ДОДАТОК А

Апробація результатів роботи

УДК 681.5

ДОСЛІДЖЕННЯ СИСТЕМ АВТОМАТИЗАЦІЇ АНАЛІЗУ ҐРУНТУ НА БАЗІ ТЕХНОЛОГІЇ ІНТЕРНЕТУ РЕЧЕЙ

І.М. Надьожкіна
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр. Науки 14
E-mail: iryna.nadozhkina@nure.ua

***Анотація*:** У даній статті розглядаються системи автоматизації аналізу ґрунту на базі технології IoT, описуються методи збору інформації і передачі. Проводиться аналіз переваг і недоліків кожної системи.

Ключові слова: IoT, система, аналіз ґрунту, датчики.

RESEARCH OF SOIL ANALYSIS AUTOMATION SYSTEMS BASED ON INTERNET OF THINGS TECHNOLOGY

I. Nadozhkina
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, Nauky av.,14
E-mail: iryna.nadozhkina@nure.ua

***Annotation*:** This article discusses automation systems for analyzing the justification of the IoT technology base, describes methods for collecting and transmitting information, and analyzes the advantages and disadvantages of each system.

Keywords: IoT, system, soil analysis, sensors.

У сучасних реаліях розвитку та впровадження новітніх технологій, для сільського господарства автоматизація процесів також є не менш актуальною. У цьому напрямі необхідний точний та своєчасний аналіз характеристик ґрунту, що безпосередньо впливає на саму родючість, на якість та обсяги врожаю, ефективність використання ресурсів, а також екологічну сталість аграрного виробництва [1].

На зміну традиційним методам приходять нові, удосконалені, тому що ті, які зазвичай використовувалися і були ефективними у минулому, вже не відповідають сучасним вимогам. Але з появою концепції точного землеробства ефективність моніторингу полів та стану ґрунту підвищилася, такі системи використовують різні технології для цього. Наприклад, технологія Інтернету речей (IoT), що є мережевим поєднанням взаємопов’язаних фізичних пристроїв,

дозволяє у режимі реального часу передавати та обробляти дані про стан ґрунту [2]. Так об'єднувати однією мережею можна сенсори, датчики, виконавчі механізми тощо. Інтеграція IoT у сільське господарство дає можливість налагодити безперервний моніторинг вологості, температурного режиму, кислотності, щільності ґрунту, вмісту поживних речовин, тощо. Завдяки цьому фермери або сільськогосподарські роботи можуть оперативно реагувати на зміну умов або заздалегідь мати інформацію про можливі проблеми.

В залежності від потреб, розвилися різні системи автоматизованого аналізу ґрунту, але деякі сучасні можуть поєднувати у собі різні методи та технології одразу. Вибір типу системи автоматизації для аналізу ґрунту безпосередньо впливає на подальшу архітектуру всього рішення: типи сенсорів, методи збору та передачі даних, а також технології, що використовуються для обробки інформації. Саме від цього чинника залежить масштаб застосування системи, ступінь мобільності, частота оновлення даних, потреба у додатковому обладнанні та програмному забезпеченні.

Наразі ці системи можна умовно класифікувати на стаціонарні, мобільні та гібридні.

Стаціонарні системи. Є найбільш розповсюдженим типом рішень для аналізу ґрунту, тому що прості у використанні і доступні. Датчики встановлюють на визначених ділянках у землю для постійного моніторингу даних про фізико-хімічні параметри ґрунту. Зазвичай, ці системи використовуються у теплицях, невеликих полях з однорідними умовами або прямо вдома для кімнатних рослин [3]. На рисунку 1 зображено датчик вологості та температури ґрунту GXM-01. Недоліками можна вважати обмежене охоплення території, так як кожен датчик фіксує дані лише з однієї точки, а для отримання повної картини необхідно встановлювати велику кількість таких, що істотно підвищує вартість системи та її обслуговування, тому більшість стаціонарних систем не підходять для громіздких задач.

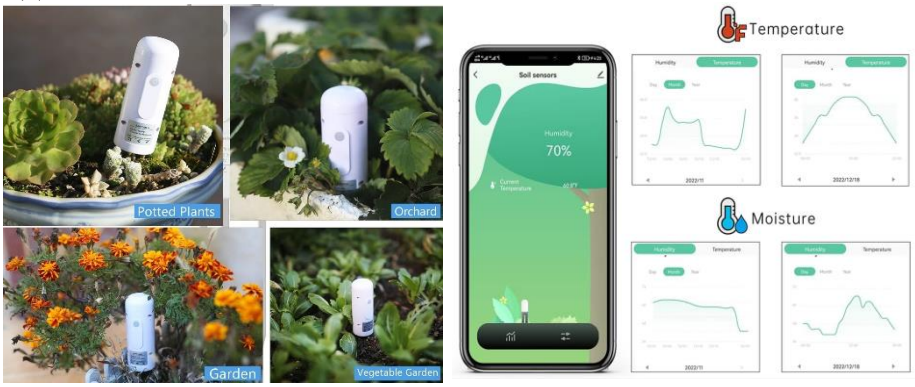

Рисунок 1 – Застосування датчику GXM-01 та вигляд мобільного застосунку

Мобільні системи. Їх застосовують, коли є потреба в охопленні великої площі для отриманні повної картини стану ділянок ґрунту. Такі рішення базуються на рухомих платформах, зокрема дронах, наземних роботах або сільськогосподарській техніці, яка оснащується відповідними датчиками. Під час переміщення територією ці пристрої здійснюють збір та фіксацію показників

ґрунту в різних точках, після чого створюється мапа стану ґрунтового покриву і надається користувачеві у мобільному застосунку. На рисунку 2 представлено вигляд платформи Veris U3 та pH стану ґрунту на мапі після об'їзду поля [4]. Недоліками є висока вартість, потреба у регулярному технічному обслуговуванні; точність даних залежить від маршруту та якості проходження платформи, а погодні умови можуть негативно впливати на роботу дронів або наземних апаратів; вимагають більшої енергетичної потужності та попереднього планування маршрутів збору інформації.

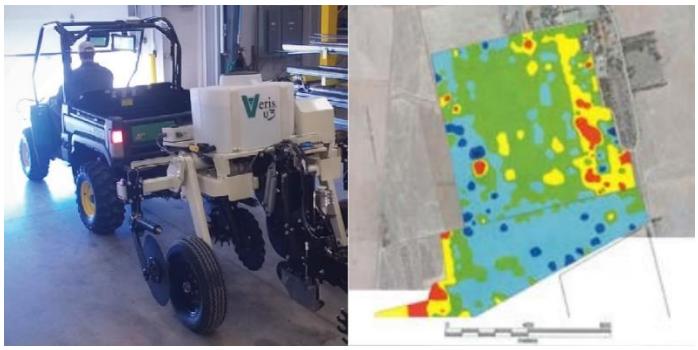

Рисунок 2 – Система Veris

Гібридні системи. Вони поєднують елементи як стаціонарного, так і мобільного моніторингу. В таких рішеннях використовуються стаціонарні сенсори для регулярного збору базових даних у ключових зонах, у той час як мобільні засоби (дрони на супутникові знімки) дають змогу періодично здійснювати детальне просторове обстеження полів, далі дані аналізуються в хмарі, а результатом є мапа посіву або урожайності, вологості тощо. У приклад можна навести систему Climate FieldView [5]. На рисунку 3 показано завдання з посіву у застосунку. Недоліками є реалізація координації між двома типами обладнання, що ускладнює інтеграцію і підвищує витрати; складніші у налаштуванні, бо потребують кваліфікованого персоналу для обслуговування, можуть бути чутливими до збоїв у будь-якому з компонентів.

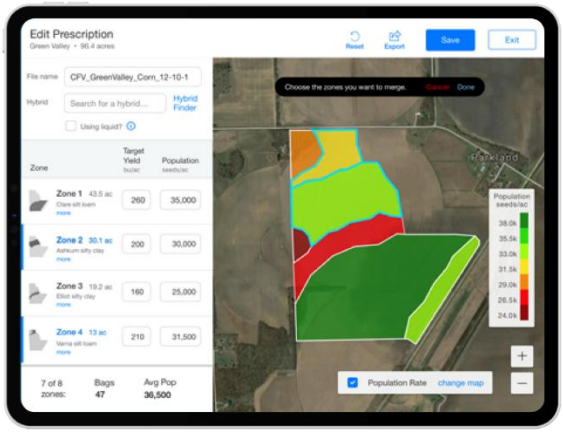

Рисунок 3 – Редагування посівів за допомогою FieldView

Усі вищезгадані системи працюють за різними методами та технологіями аналізу ґрунту. Розберемо їх окремо.

Моніторинг базових фізичних параметрів. З назви можна зрозуміти, що цей метод зосереджений на вимірюванні ключових фізичних характеристик ґрунту, таких як вологість, температура, електропровідність та pH. Цього можна досягнути за допомогою датчиків. Наприклад, датчики вологості використовують принцип зміни діелектричної проникності для визначення вмісту води в ґрунті, тобто, зміна рівня вологості змінює діелектричну проникність, що далі змінює ємність конденсатора [6]. Подібні датчики дозволяють оптимізувати зрошення та уникнути надмірного або недостатнього поливу. Температурні сенсори допомагають контролювати цикли росту культур та активність мікроорганізмів, які допомагають розкладати органіку та робити поживні речовини доступними для рослин, визначити найкращий час для посіву. Наразі досить популярним є поєднання датчиків вологості і температури в один. Датчики електропровідності вимірюють солоність ґрунту, що є критичним для запобігання засоленню та забезпечення здорового росту рослин. pH-сенсори допомагають визначити кислотність або лужність ґрунту, що впливає на доступність поживних речовин для рослин [7].

Аналіз хімічного складу ґрунту. Цей призначений для визначення концентрацій основних поживних елементів у ґрунті, таких як азот (N), фосфор (P) і калій (K). Для цього є NPK-сенсори, що електромагнітною індукцією та спектроскопією оцінюють вміст цих елементів, щоб потім фермери точно дозували добрива та уникали їх надмірного використання [8]. Якщо потрібен детальніший аналіз, наприклад, вміст більш ширшого спектру іонів, існують Іон-селективні електроди.

Комплексний моніторинг. Він об’єднує вимірювання як фізичних, так і хімічних параметрів ґрунту. Наприклад, платформа RapidMapper, розроблена в рамках проєкту «Intelligence for Soil I4S» у Німеччині, комбінує методи електропровідності, pH-потенціометрії, гамма-спектроскопії та інфрачервоної спектроскопії для детального аналізу ґрунту в реальному часі і передає їх через систему підтримки прийняття рішень [9]. Робота цієї системи можна побачити на рисунку 4.

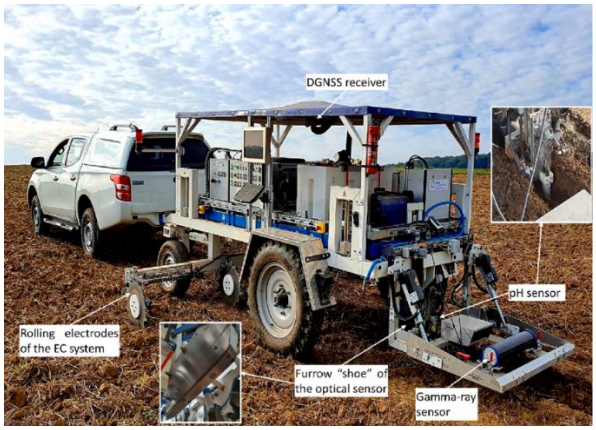

Рисунок 4 – Платформа RapidMapper в робочому режимі

Системи, що працюють, використовуючи IoT, можуть передавати інформацію різними способами.

Бездротові Wi-Fi-модулі забезпечують передачу даних з датчиків до хмарних сервісів або мобільних застосунків через локальну бездротову мережу. Як приклад реалізації впровадження Wi-Fi-модуля можна навести пристрій Soilmote від Zynect Sensors. У режимі реального часу, використовуючи спеціальний мобільний застосунок, користувач може бачити виміряні параметри вологості та температури ґрунту (рис. 5). Обмежений радіус дії стандартних Wi-Fi мереж робить їх менш придатними для використання на великих відкритих площах, де перевага надається технологіям з більшим радіусом покриття, тому це рішення підходить для теплиць, садів та домашніх рослин [10].

Рисунок 5 – Вигляд мобільного застосунку Soilmote

Стільникові мережі GSM, 4G, 5G. Там, де відсутнє Wi-Fi покриття використовують стільникові мережі, це актуально для віддалених сільськогосподарських ділянок. Наразі по усьому світу впроваджують мобільну мережу 5G, бо вона переважає по швидкості передачі даних і низької затримці сигналу. У 2022 був представлений повністю автономний трактор John Deere (рис. 6), здатний обробляти поля без участі людини, але завдяки потужному набору камер, сенсорів і бездротовій технології 5G, які дозволяють їй самостійно орієнтуватися на місцевості, приймати рішення та передавати дані в реальному часі до хмари через мобільну мережу [11].

Рисунок 6 – Автономний трактор John Deere і мобільний застосунок

Низькоенергетичні протоколи передачі LoRaWAN. LoRaWAN (Long Range Wide Area Network) використовує мережеву топологію «зірка зірок», де кінцеві пристрої зв'язуються з одним або кількома шлюзами, які потім передають повідомлення на центральний мережевий сервер через стандартні IP-з'єднання (рис. 7). Вони актуальні для сільського господарства завдяки великим географічним площам, які часто залучаються, та потребі в датчиках, що працюють від батарей, у віддалених місцях без легкого доступу до живлення [12].

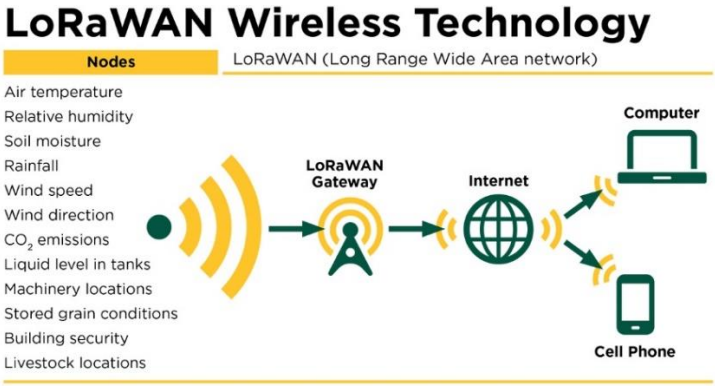

Рисунок 7 – Схема передачі даних у мережі LoRaWAN для аграрного моніторингу

Мережі ближнього радіусу дії Bluetooth та Zigbee. Ці технології особливо зручні в умовах, де немає потреби у великому радіусі дії: у теплицях або навіть на присадибній ділянці. Їх можна використовувати разом із простими датчиками вологості, температури, освітленості тощо. Такі сенсори зручні для інтеграції в системи розумного дому та автоматизації поливу [13]. Наприклад, на рисунку 8 зображено датчик, що використовує Bluetooth для аналізу ґрунту.

Рисунок 8 – Bluetooth-датчика ґрунту N335 із шлюзом

ВИСНОВКИ. У результаті дослідження було проаналізовано розповсюджені види систем автоматизації аналізу ґрунту на базі технології IoT, а саме: стаціонарні, мобільні та гібридні; перелічені їх переваги та недоліки. Розглянуто методи збору, обробки й передачі даних, які лежать в основі функціонування таких систем. Подальший розвиток та впровадження дозволять аграріям краще приймати обґрунтовані рішення або зовсім дозволити робити розвинути систем це за них щодо зрошення, використання добрив та інших агротехнічних заходів, що, в свою чергу, сприяє підвищенню врожайності, ефективному використанню ресурсів та зниженню впливу на довкілля.

ЛІТЕРАТУРА

1. Internet-of-Things (iot)-based smart agriculture: toward making the fields talk / M. Ayaz et al. *IEEE access*. 2019. Vol. 7.

2. Інтернет речей – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Інтернет_речей (дата звернення: 17.04.2025).

3. Soil moisture sensors for irrigation scheduling // Extension University of Minnesota. URL: https://extension.umn.edu/irrigation/soil-moisture-sensors-irrigation-scheduling (дата звернення: 17.04.2025)

4. Veris Technologies. URL: https://www.veristech.com/ (дата звернення: 17.04.2025).

5. Climate FieldView Україна. – URL: https://www.climatefieldview.com.ua/ (дата звернення: 17.04.2025).

5. 5 types of soil sensors - which is best for you? - renke. *Environment Monitoring Sensors Manufacturer*. URL: https://www.renkeer.com/5-types-soil-sensors/ (date of access: 17.04.2025).

6. Types of soil sensors. *Best Weather Station Manufacturer | Coda Sensors*. URL: https://www.codasensor.com/types-of-soil-sensors.html (date of access: 17.04.2025).

7. Soil NPK sensors - leading environmental sensor solutions for businesses | XPS. *Leading Environmental Sensor Solutions for Businesses | XPS*. URL: https://sz-xps.com/soil-npk-sensors/ (date of access: 17.04.2025).

8. Rapid in-field soil analysis of plant-available nutrients and pH for precision agriculture–a review / E. Najdenko et al. *Precision agriculture*. 2024. URL: https://doi.org/10.1007/s11119-024-10181-6 (date of access: 17.04.2025).

9. Soilmote - Wi-Fi + LoRaWAN - Zynect. URL: https://www.zynect.com/home/store/buy-soilmote/ (date of access: 17.04.2025).

10. John Deere. John Deere Unveils Fully Autonomous Tractor at CES 2022. URL: https://www.deere.com/en/our-company/digital-security/autonomous-tractor-reveal/ (date of access: 17.04.2025).

11. Basics of lora technology for crop and livestock management. *NDSU Agriculture*. URL: https://www.ndsu.edu/agriculture/extension/publications/basics-lora-technology-crop-and-livestock-management (date of access: 17.04.2025).

12. LoRa // Вікіпедія. URL: https://uk.wikipedia.org/wiki/LoRa (дата звернення: 17.04.2025).

13. TimerGarden. N335 2-in-1 Bluetooth Soil Sensor with Gateway. URL: https://timergarden.com/products/n335-2-in-1-bluetooth-soil-sensor-with-gateway (дата звернення: 17.04.2025).

Науковий керівник: *Хрустальова Софія Володимирівна, доцент кафедри КІТАР Харківського національного університету радіоелектроніки*

ДОДАТОК Б

Програмний код підключення датчика до мікроконтролера та передачі даних у BLYNK

```
#define BLYNK_TEMPLATE_ID "TMPL4oOFG8RvY"
#define BLYNK_TEMPLATE_NAME "system"
#define BLYNK_AUTH_TOKEN "PEmEBE2rcN9OcoQFHt3ccPM3MUJ0bQio"

#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

char ssid[] = "Wokwi-GUEST";
char pass[] = "";

typedef struct {
  uint16_t nitrogen;
  uint16_t phosphorus;
  uint16_t potassium;
  uint16_t ph_value;
  uint16_t conductivity;
  int16_t  temperature;
  uint16_t humidity;
} SensorData;

SensorData sensorValues;
```

```
// Blynk віртуальні піни
#define VPIN_N V0
#define VPIN_P V1
#define VPIN_K V2
#define VPIN_PH V3
#define VPIN_EC V6
#define VPIN_TEMP V4
#define VPIN_HUMI V5

BlynkTimer timer;

float getSerialInputFloat(const char* prompt) {
  Serial.print(prompt);
  while (Serial.available() == 0) {
    delay(100);
  }
  float value = Serial.parseFloat();

  while(Serial.available() > 0 && Serial.read() != '\n');
  Serial.println(value);
  return value;
}

bool readSensorDataFromSerial() {
  Serial.println("Введіть параметри ґрунту");

  float n_float = getSerialInputFloat("Азот (N, mg/kg): ");
  sensorValues.nitrogen = (uint16_t)n_float;

  float p_float = getSerialInputFloat("Фосфор (P, mg/kg): ");
```

```
  sensorValues.phosphorus = (uint16_t)p_float;

  float k_float = getSerialInputFloat("Калій (K, mg/kg): ");
  sensorValues.potassium = (uint16_t)k_float;

  float ph_float = getSerialInputFloat("pH: ");
  sensorValues.ph_value = (uint16_t)(ph_float * 100.0);

  float ec_float = getSerialInputFloat("Електропровідність (EC, µS/cm): ");
  sensorValues.conductivity = (uint16_t)ec_float;

  float temp_float = getSerialInputFloat("Температура °C: ");
  sensorValues.temperature = (int16_t)(temp_float * 10.0);

  float humi_float = getSerialInputFloat("Вологість %: ");
  sensorValues.humidity = (uint16_t)(humi_float * 10.0);

  Serial.println("------------------------------------------------");
  return true;
}

void sendSensorData() {

  if (readSensorDataFromSerial()) {
    // Конвертуємо значення (якщо потрібно) і виводимо
    float nVal = (float)sensorValues.nitrogen;
    float pVal = (float)sensorValues.phosphorus;
    float kVal = (float)sensorValues.potassium;
    float phVal = (float)sensorValues.ph_value / 100.0;
    float ecVal = (float)sensorValues.conductivity;
```

```
    float tempVal = (float)sensorValues.temperature / 10.0;
    float humiVal = (float)sensorValues.humidity / 10.0;

    Serial.println("Дані, введені вручну:");
    Serial.printf("N: %.1f mg/kg\n", nVal);
    Serial.printf("P: %.1f mg/kg\n", pVal);
    Serial.printf("K: %.1f mg/kg\n", kVal);
    Serial.printf("pH: %.2f\n", phVal);
    Serial.printf("EC: %.0f µS/cm\n", ecVal);
    Serial.printf("Температура: %.1f °C\n", tempVal);
    Serial.printf("Вологість: %.1f %%\n", humiVal);
    Serial.println("---------------------------------------");

    Blynk.virtualWrite(VPIN_N, nVal);
    Blynk.virtualWrite(VPIN_P, pVal);
    Blynk.virtualWrite(VPIN_K, kVal);
    Blynk.virtualWrite(VPIN_PH, phVal);
    Blynk.virtualWrite(VPIN_EC, ecVal);
    Blynk.virtualWrite(VPIN_TEMP, tempVal);
    Blynk.virtualWrite(VPIN_HUMI, humiVal);

  } else {
    Serial.println("Не вдалося отримати дані");
  }
}

void setup() {
  Serial.begin(9600);

  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
```

```
  timer.setInterval(500L, sendSensorData);
  Serial.println("Введіть значення");
}

void loop() {
  Blynk.run();
  timer.run();
}
```

ДОДАТОК В

Основний код програмного засобу системи автоматизації аналізу ґрунту у VS CODE

```
import streamlit as st
import requests
import pandas as pd
from data_loader import load_from_json, load_from_google_sheets
from analysis_logic import analyze_soil_conditions, recommend_suitable_crops, get_fertilizer_and_care_advice
from streamlit_echarts import st_echarts
import base64
import streamlit.components.v1 as components

st.set_page_config(
    page_title="Аналіз ґрунту",
    layout="wide",
    initial_sidebar_state="expanded"
)

def img_to_base64(image_path):
    try:
        with open(image_path, "rb") as img_file:
            return base64.b64encode(img_file.read()).decode('utf-8')
    except Exception as e:
        st.error(f"Помилка завантаження зображення '{image_path}': {str(e)}")
        return ""

def get_weather(lat, lon):
```

```
    try:
        API_KEY = "6be5db735b430e26e46729bd12d392f7"
                                                                    url =
f"http://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API
_KEY}&units=metric&lang=uk"
        response = requests.get(url, timeout=5)
        response.raise_for_status()
        data = response.json()

        weather_translations = {
            'clear': 'Ясно', 'clouds': 'Хмарно', 'rain': 'Дощ',
            'thunderstorm': 'Гроза', 'snow': 'Сніг', 'mist': 'Туман',
            'overcast clouds': 'Суцільна хмарність', 'few clouds': 'Невелика хмарність',
                'scattered clouds': 'Розсіяна хмарність', 'broken clouds': 'Хмарно з
прояснennями',
            'shower rain': 'Злива'
        }

        weather_desc_main = data['weather'][0]['main'].lower()
        weather_desc_detail = data['weather'][0]['description'].lower()

        translated_desc = weather_translations.get(weather_desc_detail,
                                    weather_translations.get(weather_desc_main,
                                        data['weather'][0]['description'].capitalize()))

        return {
            'temp': round(data['main']['temp']),
            'desc': translated_desc,
            'icon': data['weather'][0]['icon'],
            'humidity': data['main']['humidity'],
```

```
            'city': data['name']
        }
    except requests.exceptions.Timeout:
        st.warning("Не вдалося отримати погоду: час очікування вичерпано.")
        return None
    except requests.exceptions.RequestException as e:
        st.error(f"Помилка отримання погоди (мережа/API): {str(e)}")
        return None
    except Exception as e:
        st.error(f"Загальна помилка отримання погоди: {str(e)}")
        return None

css_styles = """
<style>
    body { margin: 0; padding: 0; font-family: Arial, sans-serif; }
    .header-container {
        position: relative; width: 100%; height: 150px; overflow: hidden;
    }
    .header-bg {
        position: absolute; width: 100%; height: 100%; object-fit: cover; z-index: 1;
    }
    .weather-widget {
        position: absolute; left: 20px; top: 15px;
        color: white; display: flex; align-items: center;
        background: rgba(0,0,0,0.5); border-radius: 10px;
        padding: 8px 12px; backdrop-filter: blur(4px);
        -webkit-backdrop-filter: blur(4px); z-index: 3;
    }
    .weather-widget img { margin-right: 10px; }
    .weather-city {
```

```
        font-size: 1.05em;
        font-weight: bold;
        margin-bottom: 4px;
        text-shadow: 1px 1px 2px rgba(0,0,0,0.6);
    }
    .weather-temp { font-weight: bold; font-size: 1.15em; }
    .weather-desc { font-size: 0.9em; }
    .weather-humidity { font-size: 0.9em; }
    .header-title {
        position: absolute; top: 50%; left: 50%;
        transform: translate(-50%, -50%); color: white;
        font-size: 28px; font-weight: bold;
        text-shadow: 2px 2px 4px rgba(0,0,0,0.8);
        background-color: rgba(0,0,0,0.5); padding: 10px 20px;
        border-radius: 6px; z-index: 3; text-align: center;
    }
</style>
"""

image_base64 = img_to_base64('assets/back1.png')
if not image_base64:
    image_base64 = img_to_base64('back1.png')

weather_data_display = None

if ('soil_data' in st.session_state and
    isinstance(st.session_state.soil_data, pd.DataFrame) and
    not st.session_state.soil_data.empty and
    'selected_record_index' in st.session_state and
    st.session_state.selected_record_index < len(st.session_state.soil_data)):
```

```
        current_data_for_header = st.session_state.soil_data.iloc[st.session_state.selected_record_index]
    lat_header = current_data_for_header.get('latitude')
    lon_header = current_data_for_header.get('longitude')

    if pd.notna(lat_header) and pd.notna(lon_header):
        weather_data_display = get_weather(lat_header, lon_header)
    else:
        st.info("Координати для поточного запису відсутні, погоду для заголовка не завантажено.")

weather_widget_html = ""
if weather_data_display:
    weather_widget_html = f"""
    <div class="weather-widget">
        <img src="http://openweathermap.org/img/wn/{weather_data_display['icon']}@2x.png" width="40">
        <div>
            {f'<div class="weather-city">{weather_data_display["city"]}</div>' if 'city' in weather_data_display else ''}
            <div class="weather-temp">{weather_data_display['temp']}°C</div>
            <div class="weather-desc">{weather_data_display['desc']}</div>
            <div class="weather-humidity">Вологість: {weather_data_display['humidity']}%</div>
        </div>
    </div>
    """
else:
```

```
    weather_widget_html = """
    <div class="weather-widget">
        <div style="padding: 5px;">Погода: дані відсутні</div>
    </div>
    """

final_html_header = f"""
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    {css_styles}
</head>
<body>
    <div class="header-container">
        <img class="header-bg" src="{image_base64}">
        {weather_widget_html}
    </div>
</body>
</html>
"""
components.html(final_html_header, height=150)

if 'soil_data' not in st.session_state:
    st.session_state.soil_data = pd.DataFrame()
if 'current_mode' not in st.session_state:
    st.session_state.current_mode = "monitoring"
if 'selected_record_index' not in st.session_state:
    st.session_state.selected_record_index = 0
if 'selected_crop_for_advice' not in st.session_state:
```

```
    st.session_state.selected_crop_for_advice = None
if 'last_weather_update' not in st.session_state:
    st.session_state.last_weather_update = None
if 'weather_data' not in st.session_state:
    st.session_state.weather_data = None
if 'data_loaded_message_shown' not in st.session_state:
    st.session_state.data_loaded_message_shown = False

def load_css(file_name):
    try:
        with open(file_name, "r", encoding="utf-8") as f:
            st.markdown(f'<style>{f.read()}</style>', unsafe_allow_html=True)
    except FileNotFoundError:
        st.warning(f"Файл стилів '{file_name}' не знайдено.")

load_css("assets/style.css")

def get_indicator_color(param, value):
    try:
        value = float(value)
        if pd.isna(value):
            return "#ccc"
    except (ValueError, TypeError):
        return "#ccc"

    param_lower = param.lower()

    if param_lower == "ph":
        if value < 3.0 or value > 9.0: return "#f44336"
        elif 4.5 <= value <= 6.0 or 7.5 <= value <= 8.5: return "#FDDD60"
```

```
        else: return "#91cc75"
    elif param_lower == "азот (n)":
        if value < 20 or value > 200: return "#f44336"
        elif 20 <= value <= 50 or 150 <= value <= 200: return "#FDDD60"
        else: return "#91cc75"
    elif param_lower == "фосфор (p)":
        if value < 5 or value > 80: return "#f44336"
        elif 5 <= value <= 15 or 60 <= value <= 80: return "#FDDD60"
        else: return "#91cc75"
    elif param_lower == "калій (k)":
        if value < 50 or value > 400: return "#f44336"
        elif 50 <= value <= 100 or 300 <= value <= 400: return "#FDDD60"
        else: return "#91cc75"
    elif param_lower == "вологість":
        if value < 10 or value > 90: return "#f44336"
        elif 10 <= value <= 30 or 80 <= value <= 90: return "#FDDD60"
        else: return "#91cc75"
    elif param_lower == "температура":
        if value < 0 or value > 45: return "#f44336"
        elif 0 <= value <= 10 or 35 <= value <= 45: return "#FDDD60"
        else: return "#91cc75"
    elif param_lower == "ec":
        if value < 800 or value >= 3500: return "#f44336"
        elif (800 <= value <= 1000) or (2500 <= value <= 3500): return "#FDDD60"
        else: return "#91cc75"
    return "#ccc"

def render_gauge(title, value, min_value, max_value, unit):
    try:
        if value is None or pd.isna(value):
```

```
            value_float = 0.0
            display_value = "N/A"
        else:
            value_float = float(value)
             display_value = f"{value_float:.1f}" if isinstance(value_float, float) and not title.lower() == "ec" else str(int(value_float)) if isinstance(value_float, float) and title.lower() == "ec" else str(value_float)
            if title.lower() == "ec" and display_value != "N/A":
                display_value = f"{int(value_float)}"
    except (ValueError, TypeError):
        value_float = 0.0
        display_value = "Error"

    try: min_val_float = float(min_value)
    except (ValueError, TypeError): min_val_float = 0.0
    try: max_val_float = float(max_value)
    except (ValueError, TypeError): max_val_float = 100.0

    range_val = max_val_float - min_val_float
    percentage = min(1, max(0, (value_float - min_val_float) / range_val)) if range_val != 0 and display_value != "N/A" else 0

      color = get_indicator_color(title, value_float if display_value != "N/A" else min_val_float -1)

    option = {
        "series": [{
            "type": "gauge",
            "startAngle": 180,
            "endAngle": 0,
```

```
        "min": min_val_float,
        "max": max_val_float,
        "axisLine": {
            "lineStyle": {
                "width": 20,
                "color": [[percentage, color], [1, "#eee"]],
                "cap": 'round'
            }
        },
        "pointer": {
            "show": False
        },
        "axisTick": { "show": False },
        "splitLine": { "show": False },
        "axisLabel": { "show": False },
        "detail": {
            "valueAnimation": True,
            "formatter": f"{display_value}{unit}" if display_value != "N/A" else "N/A",
            "color": "auto" if display_value != "N/A" else "#ccc",
            "fontSize": 20,
            "offsetCenter": [0, "-15%"],
            "fontWeight": "bold"
        },
         "data": [{"value": value_float if display_value != "N/A" else None, "name":
title}],
        "title": {
            "offsetCenter": [0, "30%"],
            "fontSize": 14,
            "fontWeight": "bold"
        },
```

```
            "radius": "90%"
        }]
    }
    st_echarts(option, height="220px")

def show_weather_widget_main_logic():
    try:
                    if not isinstance(st.session_state.soil_data, pd.DataFrame) or
st.session_state.soil_data.empty:
            return None

                                                        current_data =
st.session_state.soil_data.iloc[st.session_state.selected_record_index]
        lat = current_data.get('latitude')
        lon = current_data.get('longitude')

        if pd.isna(lat) or pd.isna(lon):
            return None

        API_KEY = "6be5db735b430e26e46729bd12d392f7"
        UNITS = "metric"
                                                            url =
f"http://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API
_KEY}&units={UNITS}&lang=uk"

        response = requests.get(url, timeout=5)
        response.raise_for_status()
        data = response.json()

        city_name = data.get('name', 'Невідоме місце')
```

```
        temp = round(data['main']['temp'], 1)
        weather_desc_api = data['weather'][0]['description']
        humidity = data['main']['humidity']
        icon_code = data['weather'][0]['icon']

        weather_translations = {
            'overcast clouds': 'Хмарно', 'clear sky': 'Ясно',
            'few clouds': 'Невелика хмарність', 'scattered clouds': 'Розсіяна хмарність',
            'broken clouds': 'Хмарно з проясненнями', 'shower rain': 'Злива',
            'rain': 'Дощ', 'thunderstorm': 'Гроза', 'snow': 'Сніг', 'mist': 'Туман'
        }
          weather_desc_translated = weather_translations.get(weather_desc_api.lower(),
weather_desc_api.capitalize())

        return {
            'city': city_name, 'temp': temp, 'desc': weather_desc_translated,
            'humidity': humidity, 'icon': icon_code, 'lat': lat, 'lon': lon
        }
    except requests.exceptions.Timeout:
        st.warning("Не вдалося оновити погоду: час очікування вичерпано.")
        return None
    except requests.exceptions.RequestException as e:
        st.error(f"Помилка оновлення погоди: {str(e)}")
        return None
    except Exception as e:
        st.error(f"Загальна помилка при оновленні погоди: {str(e)}")
        return None

if isinstance(st.session_state.soil_data, pd.DataFrame) and not
st.session_state.soil_data.empty:
```

```
                current_data_for_weather_update = st.session_state.soil_data.iloc[st.session_state.selected_record_index]
        current_coords = (current_data_for_weather_update.get('latitude'), current_data_for_weather_update.get('longitude'))

    if (st.session_state.last_weather_update != current_coords and
        pd.notna(current_coords[0]) and pd.notna(current_coords[1])):
        st.session_state.weather_data = show_weather_widget_main_logic()
        st.session_state.last_weather_update = current_coords
        #st.rerun()

st.markdown("Аналіз ґрунту, моніторинг та вибір оптимальних культур для вирощування")
st.markdown("---")

switch_mode = st.toggle(
    "Режим порад",
    value=(st.session_state.current_mode == "advice"),
     help="Увімкнено: Режим порад щодо вибору культур. Вимкнено: Режим моніторингу стану ґрунту.",
    key="mode_selection_toggle_labeled"
)

if switch_mode:
    st.session_state.current_mode = "advice"
else:
    st.session_state.current_mode = "monitoring"

st.sidebar.title("Налаштування та дії")
st.sidebar.header("Джерело даних")
```

```
data_source_type = st.sidebar.radio(
    "Оберіть тип джерела:",
    ('JSON файл', 'Google Sheets', 'Ввести дані вручну'),
    key="data_source_type_radio"
)

if data_source_type == 'JSON файл':
        uploaded_file = st.sidebar.file_uploader("Завантажте ваш JSON файл", type=["json"])
    if uploaded_file:
        with st.spinner("Завантаження даних..."):
            st.session_state.soil_data = load_from_json(uploaded_file)
            st.session_state.selected_record_index = 0
            st.session_state.selected_crop_for_advice = None
            st.session_state.data_loaded_message_shown = False
            if st.session_state.soil_data.empty:
                    st.sidebar.error("Помилка завантаження даних з JSON або файл порожній.")
            else:
                st.sidebar.success("Дані з JSON завантажено!")
                st.rerun()

elif data_source_type == 'Google Sheets':
    sheet_input = st.sidebar.text_input("Введіть ID або URL Google Sheets:")
    if st.sidebar.button("Завантажити з Google Sheets"):
        if sheet_input:
            with st.spinner("Завантаження даних..."):
                st.session_state.soil_data = load_from_google_sheets(sheet_input)
                st.session_state.selected_record_index = 0
                st.session_state.selected_crop_for_advice = None
```

```
            st.session_state.data_loaded_message_shown = False
            if st.session_state.soil_data.empty:
                st.sidebar.error("Помилка завантаження даних з Google Sheets або таблиця порожня.")
            else:
                st.sidebar.success("Дані з Google Sheets завантажено!")
                st.rerun()
        else:
            st.sidebar.warning("Ведіть коректне посилання або ID таблиці.")

elif data_source_type == 'Ввести дані вручну':
    st.sidebar.subheader("Введіть показники ґрунту:")
    current_manual_data_dict = {}
    if isinstance(st.session_state.soil_data, pd.DataFrame) and \
       not st.session_state.soil_data.empty and \
       st.session_state.selected_record_index < len(st.session_state.soil_data):
                                                    current_manual_data_dict = st.session_state.soil_data.iloc[st.session_state.selected_record_index].to_dict()

    col1_sidebar, col2_sidebar = st.sidebar.columns(2)
    default_lat, default_lon = 50.231766, 36.431977

     manual_ph = col1_sidebar.number_input("pH", min_value=3.0, max_value=9.0, value=None, step=0.1, format="%.1f")
        manual_n = col2_sidebar.number_input("Азот (N, ppm)", min_value=0, max_value=200, value=None, step=1)
        manual_p = col1_sidebar.number_input("Фосфор (P, ppm)", min_value=0, max_value=80, value=None, step=1)
        manual_k = col2_sidebar.number_input("Калій (K, ppm)", min_value=0, max_value=400, value=None, step=1)
```

```
        manual_hum = col1_sidebar.number_input("Вологість (%)", min_value=0, max_value=100, value=None, step=1)
    manual_temp = col2_sidebar.number_input("Температура (°C)", min_value=-10.0, max_value=50.0, value=None, step=0.1, format="%.1f")
        manual_ec = col1_sidebar.number_input("EC (µS/cm)", min_value=0, max_value=10000, value=None, step=10)
        manual_lat = col2_sidebar.number_input("Широта", value=default_lat, format="%.6f")
        manual_lon = col1_sidebar.number_input("Довгота", value=default_lon, format="%.6f")

    if st.sidebar.button("Використати введені дані"):
        required_fields = [manual_ph, manual_n, manual_p, manual_k, manual_hum, manual_temp, manual_ec]
        if any(field is None for field in required_fields):
            st.sidebar.error("Будь ласка, заповніть всі обов'язкові поля")
        else:
            manual_data = {
                "pH": manual_ph, "N_ppm": manual_n, "P_ppm": manual_p, "K_ppm": manual_k,
                "humidity_percent": manual_hum, "temperature_celsius": manual_temp, "ec": manual_ec,
                "latitude": manual_lat, "longitude": manual_lon,
                "sensor_id": "ManualInput", "timestamp": pd.Timestamp.now().isoformat()
            }
            st.session_state.soil_data = pd.DataFrame([manual_data])
            st.session_state.selected_record_index = 0
            st.session_state.selected_crop_for_advice = None
            st.session_state.data_loaded_message_shown = False
            st.sidebar.success("Введені дані застосовано!")
```

```
            st.rerun()

current_soil_data_row = pd.Series(dtype='object')

if isinstance(st.session_state.soil_data, pd.DataFrame) and not st.session_state.soil_data.empty:
    if not st.session_state.data_loaded_message_shown:
        st.session_state.data_loaded_message_shown = True

    st.sidebar.caption(f"Знайдено {len(st.session_state.soil_data)} запис(ів).")

    if len(st.session_state.soil_data) > 1:
        record_options_labels = []
        for i in range(len(st.session_state.soil_data)):
            ts_val = st.session_state.soil_data.iloc[i].get('timestamp', 'N/A')
            try:
                if isinstance(ts_val, str) and ts_val != 'N/A':
                    dt_obj = pd.to_datetime(ts_val)
                    if dt_obj.tzinfo: dt_obj = dt_obj.tz_convert(None)
                    formatted_ts = dt_obj.strftime('%Y-%m-%d %H:%M:%S')
                elif isinstance(ts_val, pd.Timestamp):
                    if ts_val.tzinfo: ts_val = ts_val.tz_convert(None)
                    formatted_ts = ts_val.strftime('%Y-%m-%d %H:%M:%S')
                else:
                    formatted_ts = str(ts_val)
            except Exception:
                formatted_ts = str(ts_val)
            record_options_labels.append(f"Запис {i+1} ({formatted_ts})")

        selected_option_label = st.sidebar.selectbox(
```

```
            "Оберіть запис для аналізу:",
            options=record_options_labels,
            index=st.session_state.selected_record_index,
            key="record_selector"
        )
        new_selected_index = record_options_labels.index(selected_option_label)
        if st.session_state.selected_record_index != new_selected_index:
            st.session_state.selected_record_index = new_selected_index
            st.session_state.selected_crop_for_advice = None
            st.rerun()
                                                    current_soil_data_row =
st.session_state.soil_data.iloc[st.session_state.selected_record_index]

    elif len(st.session_state.soil_data) == 1:
        current_soil_data_row = st.session_state.soil_data.iloc[0]
        st.session_state.selected_record_index = 0

    if not current_soil_data_row.empty:
        if st.session_state.current_mode == "monitoring":
            st.header("Моніторинг стану ґрунту")

            st.subheader("Показники ґрунту")
            cols_top_row = st.columns(4)
                                with cols_top_row[0]: render_gauge("Температура",
current_soil_data_row.get("temperature_celsius"), -10, 50, " °C")
                                  with cols_top_row[1]: render_gauge("Вологість",
current_soil_data_row.get("humidity_percent"), 0, 100, "%")
             with cols_top_row[2]: render_gauge("pH", current_soil_data_row.get("pH"),
3, 9, "")
```

```
        with cols_top_row[3]: render_gauge("EC", current_soil_data_row.get("ec"), 0, 4000, " µS/cm")

        st.markdown("<br>", unsafe_allow_html=True)
        cols_bottom_row = st.columns(3)
                        with cols_bottom_row[0]: render_gauge("Азот (N)", current_soil_data_row.get("N_ppm"), 0, 200, " ppm")
                    with cols_bottom_row[1]: render_gauge("Фосфор (P)", current_soil_data_row.get("P_ppm"), 0, 100, " ppm")
                        with cols_bottom_row[2]: render_gauge("Калій (K)", current_soil_data_row.get("K_ppm"), 0, 400, " ppm")

        analysis_result = analyze_soil_conditions(current_soil_data_row)
        st.subheader("Оцінка стану ґрунту")
        st.markdown(analysis_result)

        if 'latitude' in current_soil_data_row and 'longitude' in current_soil_data_row:
            lat_val = current_soil_data_row.get('latitude')
            lon_val = current_soil_data_row.get('longitude')

            if pd.notna(lat_val) and pd.notna(lon_val):
                try:
                    lat = pd.to_numeric(lat_val, errors='coerce')
                    lon = pd.to_numeric(lon_val, errors='coerce')

                    if pd.notna(lat) and pd.notna(lon):
                        if -90 <= lat <= 90 and -180 <= lon <= 180:
                                if st.sidebar.checkbox("Показати розташування на карті", value=True, key="show_map_checkbox"):
                                    st.header("Розташування датчика")
```

```
                    map_df = pd.DataFrame({'lat': [lat], 'lon': [lon]})
                    st.map(map_df, zoom=13)
                else:
                 st.sidebar.warning("Некоректні координати для карти (за межами діапазону).")
            except Exception as e:
                st.sidebar.error(f"Помилка при обробці координат для карти: {e}")

    elif st.session_state.current_mode == "advice":
        st.header("Підбір культур та поради")
        recommended = recommend_suitable_crops(current_soil_data_row)

        if not recommended:
            st.warning("Не вдалося підібрати відповідні культури для поточних умов ґрунту.")
        else:
            st.subheader(f"Рекомендовані культури ({len(recommended)}):")
            categories = {}
            for crop_info in recommended:
                if isinstance(crop_info, dict):
                    name = crop_info.get('name', 'Невідома культура')
                    category = crop_info.get('category', 'Інше')
                elif isinstance(crop_info, str):
                    name = crop_info.split('(')[0].strip() if '(' in crop_info else crop_info
                    category = 'Інше'
                    if '(' in crop_info and '→' in crop_info:
                        try:
                            category = crop_info.split('(')[1].split('→')[0].strip(' )')
                        except:
                            pass
```

```
                else:
                    name = str(crop_info)
                    category = 'Інше'

                if category not in categories: categories[category] = []
                categories[category].append(name)

            for category, crops_in_cat in categories.items():
                with st.expander(f"{category} ({len(crops_in_cat)})"):
                    for crop_name_advice in crops_in_cat:
                                                        if     st.button(crop_name_advice,
key=f"crop_{category}_{crop_name_advice.replace(' ','_')}"):
                            st.session_state.selected_crop_for_advice = crop_name_advice

            if st.session_state.selected_crop_for_advice:
                st.markdown("---")
               st.subheader(f"Поради для: {st.session_state.selected_crop_for_advice}")
                advice = get_fertilizer_and_care_advice(
                    st.session_state.selected_crop_for_advice,
                    current_soil_data_row
                )
                st.markdown(advice)

elif data_source_type != 'Ввести дані вручну':
    st.info("Завантажте дані про стан ґрунту або введіть їх вручну для початку аналізу.")
```

ДОДАТОК Г

Код логіки аналізу програмного засобу системи автоматизації аналізу ґрунту у VS CODE

```
import pandas as pd
import joblib
import os

CROP_DATABASE = {
   "Grains": {
      "Пшениця озима": {
         "category": "Зернові",
         "subcategory": "Зернові культури",
         "pH_optimal": (5.5, 7.5),
         "N_ppm_optimal": (100, 180),
         "P_ppm_optimal": (40, 70),
         "K_ppm_optimal": (100, 200),
         "EC_optimal": (1000, 2500),
             },
      "Кукурудза на зерно": {
         "category": "Зернові",
         "subcategory": "Технічні культури",
         "pH_optimal": (5.8, 7.0),
         "N_ppm_optimal": (120, 200),
         "P_ppm_optimal": (50, 80),
         "K_ppm_optimal": (120, 220),
         "EC_optimal": (1000, 1700),
             },
      "Ячмінь": {
```

```
        "category": "Зернові",
        "subcategory": "Зернові культури",
        "pH_optimal": (6.0, 7.5),
        "N_ppm_optimal": (80, 150),
        "P_ppm_optimal": (30, 60),
        "K_ppm_optimal": (90, 180),
        "EC_optimal": (1500, 3500),
    }
},
"Vegetables": {
    "Томати (відкритий ґрунт)": {
        "category": "Овочеві",
        "subcategory": "Пасльонові",
        "pH_optimal": (6.0, 6.8),
        "N_ppm_optimal": (100, 150),
        "P_ppm_optimal": (60, 100),
        "K_ppm_optimal": (150, 250),
        "EC_optimal": (1500, 2500),
            },
    "Огірки": {
        "category": "Овочеві",
        "subcategory": "Гарбузові",
        "pH_optimal": (6.0, 7.0),
        "N_ppm_optimal": (120, 180),
        "P_ppm_optimal": (50, 90),
        "K_ppm_optimal": (150, 250),
        "EC_optimal": (1500, 2500),
            },
    "Картопля": {
        "category": "Овочеві",
```

```
    "subcategory": "Коренеплоди",
    "pH_optimal": (5.0, 6.5),
    "N_ppm_optimal": (100, 180),
    "P_ppm_optimal": (80, 150),
    "K_ppm_optimal": (150, 300),
    "EC_optimal": (1500, 2000),
        },
"Морква": {
    "category": "Овочеві",
    "subcategory": "Коренеплоди",
    "pH_optimal": (6.0, 7.0),
    "N_ppm_optimal": (80, 140),
    "P_ppm_optimal": (50, 100),
    "K_ppm_optimal": (120, 240),
    "EC_optimal": (1000, 2000),
},
"Буряк столовий": {
    "category": "Овочеві",
    "subcategory": "Коренеплоди",
    "pH_optimal": (6.0, 7.5),
    "N_ppm_optimal": (90, 150),
    "P_ppm_optimal": (60, 100),
    "K_ppm_optimal": (150, 250),
    "EC_optimal": (1500, 3000),
},
"Гарбуз": {
    "category": "Гарбузові",
    "subcategory": "Гарбузові",
    "pH_optimal": (6.0, 7.0),
    "N_ppm_optimal": (100, 160),
```

```
        "P_ppm_optimal": (70, 120),
        "K_ppm_optimal": (150, 280),
        "EC_optimal": (1500, 2500),
            },
    "Диня": {
        "category": "Гарбузові",
        "subcategory": "Гарбузові",
        "pH_optimal": (6.0, 7.0),
        "N_ppm_optimal": (100, 150),
        "P_ppm_optimal": (60, 100),
        "K_ppm_optimal": (150, 250),
        "EC_optimal": (2000, 3500),
            },
    "Кавун": {
        "category": "Гарбузові",
        "subcategory": "Гарбузові",
        "pH_optimal": (6.0, 7.0),
        "N_ppm_optimal": (90, 140),
        "P_ppm_optimal": (50, 90),
        "K_ppm_optimal": (120, 220),
        "EC_optimal": (1500, 2600),
    }
},
"Legumes": {
    "Соя": {
        "category": "Бобові",
        "subcategory": "Зернобобові",
        "pH_optimal": (6.0, 7.0),
        "N_ppm_optimal": (20, 60),
        "P_ppm_optimal": (40, 70),
```

```
        "K_ppm_optimal": (80, 150),
        "EC_optimal": (1000, 2500),
    },
    "Горох": {
        "category": "Бобові",
        "subcategory": "Зернобобові",
        "pH_optimal": (6.0, 7.5),
        "N_ppm_optimal": (20, 50),
        "P_ppm_optimal": (30, 60),
        "K_ppm_optimal": (70, 140),
        "EC_optimal": (800, 1500),
            }
},
"Berries": {
    "Полуниця": {
        "category": "Ягідні",
        "subcategory": "Ягоди",
        "pH_optimal": (5.5, 6.5),
        "N_ppm_optimal": (80, 120),
        "P_ppm_optimal": (50, 80),
        "K_ppm_optimal": (120, 200),
        "EC_optimal": (800, 2000),
            },
    "Малина": {
        "category": "Ягідні",
        "subcategory": "Ягоди",
        "pH_optimal": (5.5, 6.5),
        "N_ppm_optimal": (90, 140),
        "P_ppm_optimal": (40, 70),
        "K_ppm_optimal": (130, 220),
```

```
            "EC_optimal": (800, 1500),
        }
    },
    "Industrial": {
        "Соняшник": {
            "category": "Технічні",
            "subcategory": "Олійні",
            "pH_optimal": (6.0, 7.5),
            "N_ppm_optimal": (50, 100),
            "P_ppm_optimal": (40, 70),
            "K_ppm_optimal": (150, 250),
            "EC_optimal": (1500, 3000),
                },
        "Ріпак": {
            "category": "Технічні",
            "subcategory": "Олійні",
            "pH_optimal": (6.0, 7.0),
            "N_ppm_optimal": (120, 180),
            "P_ppm_optimal": (50, 90),
            "K_ppm_optimal": (150, 250),
            "EC_optimal": (1500, 2500),
        }
    }
}

ML_COMPONENTS = {}
MODELS_DIR = "ml_models"
SHARED_CROP_ENCODER_PATH = os.path.join(MODELS_DIR,
"crop_encoder_shared.joblib")
MODEL_PARAMETERS = ["N", "P", "K", "pH", "EC"]
```

```
parameters_to_train_config = {
   "N": {"optimal_key": "N_ppm_optimal", "current_key": "current_N_ppm", "unit": " ppm", "full_name": "Азот (N)"},
   "P": {"optimal_key": "P_ppm_optimal", "current_key": "current_P_ppm", "unit": " ppm", "full_name": "Фосфор (P)"},
   "K": {"optimal_key": "K_ppm_optimal", "current_key": "current_K_ppm", "unit": " ppm", "full_name": "Калій (K)"},
    "pH": {"optimal_key": "pH_optimal", "current_key": "current_pH", "unit": "", "full_name": "Кислотність (pH)"},
    "EC": {"optimal_key": "EC_optimal", "current_key": "current_EC", "unit": " µS/cm", "full_name": "Електропровідність (EC)"}
}

def load_all_ml_components():
   global ML_COMPONENTS, ARE_ALL_MODELS_LOADED
   all_loaded_successfully = True
   if os.path.exists(SHARED_CROP_ENCODER_PATH):
      try:
                                    ML_COMPONENTS["crop_encoder_shared"] = joblib.load(SHARED_CROP_ENCODER_PATH)
      except Exception as e:
         print(f"Помилка завантаження спільного crop_encoder: {e}")
         all_loaded_successfully = False
         ML_COMPONENTS["crop_encoder_shared"] = None
   else:
                  print(f"Попередження: Файл спільного crop_encoder '{SHARED_CROP_ENCODER_PATH}' не знайдено.")
      all_loaded_successfully = False
      ML_COMPONENTS["crop_encoder_shared"] = None
```

```
    for param_code in MODEL_PARAMETERS:
        model_p = os.path.join(MODELS_DIR, f"{param_code.lower()}_model.joblib")
                                        encoder_p        =        os.path.join(MODELS_DIR,
f"{param_code.lower()}_need_encoder.joblib")
        param_components_loaded = True
        if not os.path.exists(model_p):
            param_components_loaded = False
        if not os.path.exists(encoder_p):
            param_components_loaded = False
        if param_components_loaded:
            try:
                                ML_COMPONENTS[f"{param_code.lower()}_model"]    =
joblib.load(model_p)
                        ML_COMPONENTS[f"{param_code.lower()}_need_encoder"]   =
joblib.load(encoder_p)
            except Exception as e:
                print(f"Помилка завантаження ML компонентів для {param_code}: {e}")
                all_loaded_successfully = False
                ML_COMPONENTS[f"{param_code.lower()}_model"] = None
                ML_COMPONENTS[f"{param_code.lower()}_need_encoder"] = None
        else:
            all_loaded_successfully = False
            ML_COMPONENTS[f"{param_code.lower()}_model"] = None
            ML_COMPONENTS[f"{param_code.lower()}_need_encoder"] = None


    if not all_loaded_successfully:
            print("\nУВАГА: Не всі ML компоненти були успішно завантажені. ML
поради можуть бути неповними або недоступними.")
```

```
    ARE_ALL_MODELS_LOADED = all_loaded_successfully
    return all_loaded_successfully

ARE_ALL_MODELS_LOADED = False
load_all_ml_components()

def flatten_crop_database():
    flat_db = {}
    for category_data in CROP_DATABASE.values():
        for crop_name, params in category_data.items():
            flat_db[crop_name] = params
    return flat_db

def analyze_soil_conditions(soil_data_row):
    if soil_data_row.empty: return "Немає даних для аналізу."
    status_parts = []
    ph_val = soil_data_row.get('pH')
    if ph_val is not None:
        try:
            ph = float(ph_val)
            if ph < 5.0: status_parts.append(f"- pH: {ph:.1f} (Дуже кислий)")
            elif ph < 6.0: status_parts.append(f"- pH: {ph:.1f} (Кислий)")
                elif ph <= 7.5: status_parts.append(f"- pH: {ph:.1f} (Нейтральний до слаболужного)")
            else: status_parts.append(f"- pH: {ph:.1f} (Лужний)")
              except ValueError: status_parts.append(f"- pH: некоректне значення ({ph_val})")
    else: status_parts.append("- pH: дані відсутні")

    ec_val = soil_data_row.get('ec')
```

```
    if ec_val is not None:
        try:
            ec = float(ec_val)
            if ec < 200: status_parts.append(f"- EC: {ec:.0f} µS/cm (Дуже низька)")
            elif ec < 800: status_parts.append(f"- EC: {ec:.0f} µS/cm (Низька)")
            elif ec <= 2500: status_parts.append(f"- EC: {ec:.0f} µS/cm (Оптимальна)")
            elif ec <= 5000: status_parts.append(f"- EC: {ec:.0f} µS/cm (Підвищена)")
             else: status_parts.append(f"- EC: {ec:.0f} µS/cm (Занадто висока, можливе засолення)")
                except ValueError: status_parts.append(f"- EC: некоректне значення ({ec_val})")
    else: status_parts.append("- EC: дані відсутні")

    for param_key_soil, default_low, default_medium, display_name in [
        ('N_ppm', 80, 150, 'Азот (N)'), ('P_ppm', 30, 60, 'Фосфор (P)'), ('K_ppm', 100, 200, 'Калій (K)')
    ]:
        val = soil_data_row.get(param_key_soil)
        if val is not None:
            try:
                nutrient_val = float(val)
                    if nutrient_val < default_low: status_parts.append(f"- {display_name}: {nutrient_val} ppm (Низький)")
                        elif nutrient_val <= default_medium: status_parts.append(f"- {display_name}: {nutrient_val} ppm (Середній)")
                    else: status_parts.append(f"- {display_name}: {nutrient_val} ppm (Високий)")
                except ValueError: status_parts.append(f"- {display_name}: некоректне значення ({val})")
        else: status_parts.append(f"- {display_name}: дані відсутні")
```

```
    return "\n" + "\n".join(status_parts)


def recommend_suitable_crops(soil_data_row, num_recommendations=5):
    if soil_data_row.empty: return [{"name": "Пшениця озима", "category": "Зернові",
"subcategory": "Зернові культури", "score": 5},
                                    {"name": "Огірки", "category": "Овочеві", "subcategory":
"Гарбузові", "score": 4}]
    ph_val = soil_data_row.get('pH')
    n_ppm_val = soil_data_row.get('N_ppm')
    p_ppm_val = soil_data_row.get('P_ppm')
    k_ppm_val = soil_data_row.get('K_ppm')
    ec_val = soil_data_row.get('ec')
    temp_val = soil_data_row.get('temperature_celsius')
    humidity_val = soil_data_row.get('humidity_percent')
    flat_db = flatten_crop_database()
    scored_crops = []
    for crop_name, params in flat_db.items():
        score = 0
        try:
            if ph_val is not None and params.get("pH_optimal"):
                ph_min, ph_max = params["pH_optimal"]
                current_ph = float(ph_val)
                if ph_min <= current_ph <= ph_max: score += 3
                elif (ph_min - 0.5) <= current_ph <= (ph_max + 0.5): score += 1
                else: continue
                  for nutrient_val_soil, opt_key_db in [(n_ppm_val, "N_ppm_optimal"),
(p_ppm_val, "P_ppm_optimal"), (k_ppm_val, "K_ppm_optimal")]:
                if nutrient_val_soil is not None and params.get(opt_key_db):
                    opt_min_n, opt_max_n = params[opt_key_db]
                    current_nutrient = float(nutrient_val_soil)
```

```
                if opt_min_n <= current_nutrient <= opt_max_n: score += 2
                 elif (opt_min_n * 0.7) <= current_nutrient <= (opt_max_n * 1.3): score
+= 1

        if ec_val is not None and params.get("EC_optimal"):
            ec_min, ec_max = params["EC_optimal"]
            current_ec = float(ec_val)
            if ec_min <= current_ec <= ec_max: score += 2
            elif (ec_min * 0.7) <= current_ec <= (ec_max * 1.3): score += 1
            elif current_ec > ec_max * 1.5: score -= 1

        if temp_val is not None:
            current_temp = float(temp_val)
            if crop_name in ["Картопля", "Ріпак"] and current_temp > 25: score -= 1
              elif crop_name in ["Томати (відкритий ґрунт)", "Кукурудза на зерно",
"Соняшник"] and current_temp < 15: score -= 1
        if humidity_val is not None:
            current_humidity = float(humidity_val)
            if crop_name in ["Огірки"] and current_humidity < 60: score -=1
            elif crop_name in ["Соняшник"] and current_humidity > 75 : score -=1
      except ValueError: continue
            if score > 0: scored_crops.append({"name": crop_name, "category":
params.get("category", "Інше"), "subcategory": params.get("subcategory", "Інше"),
"score": score})
   scored_crops.sort(key=lambda x: (-x["score"], x["category"], x["subcategory"]))
   return scored_crops[:num_recommendations] if scored_crops else []

def get_ml_simplified_advice(param_code, predicted_category_raw,
current_value_float, opt_min, opt_max, final_crop_name_for_encoding,
param_config):
```

```
    """Генерація спрощених текстів для таблиці на основі ML прогнозу."""
    full_param_name_genitive = param_config.get("full_name", param_code)
    unit = param_config.get("unit", "")
    current_display = f"{current_value_float:.1f} {unit}"
    optimal_display = f"{opt_min:.1f}-{opt_max:.1f} {unit}"

    status_col2 = predicted_category_raw

      action_col3 = f"Поточний показник: {current_display}. Оптимальний для '{final_crop_name_for_encoding}': {optimal_display}. "

    if param_code in ["N", "P", "K"]:
        if predicted_category_raw == "Висока потреба":
                action_col3 += f"Рекомендується значно збільшити внесення {full_param_name_genitive}."
        elif predicted_category_raw == "Середня потреба":
                    action_col3 += f"Рекомендується внести додатково {full_param_name_genitive}."
        elif predicted_category_raw == "Низька потреба / Оптимум":
            status_col2 = "Рівень оптимальний"
            action_col3 += "Додаткове внесення зазвичай не потрібне."
        elif predicted_category_raw == "Надлишок / Потребує уваги":
            status_col2 = "Невеликий надлишок"
            action_col3 += f"Уникайте внесення {full_param_name_genitive}."
        elif predicted_category_raw == "Значний надлишок / Застереження":
            status_col2 = "Значний надлишок"
                action_col3 += f"Категорично не рекомендується вносити {full_param_name_genitive}, можлива токсичність."

    elif param_code == "pH":
```

```
        if "Корекція" in predicted_category_raw:
            if "кислий" in predicted_category_raw.lower():
                action_col3 += "Рекомендується вапнування."
                status_col2 = "Кислий ґрунт (потребує корекції)"
            elif "лужний" in predicted_category_raw.lower():
                action_col3 += "Рекомендується підкислення ґрунту."
                status_col2 = "Лужний ґрунт (потребує корекції)"
        elif predicted_category_raw == "Оптимальний":
            action_col3 += "Корекція pH не потрібна."
            status_col2 = "Кислотність оптимальна"
        else:
             action_col3 += "Оцініть необхідність корекції pH."


    elif param_code == "EC":
         if predicted_category_raw == "Висока потреба" or predicted_category_raw ==
"Низька потреба":
            status_col2 = "Низька електропровідність"
            action_col3 += "Значення занадто низьке. Рекомендується внести добрива
або покращити органічне живлення."
        elif predicted_category_raw == "Низька потреба" or predicted_category_raw ==
"Оптимальна":
            status_col2 = "Електропровідність оптимальна"
            action_col3 += "Живлення в межах норми. Додаткові дії не потрібні."
        elif predicted_category_raw == "Надлишок / Потребує уваги":
            status_col2 = "Висока електропровідність"
           action_col3 += "Рекомендується обмежити добрива. Можливе накопичення
солей."
        elif predicted_category_raw == "Висока / Застереження":
            status_col2 = "Дуже висока електропровідність"
```

```
            action_col3 += "Рекомендується промивання ґрунту. Є ризик засолення або токсичності."
        else:
            action_col3 += "Можлива потреба у корекції."

    return status_col2, action_col3

def get_ml_prediction_for_parameter(param_code, crop_name_query, current_value, db_flat):
    model_key = f"{param_code.lower()}_model"
    need_encoder_key = f"{param_code.lower()}_need_encoder"
    crop_encoder_shared = ML_COMPONENTS.get("crop_encoder_shared")
    error_status_prefix = "Помилка ML"

    if not ARE_ALL_MODELS_LOADED: return error_status_prefix, "ML компоненти не завантажені."
    if not crop_encoder_shared: return error_status_prefix, "Кодувальник культур не завантажений."
    if not ML_COMPONENTS.get(model_key): return error_status_prefix, f"Модель для {param_code} не завантажена."
    if not ML_COMPONENTS.get(need_encoder_key): return error_status_prefix, f"Кодувальник потреб для {param_code} не завантажений."

    final_crop_name_for_encoding = crop_name_query
    if crop_name_query not in crop_encoder_shared.classes_:
        cleaned_name = crop_name_query.split('(')[0].strip()
        if cleaned_name in crop_encoder_shared.classes_:
            final_crop_name_for_encoding = cleaned_name
        else: return f"Культура невідома", f"Культура '{crop_name_query}' не відома моделі."
```

```
    if final_crop_name_for_encoding not in db_flat:
        return f"Дані відсутні", f"Культура '{final_crop_name_for_encoding}' не знайдена в базі даних."

    param_config = parameters_to_train_config.get(param_code)
    if not param_config: return f"Конфіг відсутній", f"Немає конфігурації для параметра {param_code}."

    optimal_key_db = param_config["optimal_key"]
    if optimal_key_db not in db_flat[final_crop_name_for_encoding]:
        return f"Опт. дані відсутні", f"Немає оптимальних даних '{optimal_key_db}' для '{final_crop_name_for_encoding}'."

    opt_min, opt_max = db_flat[final_crop_name_for_encoding][optimal_key_db]

    if current_value is None: return f"Значення відсутнє", "Поточне значення не надано."
    try: current_value_float = float(current_value)
    except (ValueError, TypeError): return f"Некоректне знач.", f"Некоректне значення: '{current_value}'."

    try: crop_encoded_val = crop_encoder_shared.transform([final_crop_name_for_encoding])[0]
    except ValueError: return f"Кодування культури", f"Не вдалося закодувати '{final_crop_name_for_encoding}'."

    current_key_model = param_config["current_key"]
    optimal_min_key_model = f"{param_code}_optimal_min"
    optimal_max_key_model = f"{param_code}_optimal_max"
```

```
        features_df = pd.DataFrame([[crop_encoded_val, current_value_float, float(opt_min), float(opt_max)]],
                                   columns=["crop_name_encoded", current_key_model, optimal_min_key_model, optimal_max_key_model])
    model = ML_COMPONENTS[model_key]
    need_encoder = ML_COMPONENTS[need_encoder_key]

    try:
        prediction_encoded = model.predict(features_df)
                                                predicted_category_raw = need_encoder.inverse_transform(prediction_encoded)[0]
        status_col2, action_col3 = get_ml_simplified_advice(
            param_code, predicted_category_raw, current_value_float,
            opt_min, opt_max, final_crop_name_for_encoding, param_config
        )
        return status_col2, action_col3
    except Exception as e:
                        print(f"Помилка ML прогнозу для {param_code} ({final_crop_name_for_encoding}): {e}")
        return f"Помилка прогнозу", f"Помилка ML для {param_config.get('full_name', param_code)}."

def get_rule_based_simplified_advice(param_code, current_value_float, opt_min, opt_max, final_crop_name_for_encoding, param_config):
    """Генерація спрощених текстів для таблиці на основі правил."""
    full_param_name_genitive = param_config.get("full_name", param_code)
    unit = param_config.get("unit", "")
    current_display = f"{current_value_float:.1f} {unit}"
    optimal_display = f"{opt_min:.1f}-{opt_max:.1f} {unit}"
```

```
    status_col2 = "Стан не визначено"
      action_col3 = f"Поточний показник: {current_display}. Оптимальний для '{final_crop_name_for_encoding}': {optimal_display}. "

    if param_code == "pH":
        if current_value_float < opt_min:
            status_col2 = "Кислий ґрунт"
            action_col3 += "Рекомендується вапнування."
        elif current_value_float > opt_max:
            status_col2 = "Лужний ґрунт"
            action_col3 += "Рекомендується підкислення ґрунту."
        else:
            status_col2 = "Кислотність оптимальна"
            action_col3 += "Корекція pH не потрібна."
    elif param_code == "EC":
        if current_value_float < opt_min:
            status_col2 = "Низька електропровідність"
            action_col3 += "Значення занадто низьке. Рекомендується внести добрива або покращити живлення."
        elif current_value_float > opt_max * 2:
            status_col2 = "Дуже висока електропровідність"
              action_col3 += "Критично високий рівень! Можливе сильне засолення ґрунту. Рекомендується промивання ґрунту та обмеження добрив."
        elif current_value_float > opt_max:
            status_col2 = "Висока електропровідність"
           action_col3 += "Рекомендується обмежити добрива. Можливе накопичення солей."
        else:
            status_col2 = "Електропровідність оптимальна"
            action_col3 += "Рівень оптимальний. Живлення задовільне."
```

```
    return status_col2, action_col3

def get_fertilizer_and_care_advice(crop_name_input, soil_data_row):
    flat_db = flatten_crop_database()
    db_crop_name = crop_name_input
    if crop_name_input not in flat_db:
        clean_name = crop_name_input.split('(')[0].strip()
        if clean_name in flat_db: db_crop_name = clean_name
        else: return f"Інформація по культурі '**{crop_name_input}**' відсутня в базі даних."

    crop_info = flat_db[db_crop_name]
    advice_markdown_parts = []

    #Таблиця Аналіз ґрунту та рекомендації
    soil_recommendations_table_rows = []
            param_values_from_soil = {"pH": soil_data_row.get('pH'), "N": soil_data_row.get('N_ppm'),
                        "P": soil_data_row.get('P_ppm'), "K": soil_data_row.get('K_ppm'), "EC": soil_data_row.get('ec')}

    for param_code in MODEL_PARAMETERS:
        current_soil_val = param_values_from_soil.get(param_code)
        param_config = parameters_to_train_config.get(param_code)

                        param_name_for_column = param_config.get("full_name", param_code).replace("y (", " (").replace("ість (", "ість (")
        if param_code == "pH": param_name_for_column = "Кислотність (pH)"
                    else: param_name_for_column = param_config.get("full_name", param_code).split(" ")[0] + f" ({param_code})"
```

```
        status_col2_display = "N/A"
        action_col3_display = "N/A"

        ml_status_col2, ml_action_col3 = None, None
                                        if ARE_ALL_MODELS_LOADED and
ML_COMPONENTS.get(f"{param_code.lower()}_model"):
            ml_status_col2, ml_action_col3 = get_ml_prediction_for_parameter(
                param_code, db_crop_name, current_soil_val, flat_db
            )

        is_ml_successful = ml_status_col2 and ml_action_col3 and \
                           not any(err_indicator in ml_status_col2.lower() for err_indicator in
["помилка", "невідоме", "немає даних", "неможливо", "недоступний",
"відсутній"])

        if is_ml_successful:
            status_col2_display = ml_status_col2
            action_col3_display = ml_action_col3
        else:
                                        if current_soil_val is not None and
crop_info.get(param_config["optimal_key"]):
                opt_min_rule, opt_max_rule = crop_info[param_config["optimal_key"]]
                try:
                    current_soil_val_float_rule = float(current_soil_val)
                    rule_status_col2, rule_action_col3 = get_rule_based_simplified_advice(
                        param_code, current_soil_val_float_rule, opt_min_rule, opt_max_rule,
                        db_crop_name, param_config
                    )
                    status_col2_display = rule_status_col2
```

```
                action_col3_display = rule_action_col3
            except ValueError:
                status_col2_display = f"Некоректні дані"
                        action_col3_display = f"Не вдалося обробити значення '{current_soil_val}'."
        else:
            status_col2_display = "Немає даних для правила"
            action_col3_display = f"Неможливо дати пораду."

        soil_recommendations_table_rows.append(f"| {param_name_for_column} | {status_col2_display} | {action_col3_display} |")

    if soil_recommendations_table_rows:
        advice_markdown_parts.append(f"### Аналіз ґрунту та рекомендації")
        advice_markdown_parts.append("| Параметр | Стан | Рекомендовані дії |")
        advice_markdown_parts.append("|:---|:---|:---|")
        advice_markdown_parts.extend(soil_recommendations_table_rows)
        advice_markdown_parts.append("\n---")

    has_fertilizer_tips = crop_info.get("fertilizer_tips")
    has_notes = crop_info.get("notes")

    #Загальні рекомендації, добрива та особливості вирощування
    if has_fertilizer_tips or has_notes:
        advice_markdown_parts.append(f"## Загальні рекомендації")

    if has_fertilizer_tips:
        advice_markdown_parts.append(f"### Добрива")
        advice_markdown_parts.append(crop_info["fertilizer_tips"])
        advice_markdown_parts.append("\n")
```

```
    if has_notes:
        advice_markdown_parts.append(f"### Особливості вирощування")
        advice_markdown_parts.append(crop_info["notes"])
        advice_markdown_parts.append("\n")

    if not advice_markdown_parts:
            return f"Для культури '**{db_crop_name}**' наразі немає специфічних порад."

    return "\n".join(advice_markdown_parts)
```

ДОДАТОК Д

Код програмного засобу системи автоматизації аналізу ґрунту у VS CODE для машинного навчання

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
import joblib
import sys
import os

try:
    from analysis_logic import CROP_DATABASE as CROP_DATABASE_FULL
except ImportError:
    print("Помилка: не вдалося імпортувати CROP_DATABASE з analysis_logic.py. Використовується мінімальний набір даних.")
    CROP_DATABASE_FULL = {
                "Grains": {"Пшениця озима": {"N_ppm_optimal": (100, 180), "P_ppm_optimal": (40, 70), "K_ppm_optimal": (100, 200), "pH_optimal": (6.0, 7.5)}},
            "Vegetables": {"Томати (відкритий ґрунт)": {"N_ppm_optimal": (100, 150), "P_ppm_optimal": (60, 100), "K_ppm_optimal": (150, 250), "pH_optimal": (6.0, 6.8)}}
    }

def flatten_crop_db_for_training(db):
```

```
    flat_db = {}
    for category_data in db.values():
        for crop_name, params in category_data.items():
            flat_db[crop_name] = params
    return flat_db

flat_train_db_full = flatten_crop_db_for_training(CROP_DATABASE_FULL)

all_crop_names = list(flat_train_db_full.keys())
if not all_crop_names:
        print("Помилка: CROP_DATABASE не містить культур для навчання crop_encoder.")
    sys.exit()

crop_encoder = LabelEncoder()
crop_encoder.fit(all_crop_names)

parameters_to_train = {
    "N": {
        "optimal_key": "N_ppm_optimal",
        "current_key": "current_N_ppm",
            "categories": ["Висока потреба", "Середня потреба", "Низька потреба / Оптимум", "Надлишок / Потребує уваги", "Значний надлишок / Застереження"],
        "unit": "ppm"
    },
    "P": {
        "optimal_key": "P_ppm_optimal",
        "current_key": "current_P_ppm",
            "categories": ["Висока потреба", "Середня потреба", "Низька потреба / Оптимум", "Надлишок / Потребує уваги", "Значний надлишок / Застереження"],
```

```
    "unit": "ppm"
  },
  "K": {
    "optimal_key": "K_ppm_optimal",
    "current_key": "current_K_ppm",
      "categories": ["Висока потреба", "Середня потреба", "Низька потреба / Оптимум", "Надлишок / Потребує уваги", "Значний надлишок / Застереження"],
    "unit": "ppm"
  },
  "pH": {
    "optimal_key": "pH_optimal",
    "current_key": "current_pH",
        "categories": ["Дуже кислий / Корекція", "Кислий / Корекція", "Оптимальний", "Лужний / Корекція", "Дуже лужний / Корекція"],
    "unit": ""
  },
  "EC": {
    "optimal_key": "EC_optimal",
    "current_key": "current_EC",
    "categories": [
      "Дуже низька / Потребує корекції",
      "Низька / Можлива корекція",
      "Оптимальна",
      "Підвищена / Обережність",
      "Висока / Застереження"
    ],
    "unit": "µS/cm"
  }
}
```

```
models_dir = "ml_models"
os.makedirs(models_dir, exist_ok=True)
joblib.dump(crop_encoder, os.path.join(models_dir, "crop_encoder_shared.joblib"))
print(f"Спільний crop_encoder збережено. Класи: {list(crop_encoder.classes_)}")

for param_name, config in parameters_to_train.items():
    print(f"\n--- Тренування моделі для параметра: {param_name} ---")

    generated_data_param = []
    relevant_crops_count = 0
    for crop_name, params_db in flat_train_db_full.items():
        if config["optimal_key"] in params_db and \
            isinstance(params_db[config["optimal_key"]], tuple) and \
            len(params_db[config["optimal_key"]]) == 2:

            relevant_crops_count +=1
            opt_min, opt_max = params_db[config["optimal_key"]]

            if param_name in ["N", "P", "K"]:
                scenarios = [
                    (opt_min * 0.5, config["categories"][0]),
                    (opt_min * 0.8, config["categories"][1]),
                    ((opt_min + opt_max) / 2, config["categories"][2]),
                    (opt_max * 1.2, config["categories"][3]),
                    (opt_max * 1.5, config["categories"][4])
                ]

            if param_name == "EC":
                scenarios = [
                (opt_min * 0.3, config["categories"][0]),
```

```
            (opt_min * 0.7, config["categories"][1]),
            ((opt_min + opt_max) / 2, config["categories"][2]),
            (opt_max * 1.3, config["categories"][3]),
            (opt_max * 2.0, config["categories"][4])
            ]

        elif param_name == "pH":
            delta = (opt_max - opt_min) / 4
            scenarios = [
                (opt_min - 2 * delta, config["categories"][0]),
                (opt_min - delta, config["categories"][1]),
                ((opt_min + opt_max) / 2, config["categories"][2]),
                (opt_max + delta, config["categories"][3]),
                (opt_max + 2 * delta, config["categories"][4])
            ]
        else:
            continue

        for current_val, category in scenarios:
            generated_data_param.append({
                "crop_name": crop_name,
                config["current_key"]: float(current_val),
                f"{param_name}_optimal_min": float(opt_min),
                f"{param_name}_optimal_max": float(opt_max),
                f"{param_name}_need_category": category
            })

if not generated_data_param:
```

```
            print(f"Помилка: Не згенеровано даних для тренування моделі {param_name}. Перевірте CROP_DATABASE та наявність '{config['optimal_key']}'.")
        continue

    print(f"Знайдено {relevant_crops_count} релевантних культур для параметра {param_name}.")
    df_param = pd.DataFrame(generated_data_param)

                                        df_param["crop_name_encoded"] = crop_encoder.transform(df_param["crop_name"])

    param_need_encoder = LabelEncoder()
                            df_param[f"{param_name}_need_category_encoded"] = param_need_encoder.fit_transform(df_param[f"{param_name}_need_category"])

            X = df_param[["crop_name_encoded", config["current_key"], f"{param_name}_optimal_min", f"{param_name}_optimal_max"]]
    y = df_param[f"{param_name}_need_category_encoded"]

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y if len(np.unique(y)) > 1 else None)

      model_param = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
    model_param.fit(X_train, y_train)

    if not X_test.empty:
        y_pred_test = model_param.predict(X_test)
```

```
            print(f"Точність {param_name}-моделі на тестовій вибірці: {accuracy_score(y_test, y_pred_test):.4f}")

    unique_labels_test = np.unique(y_test)
            report_target_names = [param_need_encoder.classes_[i] for i in unique_labels_test if i < len(param_need_encoder.classes_)]
            report_labels = [i for i in unique_labels_test if i < len(param_need_encoder.classes_)]

    if report_target_names and report_labels:
        try:
            print(f"Звіт по класифікації для {param_name}-моделі:\n",
                  classification_report(y_test, y_pred_test, labels=report_labels, target_names=report_target_names, zero_division=0))
        except ValueError as e:
            print(f"Помилка при генерації звіту класифікації для {param_name}: {e}")
    else:
        print(f"Недостатньо класів у тестовій вибірці для {param_name} для повного звіту.")
  else:
    print(f"Тестова вибірка для {param_name} порожня.")

                joblib.dump(model_param, os.path.join(models_dir, f"{param_name.lower()}_model.joblib"))
                joblib.dump(param_need_encoder, os.path.join(models_dir, f"{param_name.lower()}_need_encoder.joblib"))
  print(f"Модель та кодувальник для {param_name} збережено.")

print("\n--- Процес тренування моделей завершено. ---")
```

ДОДАТОК Е

Демонстраційний матеріал

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК. 411710.007 ПЗ	Пояснювальна записка	А4, 83 с.
		Додаткові матеріали	
2		Апробація результатів роботи	А4, 8 с.
3		Програмний код підключення датчика до мікроконтролера та передача даних у BLYNK	А4, 5 с.
4		Основний код програмного засобу для аналізу ґрунту у VS CODE	А4, 21 с.
5		Код логіки аналізу програмного засобу для аналізу ґрунту у VS CODE	А4, 22 с.
6		Код програмного засобу для аналізу ґрунту у VS CODE для машинного навчання	А4, 7 с.
7		Демонстраційний матеріал	А4, 13 с.

					ГЮИК. 411710.007 ВД	
Змін.	Арк.	Номер докум.	Підпис	Дата		
Розроб.		Надьожкіна			Розроблення системи автоматизації аналізу ґрунту на базі технології Інтернету речей	Літера: Н; Аркуш: 1; Аркушів: 1
Перевір.		Хрустальова				
						Кафедра КІТАР ХНУРЕ
Н.контр.		Стародубцев				
Затв.		Невлюдов				