023, step=1)
    crop = st.text_input("Crop")
```

```
    fertilizers = st.number_input("Fertilizers (in kg)", min_value=0.0, value=0.0, step=0.1)
    pesticides = st.number_input("Pesticides (in kg)", min_value=0.0, value=0.0, step=0.1)
    water = st.number_input("Water (in mm)", min_value=0.0, value=0.0, step=0.1)
    # Save new data to the dataframe and CSV
    if st.button("Save"):
        new_entry = pd.DataFrame([[year, crop, fertilizers, pesticides, water]], columns=crop_data.columns)
        crop_data = pd.concat([crop_data, new_entry], ignore_index=True)
        crop_data.to_csv('crop_data.csv', index=False)
        st.success("Data saved successfully!")
        # Display updated crop data
        display_crop_history(crop_data)
# Pest and disease management
def pest_management():
    # Implement pest management logic
    st.header("Pest and Disease Management")
    st.sidebar.header("Pest and Disease Management System")
    farm_name = st.sidebar.text_input("Farm Name")
    date = st.sidebar.date_input("Date")
    # Pest and Disease Inputs
    st.sidebar.subheader("Pest and Disease Inputs")
    pest_input = st.sidebar.text_input("Pests (comma-separated)")
    disease_input = st.sidebar.text_input("Diseases (comma-separated)")
```

```
    # Pest Management Options
    st.sidebar.subheader("Pest Management Options")
    pest_management_option_input = st.sidebar.text_input("Pest Management Options (comma-separated)")
    pest_management_options = [option.strip() for option in pest_management_option_input.split(",")]
    # Real-time Tracking
    st.sidebar.subheader("Real-time Tracking")
    temperature = st.sidebar.number_input("Temperature (°C)", min_value=-50, max_value=50, value=25, step=1)
    humidity = st.sidebar.number_input("Humidity (%)", min_value=0, max_value=100, value=50, step=1)
    rainfall = st.sidebar.number_input("Rainfall (mm)", min_value=0, value=0, step=1)
    # Data Collection
    st.header("Data Collection")
    # Create a DataFrame to store the collected data
    data = pd.DataFrame({
        "Date": [date],
        "Farm Name": [farm_name],
        "Pests": [pest_input],
        "Diseases": [disease_input],
        "Temperature (°C)": [temperature],
        "Humidity (%)": [humidity],
        "Rainfall (mm)": [rainfall]
    })
    # Display the collected data
```

```
    st.write(data)
    # Interactive Graphs
    st.header("Interactive Graphs")
    # Generate random data for demonstration
    num_days = 30
    dates = pd.date_range(end=pd.to_datetime(date), periods=num_days,
freq="D")
    temperatures = np.random.normal(25, 5, num_days)
    humidities = np.random.normal(50, 10, num_days)
    rainfalls = np.random.randint(0, 10, num_days)
    # Create a DataFrame with random data
    df = pd.DataFrame({
        "Date": dates,
        "Temperature (°C)": temperatures,
        "Humidity (%)": humidities,
        "Rainfall (mm)": rainfalls
    })
    # Line plot for temperature
    fig, ax1 = plt.subplots()
    ax1.plot(df["Date"], df["Temperature (°C)"], color="tab:red")
    ax1.set_xlabel("Date")
    ax1.set_ylabel("Temperature (°C)", color="tab:red")
    ax1.tick_params(axis="y", labelcolor="tab:red")
    # Bar plot for humidity
    ax2 = ax1.twinx()
    ax2.bar(df["Date"], df["Humidity (%)"], color="tab:blue", alpha=0.3)
    ax2.set_ylabel("Humidity (%)", color="tab:blue")
```

```
ax2.tick_params(axis="y", labelcolor="tab:blue")
# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
# Display the graph
st.pyplot(fig)

# Line plot for rainfall
fig2, ax3 = plt.subplots()
ax3.plot(df["Date"], df["Rainfall (mm)"], color="tab:green")
ax3.set_xlabel("Date")
ax3.set_ylabel("Rainfall (mm)", color="tab:green")
ax3.tick_params(axis="y", labelcolor="tab:green")
# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
# Display the graph
st.pyplot(fig2)
# Additional visualizations can be added here
# Pest Management Result
if pest_management_options:
    st.header("Pest Management Result")

    # Extract individual pests from pest_input
    pests = [pest.strip() for pest in pest_input.split(",")]

    # Check the length of pest management options
    if len(pest_management_options) >= len(pests):
        # Generate sample pest management data
```

```
            pest_management_data = pd.DataFrame({
                "Pest": pests,
                "Pest Management Option": pest_management_options[:len(pests)],
                "Effectiveness": np.random.uniform(0.5, 0.9, len(pests))
            })  # Display the pest management result as a table
            st.table(pest_management_data)
        else:
            # Use available pest management options for multiple pests
            num_pests = len(pests)
            num_options = len(pest_management_options)
            repetition = num_pests // num_options
            remainder = num_pests % num_options
            pest_management_options_extended = pest_management_options * repetition + pest_management_options[:remainder]
            pest_management_data = pd.DataFrame({
                "Pest": pests,
                "Pest Management Option": pest_management_options_extended,
                "Effectiveness": np.random.uniform(0.5, 0.9, num_pests)
            })
            # Display the pest management result as a table
            st.table(pest_management_data)
    else:
        st.info("No pest management options provided.")
    st.header("Raw Data")
    st.write(df)
    # Show the Streamlit application
    st.sidebar.markdown("---")
```

```
    st.sidebar.markdown("Developed by Sowmya Surapaneni")
# Weather tracking and integration
def weather_tracking():
    # Implement weather tracking logic
    st.header("Weather Tracking and Integration")
    st.sidebar.header("Weather Tracking")
    city = st.sidebar.text_input("City", value="New York")
    start_date = st.sidebar.date_input("Start Date")
    end_date = st.sidebar.date_input("End Date")

    # Data Collection
    st.header("Data Collection")
    # Generate random weather data
    num_days = (end_date - start_date).days + 1
    dates = pd.date_range(start=start_date, end=end_date, freq="D")
    temperatures = np.random.normal(25, 5, num_days)
    humidities = np.random.normal(50, 10, num_days)
    rainfalls = np.random.randint(0, 10, num_days)
    # Create a DataFrame with random data
    weather_data = pd.DataFrame({
        "Date": dates,
        "Temperature (°C)": temperatures,
        "Humidity (%)": humidities,
        "Rainfall (mm)": rainfalls
    })
    # Display the collected data
    st.write(weather_data)
```

```
# Visualizations
st.header("Visualizations")
# Line plot for temperature
st.subheader("Temperature")
fig_temp = plt.figure(figsize=(10, 6))
plt.plot(weather_data["Date"], weather_data["Temperature (°C)"])
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.title("Temperature Variation")
plt.xticks(rotation=45)
st.pyplot(fig_temp)

# Line plot for humidity
st.subheader("Humidity")
fig_humidity = plt.figure(figsize=(10, 6))
plt.plot(weather_data["Date"], weather_data["Humidity (%)"])
plt.xlabel("Date")
plt.ylabel("Humidity (%)")
plt.title("Humidity Variation")
plt.xticks(rotation=45)
st.pyplot(fig_humidity)
# Bar plot for rainfall
st.subheader("Rainfall")
fig_rainfall = plt.figure(figsize=(10, 6))
plt.bar(weather_data["Date"], weather_data["Rainfall (mm)"])
plt.xlabel("Date")
plt.ylabel("Rainfall (mm)")
```

```
    plt.title("Daily Rainfall")
    plt.xticks(rotation=45)
    st.pyplot(fig_rainfall)
    st.header("Summary Statistics")
    # Data summary
    st.subheader("Weather Data Summary")
    st.write(weather_data.describe())
    # Correlation heatmap
    st.subheader("Correlation Heatmap")
    corr = weather_data.corr()
    fig_corr = plt.figure(figsize=(8, 6))
    sns.heatmap(corr, annot=True, cmap="coolwarm")
    st.pyplot(fig_corr)

# Irrigation scheduling and monitoring
def irrigation_management():
    # Implement irrigation management logic
    st.header("Irrigation Scheduling and Monitoring")
    number1 = st.number_input("Enter the water content of the feild 1:", min_value=0, max_value=10, value=0, step=1)
    number2 = st.number_input("Enter the water content of the feild 2:", min_value=0, max_value=10, value=0, step=1)
    number3 = st.number_input("Enter the water content of the feild 3:", min_value=0, max_value=10, value=0, step=1)
    number4 = st.number_input("Enter the water content of the feild 4:", min_value=0, max_value=10, value=0, step=1)
    if st.button('start monitoring'):
```

```
        os.system("python cam.py")
    if st.button('start irrigation'):
# Delete previous values in the text file
        with open("water_flow_data.txt", "w") as file:
            file.write("")  # This line clears the contents of the file
        # Save the numbers to the text file
        with open("water_flow_data.txt", "a") as file:
            file.write(str(number1) + "\n")
            file.write(str(number2) + "\n")
            file.write(str(number3) + "\n")
            file.write(str(number4) + "\n")
# Display success message
        st.success("irrigation Started")
    if st.button('irrigation monitor'):
        os.system("streamlit run test.py")



# Sidebar navigation
menu_options = ["Home", "Field Mapping", "Crop Planning", "Crop History",
                "Pest Management", "Weather Tracking", "Irrigation Management"]
# Display the menu options
selected_menu = st.sidebar.selectbox("Select an option:", menu_options)
# Display corresponding page based on selected menu option
if selected_menu == "Home":
    home()
elif selected_menu == "Field Mapping":
    field_mapping()
```

```
elif selected_menu == "Crop Planning":
    crop_planning()
elif selected_menu == "Crop History":
    crop_history()
elif selected_menu == "Pest Management":
    pest_management()
elif selected_menu == "Weather Tracking":
    weather_tracking()
elif selected_menu == "Irrigation Management":
    irrigation_management()
menu_options = st.sidebar.multiselect("Select options:", ["Home", "Field Mapping", "Crop Planning", "Crop History", "Pest Management","Weather Tracking", "Irrigation Management"])
# Display corresponding pages based on selected menu options
if "Home" in menu_options:
    home()
if "Field Mapping" in menu_options:
    field_mapping()
if "Crop Planning" in menu_options:
    crop_planning()
if "Crop History" in menu_options:
    crop_history()
if "Pest Management" in menu_options:
    pest_management()
if "Weather Tracking" in menu_options:
    weather_tracking()
if "Irrigation Management" in menu_options:
```

```
    irrigation_management()
```

IRRIGATION.PY

```
import random
import csv
import time
# Constants for simulation
NUM_ZONES = 4  # Number of irrigation zones
MOISTURE_RANGE = (20, 80)  # Range of moisture values (%)
WATER_FLOW_RANGE = (1, 10)  # Range of water flow values (liters per
minute)
# Function to generate random sensor data
def generate_sensor_data():
    moisture_data = [random.randint(*MOISTURE_RANGE) for _ in
range(NUM_ZONES)]
    water_flow_data = [random.uniform(*WATER_FLOW_RANGE) for _ in
range(NUM_ZONES)]
    return moisture_data, water_flow_data
# Function to update and store sensor data
def update_sensor_data():
    while True:
        # Generate random sensor data
        moisture_data, water_flow_data = generate_sensor_data()
        # Store the updated sensor data in a data file (e.g., CSV)
        with open('sensor_data.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(['Zone', 'Moisture', 'Water Flow'])
            for i in range(NUM_ZONES):
                writer.writerow([i+1, moisture_data[i], water_flow_data[i]])
        # Wait for some time before updating again
        time.sleep(3)  # Adjust the sleep time as per your requirements
if __name__ == "__main__":
    update_sensor_data()
```

TEST.PY

```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import random
import winsound
import cv2
import time
import os


# Constants for simulation
NUM_ZONES = 4  # Number of irrigation zones
MOISTURE_RANGE = (20, 80)  # Range of moisture values (%)
WATER_FLOW_RANGE = (1, 4)  # Range of water flow values (liters per minute)
THRESHOLD = 88  # Threshold for water flow

# Function to generate random sensor data
def generate_sensor_data():
    moisture_data = [random.randint(*MOISTURE_RANGE) for _ in range(NUM_ZONES)]
    return moisture_data

# Function to gradually increase and decrease water flow until threshold is reached
def generate_water_flow_data():
    water_flow_data = []
```

```
    with open("water_flow_data.txt", "r") as prev_file, open("flow_decision.txt", "r+") as decision_file:
        flow_decision = decision_file.readline().strip()
        if flow_decision == "increase":
            increasing = True
        elif flow_decision == "decrease":
            increasing = False
        else:
            raise ValueError("Invalid flow decision in the text file")

        line = prev_file.readline().strip()
        flow = float(line)  # Read flow value from the previous file

        while flow < THRESHOLD and increasing:
            flow += random.uniform(*WATER_FLOW_RANGE)  # Increase water flow gradually
            water_flow_data.append(flow)
            if flow >= THRESHOLD:
                increasing = False  # Switch to decreasing once the threshold is reached
                decision_file.seek(0)  # Reset the file position to update the decision
                decision_file.write("decrease")  # Update the flow decision in the file
                decision_file.truncate()  # Clear any remaining content in the file

        while flow > 8 and not increasing:
            flow -= random.uniform(*WATER_FLOW_RANGE)  # Decrease water flow gradually
```

```
            if flow < 8:
                flow = 8
                increasing = True  # Switch to increasing once the minimum value is
reached
                decision_file.seek(0)  # Reset the file position to update the decision
                decision_file.write("increase")  # Update the flow decision in the file
                decision_file.truncate()
            water_flow_data.append(flow)
            if flow <= 8:
                increasing = True  # Switch to increasing once the minimum value is
reached
                decision_file.seek(0)  # Reset the file position to update the decision
                decision_file.write("increase")  # Update the flow decision in the file
                decision_file.truncate()  # Clear any remaining content in the file

    water_flow_data = water_flow_data[:NUM_ZONES]  # Truncate the list to
the desired length
    return water_flow_data




# Function to generate random wastage detection
def detect_water_wastage(water_flow_data):
    return [flow > THRESHOLD for flow in water_flow_data]

# Function to play beep sound
def play_beep_sound():
```

```
    frequency = 2500  # Adjust the frequency as per your requirements
    duration = 1000  # Adjust the duration as per your requirements
    winsound.Beep(frequency, duration)

# Function to capture camera frame



# Streamlit application
def main():
    st.title("Irrigation Management System")

    while True:
        # Generate random sensor data
        moisture_data = generate_sensor_data()

        # Display moisture data
        st.subheader("Moisture Data")
        moisture_df = pd.DataFrame({"Zone": range(1, NUM_ZONES+1),
"Moisture": moisture_data})
        st.dataframe(moisture_df)

        # Generate random water flow data
        water_flow_data = generate_water_flow_data()

        # Write water flow data to a text file
        with open("water_flow_data.txt", "w") as file:
            file.write("\n".join(str(flow) for flow in water_flow_data))
```

```
        # Display water flow data
        st.subheader("Water Flow Data")
        water_flow_df = pd.DataFrame({"Zone": range(1, NUM_ZONES+1),
"Water Flow": water_flow_data})
        st.dataframe(water_flow_df)

        # Plot graphs
        st.subheader("Data Visualization")
        fig, axes = plt.subplots(2, 1, figsize=(8, 6))
        axes[0].bar(range(1, NUM_ZONES+1), moisture_data)
        axes[0].set_ylabel("Moisture (%)")
        axes[1].bar(range(1, NUM_ZONES+1), water_flow_data)
        axes[1].set_ylabel("Water Flow (liters/min)")
        st.pyplot(fig)

        # Water wastage detection
        st.subheader("Water Wastage Detection")
        wastage_detected = detect_water_wastage(water_flow_data)
        if any(wastage_detected):
            st.warning("Water wastage detected!")
            play_beep_sound()
            break
        # Camera integration


        # Wait for some time before updating again
```

```
        time.sleep(1)
        st.experimental_rerun()
    st.header("irrigation status")
    st.success("Irrigation complete")

if __name__ == "__main__":
    main()
```

MAPPING.PY

```
import streamlit as st
from shapely.geometry import Polygon, shape
from geopy.distance import geodesic
from streamlit_folium import folium_static
import folium
from folium import plugins
import json
import base64
def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read())
    st.markdown(
    f"""
    <style>
    .stApp {{
        background-image:
url({encoded_string.decode()});
        background-size: cover
```

```
    }}
    </style>
    """,
    unsafe_allow_html=True
    )
add_bg_from_local('bg.jpg')
# App title
st.title("Field Mapping for Farmers")

# Sidebar
st.sidebar.title("Enter GPS Coordinates")
# Input GPS coordinates
latitude = st.sidebar.number_input('Enter latitude', -90.0, 90.0, 0.0)
longitude = st.sidebar.number_input('Enter longitude', -180.0, 180.0, 0.0)

# Create a map centered at the input coordinates
m = folium.Map(location=[latitude, longitude], zoom_start=13)

# Add drawing tool to the map
draw = plugins.Draw(export=True)
draw.add_to(m)

# Display the map in the Streamlit app
folium_static(m)

# File uploader for the GeoJSON file
st.sidebar.title("Upload GeoJSON")
```

```
uploaded_file = st.sidebar.file_uploader("Upload the GeoJSON file")

if uploaded_file is not None:
    # Load the GeoJSON file
    geojson_data = json.load(uploaded_file)
    i=0
    # Get the coordinates of the polygon
    for feature in geojson_data['features']:
        i+=1
        if feature['geometry']['type'] == 'Polygon':
            # Get the vertices
            vertices = feature['geometry']['coordinates'][0]
            vertices = [(lon, lat) for lon, lat in vertices]  # Flip coordinates

            # Calculate and display the area if there are enough vertices
            if len(vertices) >= 3:
                polygon = Polygon(vertices)
                area = polygon.area
                st.header(f"Area of the Feild{i}: \n{area} square units")

            # Calculate and display the distances between the vertices
            for i in range(len(vertices) - 1):
                distance = geodesic(vertices[i], vertices[i+1]).miles
                st.header("boundary distance")
                st.write(f"Distance between point {i+1} and point {i+2}: {distance}
miles")
```

CAM.PY

```
import cv2

def cam():
    cap = cv2.VideoCapture('Modern irrigation system.mp4')  # Adjust the video file path as per your system

    while True:
        ret, frame = cap.read()

        # Check if the frame is empty (end of the video)
        if not ret:
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
            continue

        cv2.imshow("Camera Feed", frame)

        # Check for 'q' key press to exit
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release the video capture and close OpenCV windows
    cap.release()
    cv2.destroyAllWindows()

cam()
```